Université de Montréal

Continuous Coordination As a Realistic Scenario for Lifelong Learning

par

Akilesh Badrinaaraayanan

Département d'informatique et de recherche opérationnelle
Faculté des arts et des sciences

Mémoire présenté en vue de l'obtention du grade de
Maître ès sciences (M.Sc.)
en informatique

April 30, 2021

Université de Montréal

Faculté des arts et des sciences

Ce mémoire intitulé

Continuous Coordination As a Realistic Scenario for Lifelong Learning

présenté par

Akilesh Badrinaaraayanan

a été évalué par un jury composé des personnes suivantes :

Liam Paull
(président-rapporteur)

Aaron Courville
(directeur de recherche)

Sarath Chandar Anbil Parthipan
(codirecteur)

Pierre Bellec
(membre du jury)

Résumé

Les algorithmes actuels d'apprentissage profond par renforcement (RL) sont encore très spécifiques à leur tâche et n'ont pas la capacité de généraliser à de nouveaux environnements. L'apprentissage tout au long de la vie (LLL), cependant, vise à résoudre plusieurs tâches de manière séquentielle en transférant et en utilisant efficacement les connaissances entre les tâches. Malgré un regain d'intérêt pour le RL tout au long de la vie ces dernières années, l'absence d'un banc de test réaliste rend difficile une évaluation robuste des algorithmes d'apprentissage tout au long de la vie. Le RL multi-agents (MARL), d'autre part, peut être considérée comme un scénario naturel pour le RL tout au long de la vie en raison de sa non-stationnarité inhérente, puisque les politiques des agents changent avec le temps. Dans cette thèse, nous présentons un banc de test multi-agents d'apprentissage tout au long de la vie qui prend en charge un paramétrage à la fois zéro et quelques-coups. Notre configuration est basée sur Hanabi - un jeu multi-agents partiellement observable et entièrement coopératif qui s'est avéré difficile pour la coordination zéro coup. Son vaste espace stratégique en fait un environnement souhaitable pour les tâches RL tout au long de la vie. Nous évaluons plusieurs méthodes MARL récentes et comparons des algorithmes d'apprentissage tout au long de la vie de pointe dans des régimes de mémoire et de calcul limités pour faire la lumière sur leurs forces et leurs faiblesses. Ce paradigme d'apprentissage continu nous fournit également une manière pragmatique d'aller au-delà de la formation centralisée qui est le protocole de formation le plus couramment utilisé dans MARL. Nous montrons empiriquement que les agents entraînés dans notre environnement sont capables de bien se coordonner avec des agents inconnus, sans aucune hypothèse supplémentaire faite par des travaux précédents.
Mots-clés: le RL multi-agents, l'apprentissage tout au long de la vie.

Abstract

Current deep reinforcement learning (RL) algorithms are still highly task-specific and lack the ability to generalize to new environments. Lifelong learning (LLL), however, aims at solving multiple tasks sequentially by efficiently transferring and using knowledge between tasks. Despite a surge of interest in lifelong RL in recent years, the lack of a realistic testbed makes robust evaluation of lifelong learning algorithms difficult. Multi-agent RL (MARL), on the other hand, can be seen as a natural scenario for lifelong RL due to its inherent non-stationarity, since the agents' policies change over time. In this thesis, we introduce a multi-agent lifelong learning testbed that supports both zero-shot and few-shot settings. Our setup is based on Hanabi — a partially-observable, fully cooperative multi-agent game that has been shown to be challenging for zero-shot coordination. Its large strategy space makes it a desirable environment for lifelong RL tasks. We evaluate several recent MARL methods, and benchmark state-of-the-art lifelong learning algorithms in limited memory and computation regimes to shed light on their strengths and weaknesses. This continual learning paradigm also provides us with a pragmatic way of going beyond centralized training which is the most commonly used training protocol in MARL. We empirically show that the agents trained in our setup are able to coordinate well with unknown agents, without any additional assumptions made by previous works. **Key words**: multi-agent reinforcement learning, lifelong learning.

Contents

List of tables

List of figures

List of acronyms and abbreviations

LLL	Lifelong Learning
RL	Reinforcement Learning
MARL	Multi-agent Reinforcement Learning
MDP	Markov Decision Process
Dec-POMDP	Decentralized Partially Observed Markov Decision Process
SAD	Simplified Action Decoder
VDN	Value Decomposition Network
AUX	Auxiliary Task
SP	Self Play
OP	Other-Play
CP	Cross-Play
SGD	Stochastic gradient descent
ER	Experience Replay
A-GEM	Averaged Gradient Episodic Memory
EWC	Elastic Weight Consolidation
MTL	Multi-task Learning

Acknowledgement

I thank my advisors Prof. Aaron Courville and Prof. Sarath Chandar for their excellent guidance throughout my master's journey at the Université de Montréal/Mila. I am fortunate to benefit from their knowledge and ideas surrounding my research. Apart from being great researchers themselves, they have good human values.

I am grateful to Hadi Nekoei for being an active contributor to the work presented in this thesis and for being there through the highs and lows of this project, the countless hours working and discussing things related to this project will remain vivid in my memory.

I also thank Olexa Bilaniuk for his help in addressing the engineering issues related to our code/clusters; Sai Krishna, Pravish Sainath, and Eeshan Dhekane for providing feedback of my thesis; Gabriele Prato for help with the French translation of the abstract.

I thank Linda Peinthière of Mila and Céline Bégin of Université de Montréal for their patience and help in all the administrative matters associated with my studies and research.

I am indebted to my undergraduate advisor Dr. Vineeth N Balasubramanian, IIT-Hyderabad for introducing me to machine learning and research, highly encouraging me to go for graduate studies.

Finally, I thank my parents Chandrika and Badrinaaraayanan for all their unconditional support, encouragement, and help throughout my life. They set high standards for me right from a young age and always prioritized my education. I dedicate my efforts and achievements to them, for without them, it would have been impossible.

Dedicated to my parents Chandrika and Badrinaaraayanan.

Chapter 1

Introduction

Recent advances in deep learning has revolutionized many fields in artificial intelligence (AI) such as computer vision (Krizhevsky et al., 2017, Ren et al., 2016), natural language understanding (Sutskever et al., 2014, Vaswani et al., 2017), reinforcement learning (RL) (Silver et al., 2017), and speech processing (Hinton et al., 2012). In particular, deep RL has shown an immense potential to achieve superhuman performance (Mnih et al., 2013, Silver et al., 2018) on some narrow and well-defined tasks. In contrast, humans can quickly and continually learn new tasks while maintaining the skills to solve previously learned tasks. The ability of an AI system to effectively update new information over time is known as lifelong learning (LLL) or continual learning, and one can postulate this as one of the fundamental ingredients of general AI. Balancing between learning from recent experiences while not forgetting the knowledge acquired from the past is a well-studied problem known as the stability-plasticity dilemma (Carpenter and Grossberg, 1987). Catastrophic forgetting is a phenomenon in which training the model with new information obstructs previously learned knowledge. This is a common failure case in training neural networks to adapt to new tasks or learning from non-stationary data streams (i.e. non-iid) (McCloskey and Cohen, 1989). Alleviating catastrophic-forgetting is crucial to enable real-world applications where input distributions can shift and where retraining on past data, or from scratch is infeasible. While lifelong learning has been identified as an important and challenging problem decades ago (Ring, 1998, Thrun, 1998), it has recently seen a surge of interest (Aljundi et al., 2018, Chaudhry et al., 2018, 2019, Kirkpatrick et al., 2017, Lopez-Paz and Ranzato, 2017) with the success of deep learning.

Several standard benchmarks have been proposed to evaluate novel lifelong learning approaches, mostly for supervised learning settings such as Permuted MNIST (Goodfellow et al., 2013), Split MNIST/CUB/CIFAR (Chaudhry et al., 2018, Zenke et al., 2017). One fundamental issue with using datasets like MNIST as a source of data is the lack of resulting task complexity especially with the large capacity of modern neural networks. Another issue with most current lifelong learning benchmarks is that the relation between tasks cannot be quantified easily. Consequently, most of the evaluation efforts have focused mainly on mitigating catastrophic forgetting, while an ideal lifelong learning system should in addition measure forward and backward transfer. Some recent works have shown limitations of lifelong learning benchmarks (Antoniou et al., 2020, Roady et al., 2020). For instance, it has been shown that after continual training, the performance of a model trained from scratch using only samples from the episodic memory at test-time, is comparable to specifically designed lifelong learning solutions for most of these benchmarks (Prabhu et al., 2020). There have been efforts to address this by proposing more challenging benchmarks like CORe50 (Lomonaco and Maltoni, 2017), CRIB (Stojanov et al., 2019), OpenLoris (Shi et al., 2020), Stream51 (Roady et al., 2020), and IIRC (Abdelsalam et al., 2020).

RL can be a natural fit for studying lifelong learning as it provides an agent-environment interaction paradigm wherein the agent is exposed to non-stationary streams of data (Kaplanis et al., 2018, 2019). However, there is a dearth of well-established benchmarks to study progress in lifelong RL. Most of these benchmarks are hand-engineered customization to the standard RL environments (Bellemare et al., 2013, Brockman et al., 2016) adding synthetic non-stationarity to the environments (Al-Shedivat et al., 2017, Henderson et al., 2017) or ordering some completely unrelated environments in a sequence (Xu et al., 2020) to facilitate the evaluation of lifelong learning performance (eg. a random sequence of Atari used in (Kirkpatrick et al., 2017)). Designing overly-tailored experiments for a specific lifelong RL problem can entail unwanted bias as shown in (Khetarpal et al., 2020). For instance, one issue appears as inherent determinism in environments such as Arcade Learning Environment (Bellemare et al., 2013), where the problem is addressed by blindly saving state-action sequence which defies the generalization purposes (Machado et al., 2018). Zintgraf et al. (2019) also showed that adaptation steps needed in some few-shot learning tasks are small and confirmed that task inference and multi-task learning is sufficient to do well in some cases.

In this thesis, we propose a new lifelong RL setup based on Hanabi (Bard et al., 2020) called *Lifelong Hanabi*. Hanabi is a partially-observable, fully cooperative multi-agent game that consists of 2-5 players. In our setup, one agent (*learner*) is trained sequentially with a set of partners (tasks). The *learner* and its partners are sampled from a large pool of pre-trained agents (≥ 100). The pre-trained pool consists of agents trained with different MARL methods such as Independent Q-learning (IQL) (Tan, 1993), Value Decomposition Networks (VDN) (Sunehag et al., 2017), Simplified Action Decoder (SAD) (Hu and Foerster, 2019), Other-Play (OP) (Hu et al., 2020) with different architectures and seeds for each method that have shown good performance in Hanabi. Bard et al. (2020) show that agents trained even with the same MARL method but different seeds do not learn to cooperate in the zero-shot scenario, thereby suggesting that these agents converge to different strategies. This large strategy space of Hanabi makes it an ideal scenario for lifelong learning. How far apart the agents are in the strategy space can be measured through the *cross-play* (CP) matrix (Bard et al., 2020) that contains the gameplay scores obtained by pairing the agents with one another. Cross-play scores can be used as a proxy for task similarity to design tasks in *Lifelong Hanabi*.

1.1. Contributions

The main contributions of this thesis are based on our paper **"Continuous Coordination As a Realistic Scenario for Continual Learning"** (Nekoei et al., 2021) (code link), authored by Akilesh Badrinaaraayanan, Hadi Nekoei, Aaron Courville, and Sarath Chandar, accepted at **ICML 2021** (also accepted as a **spotlight** at ICLR 2021 workshop - A Roadmap to Never-Ending Reinforcement Learning) and are summarized below:

- We propose a new lifelong reinforcement learning benchmark that has the following desirable properties:
  - It is challenging for state-of-the-art (SOTA) lifelong learning algorithms.
  - It is straightforward to quantify the relation between tasks through the CP matrix.

  - It is easily extendable to long sequences of diverse tasks without any synthetic modifications.

- We evaluate recent lifelong learning algorithms on this benchmark in limited memory and computation regimes and highlight their strengths and limitations.
- We obtain comparable performance on zero-shot coordination in Hanabi even when coordinating with agents trained with MARL methods different from that of the *learner*, without any additional assumptions such as exploiting handcrafted symmetries (Hu et al., 2020) or having access to other agent's greedy action or policy (Hu and Foerster, 2019).

The rest of the thesis is organized as follows: in chapter 2 we present background on machine learning and reinforcement learning, presenting specific related topics on lifelong learning and multi-agent reinforcement learning in chapter 3. We present *Lifelong Hanabi* setup in chapter 4, benchmark several recent lifelong learning algorithms in chapter 5 and present conclusions as well as future work in chapter 6.

1.1.1. Author contributions for the paper

- Hadi Nekoei and Sarath Chandar were exploring ideas of decentralized training methods for MARL considering various multi-agent environments like Checkers, MARL grid, Hanabi, etc.
- Their initial exploration around Hanabi as well as (Bard et al., 2020) work (in which Sarath Chandar was one of the main contributors) suggested large strategy space in Hanabi.
- I started working on this project when we had an initial idea to propose this as a benchmark for lifelong learning as well as an alternate training paradigm for MARL.
- The code for all the algorithms and experiment setup for this paper were designed and written by myself and Hadi Nekoei with valuable feedback from Sarath Chandar.
- All the experiments (pre-training, lifelong learning, and testing) and engineering issues with our code/Mila cluster/Compute Canada clusters were done by me with help from Hadi when relevant.
- Collection of results, organizing, and presenting them were done by myself and Hadi Nekoei.

- The entire paper was written by myself and Hadi Nekoei with feedback from Sarath Chandar and Aaron Courville.
- Aaron Courville also provided periodic feedback in steering the work in the right direction.

Chapter 2

Machine Learning and Reinforcement Learning

Learning is a fundamental characteristic of a human that can be defined as a process of acquiring knowledge and skills through experience. As we live in an era where a vast amount of data is being collected and stored every day through user presence in social media, video streaming platforms, etc.; daily interaction with Machine Learning (ML) algorithms has become inevitable.

2.1. Machine Learning basics

Mitchell (1997) defines a learning algorithm as "A computer program is said to learn from experience E with respect to some set of tasks T and performance measure P, if its performance at tasks T, as measured by P, improves with experience E". For example in the task of classifying emails as spam or not, the task is the binary classification of emails and the performance measure is accuracy i.e. what percentage of emails the ML algorithm classifies correctly. ML algorithms can be broadly classified as supervised, unsupervised, and reinforcement learning (RL) depending on the type of experience the learning algorithm has during the process. Supervised ML algorithms usually learn from labeled data and adapt in a data-driven manner.

A dataset D is a collection of n data points $\left\{\mathbf{X}^{(i)}\right\}_{i=1}^{n}$. If labeled, each data point is a tuple containing inputs $\mathbf{x}^{(i)}$ and labels $y^{(i)}$. In case of unlabeled dataset, the data point just contains inputs $\mathbf{x}^{(i)}$. A supervised learning algorithm learns a predictive function

$f_\theta : \mathcal{X} \longrightarrow \mathcal{Y}$ using a given labeled dataset $D = \left\{\mathbf{x}^{(i)}, y^{(i)}\right\}$, which when provided with similar but unseen input $\bar{\mathbf{x}} \in \mathcal{X}$, potentially correctly predicts $\bar{y} \in \mathcal{Y}$ as $f(\bar{x})$. **Supervised learning** algorithms can be used to predict discrete values corresponding to the classes of the input $\mathbf{x}$, i.e. $\left(y^{(i)} \in \mathcal{Y}, \text{ with } |\mathcal{Y}| = n \in \mathbb{N}\right)$, or values over a continuous interval $\left(y^{(i)} \in \mathcal{Y} \subseteq \mathbb{R}\right)$. The former problem is the **classification** problem and the latter **regression**. Some examples of supervised learning algorithms include linear regression for regression; random forests, decision trees for classification; regression and support vector machines for classification. Supervised learning requires curating large amounts of labeled datasets, therefore can turn out to be very laborious.

Unsupervised learning can learn from unlabeled datasets to discover the underlying structure or distribution in the data. Formally, an unsupervised learning algorithm attempts to learn the latent representation $\mathbf{z} \in \mathcal{Z}$ for the input $\mathbf{x} \in \mathcal{X}$, where $\mathcal{Z}$ represents the space of latent representation. Some examples of unsupervised learning problems include clustering that tries to find groups among data and dimensionality reduction which transforms the data from a high-dimensional space to a low-dimensional space with the hope that the low-dimensional representation retains some meaningful properties of the original data.

Semi-supervised learning utilizes both labeled and unlabeled data, trying to leverage the best of both worlds i.e. unsupervised learning techniques to discover and learn the structure in input while supervised learning techniques can be used to guess the labels for the unlabeled data which can then again be fed back to the supervised learning method as training data.

Self-supervised learning is a form of unsupervised learning where the data itself provides the supervision. This is generally done by withholding some part of the data and tasking the network to predict it. The task defines a proxy loss, and the network is forced to learn what we care about to solve it e.g. a semantic representation.

Reinforcement Learning (RL) is another paradigm in which an agent interacts with the environment to maximize its cumulative reward, thereby having a feedback loop between the learning system and its experiences. In this thesis, we mainly focus on this paradigm.

[1]https://docs.aws.amazon.com/machine-learning/latest/dg/model-fit-underfitting-vs-overfitting.html

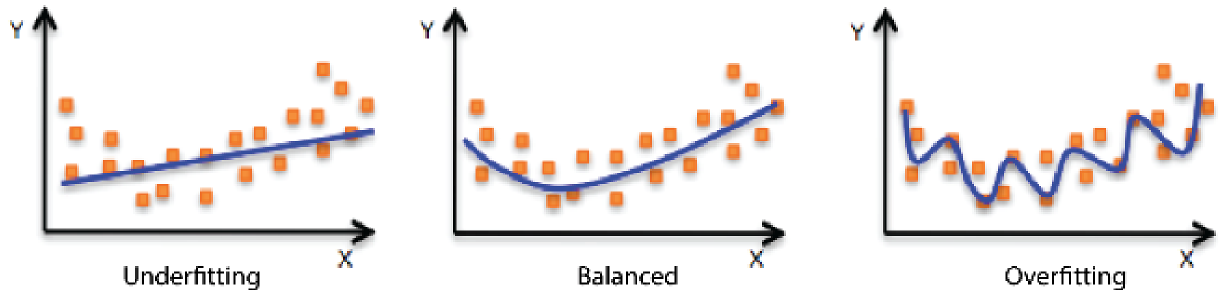


Fig. 2.1. Underfitting and Overfitting. Left: Data points being approximated by a straight line (underfitting); Right: Fitted curve passes through most data points (overfitting); Middle: Fitted curve is balanced having somewhat ideal capacity. Image source: reproduced from AWS Machine Learning.[1]

2.1.1. Overfitting, Underfitting and Capacity

The fundamental challenge in ML is that any good algorithm should perform well on novel, unseen data, an ability called **generalization**. We want the **generalization error** (test error) to be low as well, in addition to the **training error**, where generalization error is defined as the expected value of the error on a new input. Under the **i.i.d** assumptions i.e. the examples in a dataset are **independent** of each other, and that the training and test sets are **identically** distributed (i.e. drawn from the same probability distribution), we can mathematically study the relationship between train error and generalization error (Goodfellow et al., 2016). **Underfitting** refers to the case when the model is not able to obtain a sufficiently low error value on the training set. **Overfitting** occurs when the gap between training and generalization errors is significantly large. Figure 2.1 shows an example of underfitting, overfitting, and balanced model on a sample dataset. As we can observe, the underfitted curve is a straight line that does not correspond to the data distribution, while the overfitted curve passes through most of the individual points. The **capacity** of a model refers to its ability to fit a wide range of functions. A low capacity model might struggle to fit the training set well, whereas a high capacity model can potentially memorize the training set thereby overfitting and perform poorly on the test set. The relationship between capacity and error is given in Figure 2.2. The left side of the curve corresponds to the underfitting regime (low capacity), while the right part of the curve corresponds to overfitting (high capacity) as the generalization gap increases.

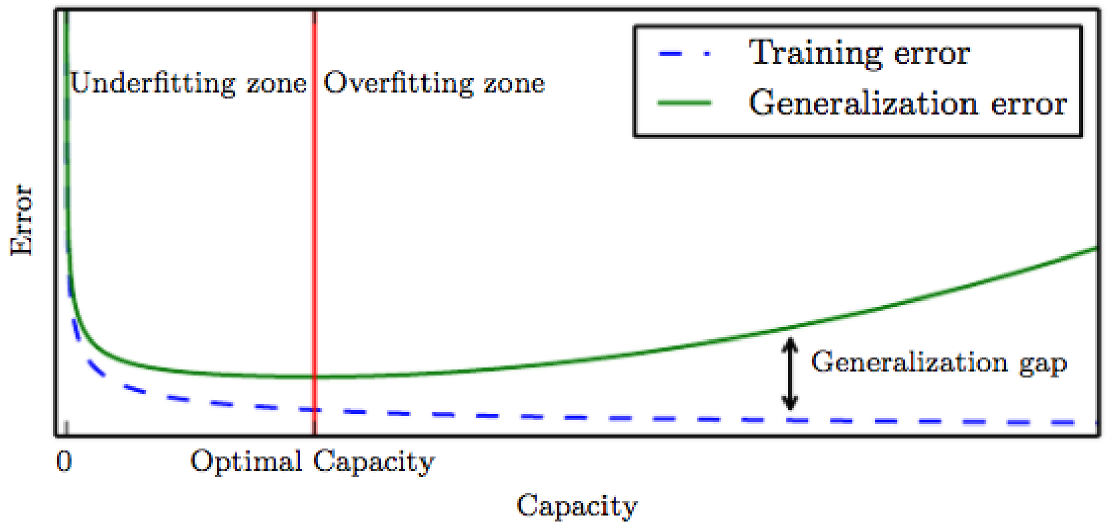


Fig. 2.2. Typical relationship between capacity and error. In the underfitting zone, the generalization error has not reached its lowest yet, while in the overfitting zone, the generalization error starts to increase steadily again while the training error continues to decrease. Image source: reproduced from the Deep Learning book (Goodfellow et al., 2016).

Parameters of an ML model (typically denoted by θ) are those that are estimated or learned from the training data and are crucial for making predictions. **Hyper-parameters** are tunable knobs of a model that are usually set manually to help learn better parameters, although there are frameworks such as Hyperopt (Bergstra et al., 2013) that could help in hyperparameter optimization. Hyperparameters have an effect in controlling the capacity of a model. For eg. in linear regression, the degree of the polynomial is a hyperparameter that controls its capacity.

Loss function is the objective function that we want to minimize (denote by $L(\theta)$). It tries to represent the different aspects of a complex model onto a single scalar value. Most commonly, in the case of classification, a parametric model defines a distribution, and using the principle of **maximum likelihood** one can define the loss function as the cross-entropy between the training data and the model's predictions. In the case of regression problems, mean squared error is commonly used. Assuming differentiability of loss function, **gradient descent** is most commonly used in ML to minimize the loss function $L(\theta)$ of the model parameterized by $\theta \in \mathbb{R}^d$ by updating its parameters in the opposite direction of the gradient of the loss function w.r.t to its parameters $\nabla_\theta L(\theta)$. The **learning rate** η determines the size of the steps we take in the opposite direction of the gradients hoping to reach a local minimum under ideal conditions. Gradient descent is typically employed in three common modes:

- Full batch or batch; $\theta = \theta - \eta.\nabla_{\theta} L(\theta)$
- Stochastic; $\theta = \theta - \eta.\nabla_{\theta} L(\theta; x^{(i)}; y^{(i)})$
- Mini-batch; $\theta = \theta - \eta.\nabla_{\theta} L(\theta; x^{(i:i+n)}; y^{(i:i+n)})$

In batch gradient descent, the complete dataset is used for computing gradient, and then the parameters are updated. Batch gradient descent is guaranteed to converge to the global minimum for convex error functions and to a local minimum for non-convex functions. In stochastic gradient descent, parameter update is done after every example. It can potentially cause the loss function to fluctuate due to frequent updates having high variance. Mini-batch gradient descent takes the best of both worlds and performs an update after every mini-batch consisting of n training examples. In this way, it reduces the variance of parameter updates and also makes use of matrix operations-based optimizations that most deep learning libraries provide.

ML models are usually prone to overfitting the training data, leading to poor generalization performance. **Regularization** is a common technique employed in ML models to improve its generalization performance, which as discussed previously is a key criterion in designing ML algorithms and for achieving success in the real-world. These methods are usually imposed as additional constraints on the loss function thereby penalizing the model (i.e. the loss function is now $L(\theta) + R(\theta)$ where $R(\theta)$ is the regularization term). $\mathcal{L}^1$ and $\mathcal{L}^2$ regularization are two commonly used methods (Goodfellow et al., 2016). $\mathcal{L}^1$ regularization ($R(\theta) = \|\theta\|_1$) involves adding the sum of absolute value of the parameters and encourages learning of sparse parameters. $\mathcal{L}^2$ regularization (a.k.a weight decay $\frac{1}{2}\|\theta\|_2^2$) encourages the parameters to be closer to the origin. Regularization techniques are crucial to prevent overfitting and controlling the capacity of the model. **Early stopping** is another commonly-used technique for preventing overfitting, where we monitor the losses on both train and validation sets and stop the training process when the loss on the validation set starts increasing while training loss continues to decrease. **Data augmentation** is also a widely used technique which is to synthetically enlarge the size of the dataset through flipping, rotation, scaling, translation, etc. (such that the labels remain unchanged) although choosing the right augmentation strategy could be a daunting task. Other approaches to prevent overfitting include injecting specific **noise** to inputs or gradients (Neelakantan et al., 2015), **dropout** (Srivastava et al., 2014), **ensembling** techniques, etc. Regularization methods can be broadly classified into data-dependent and data-independent regularizers (Guo et al., 2019). Data-dependent regularizers exploit the underlying structure of the data to constrain the parameter space. Some examples are data-augmentation methods and regularizers like mixup (Zhang et al., 2017), manifold mixup (Verma et al., 2019), patchup (Faramarzi et al., 2020), etc. that work either in the input or the latent space. Data-independent methods impose constraints on the parameters without exploiting the underlying structure of data. Some examples include weight-decay, dropout, dropblock (Ghiasi et al., 2018), etc.

2.1.2. Neural Networks

Deep Learning has seen revolutionary performance across various domains in the last decade. Fundamentally, they are neural networks having different structures and architectures.

Neural networks can consist of one or more *layers* of trainable **neurons**, where the outputs of the current layer are the inputs to the next layer. Neurons are fundamental computational units partly inspired from biological neurons, that are capable of *learning* based on *feedback*. The neurons have trainable parameters and the outputs of one layer are passed through a *non-linear* activation layer, before being fed as input to the next layer. The outputs of the neural network are computed based on the inputs and are compared against the ground-truth values to compute the *loss*. The trainable parameters are updated based on the loss function using the **backpropagation algorithm**. The gradients are propagated from the last layer to the first layer sequentially using the **chain rule**. The neural networks are typically end-to-end *differentiable* (if the activation function is differentiable too) and hence are capable of *end-to-end learning*. **Deep neural networks** are neural networks with many layers, having novel layers and techniques proposed for better learning custom to the input data type (through better gradient propagation, higher capacity, etc.). Chain rule allows for the reuse of gradients computed for parameter updates in lower layers of the neural network effectively.

Feed-forward neural networks: In feed-forward neural networks, every neuron in one layer has directed connections to neurons in the subsequent layer, i.e. $a^{(l+1)} = \sigma.\Big(W^{(l)}.a^{(l)} + b^{(l)}\Big)$; where $a^l \in \mathbb{R}^{d^l}$, where d^l is the dimensionality of layer l, $a^{l+1} \in \mathbb{R}^{d^{(l+1)}}$, where $d^{(l+1)}$ is the dimensionality of layer $l+1$, $W^{(l)} \in \mathbb{R}^{d^{(l+1)} \times d^l}$ is the weight matrix, $b^{(l)} \in \mathbb{R}^{d^{(l+1)}}$ is the bias. **Non-linearities** or *activation functions* are crucial for neural networks to learn many different function families. *Activation functions* commonly used in neural networks are sigmoid, ReLU, tanh, etc. (Goodfellow et al., 2016) (equation 2.1.1). Typically, every neuron in one layer is connected to all the neurons in the subsequent layer and the neurons in a single layer function completely independently (do not share any connections). This is called a *fully-connected* (FC) layer.

$$\text{For } x \in \mathbb{R}, \quad \begin{array}{ll} \text{sigmoid}\,(x) = \frac{1}{1+e^{-x}}, & \tanh\,(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}, \\ \text{ReLU}\,(x) = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases} & \text{ELU}\,(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha \cdot (e^x - 1) & \text{otherwise} \end{cases} \end{array} \tag{2.1.1}$$

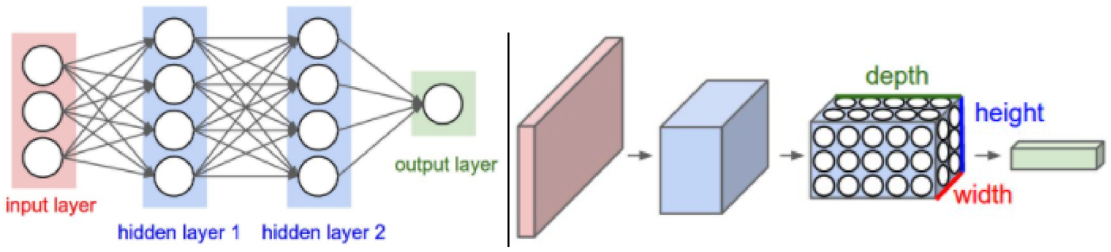


Fig. 2.3. Left - MLP; Right - CNNs. In CNNs, every neuron is only connected to a *local* region of the input volume. Image source: reproduced from Stanford CS231n [2]

Convolutional Neural Networks (CNNs) are similar to the feed-forward networks in that they are made up of neurons that have learnable weights and biases. Given an input image, they output class scores (for classification) through the softmax layer (that typically converts a vector having K real values into another having K real values that sum to 1), and loss is backpropagated to update the weights and biases. However, CNNs make an explicit assumption that they model image data, thereby encode certain structural priors. Images are very high-dimensional inputs, therefore flattening them to pass it through a feed-forward network introduces a large number of trainable parameters, which is computationally

[2]https://cs231n.github.io/convolutional-networks/

inefficient. For eg. for an input image size $256 \times 256 \times 3$, it has $256 * 256 * 3 = 196608$ weights. In addition, such flattening also makes the image lose its spatial information. Therefore, CNNs have neurons arranged in a 3D volume (*width*, *height* and *number of channels*). Every *Conv* layer transforms an input 3D volume into an output 3D volume and the final output layer has dimensions $1 \times 1 \times$ *number of classes*, i.e. a single vector of class scores arranged along the depth. Figure 2.3 illustrates this difference between CNNs and feed-forward networks. In CNNs, every neuron is only connected to a *local* region of the input volume. The spatial extent of this connectivity is called *filter-size* or *receptive-field* of the neuron. Each of the filters is shared by certain number of neurons. This is called *parameter sharing* and it's a reasonable assumption based on the rationale that if a particular filter is useful for detecting vertical edges at one location in the image, it should also be useful in detecting vertical edges at some other location as well. Convolutional networks typically consist of *convolutional*, *pooling* and *fully-connected* layers. Pooling operation will downsample the input volume along the spatial dimensions. The *Conv* and *FC* layers have parameters whereas *ReLU* and *Pool* layers do not.

Recurrent neural networks (RNNs) are typically used for modeling variable-length sequential data that have certain temporal structures. For eg. sentences are a sequence of words, videos are a sequence of images and audio is a sequence of sound patterns. Recurrent networks enable modeling the conditional distribution of a sequence i.e. $p_\theta\Big(x^{(t)}|x^{(t-1)}, \dots x^{(1)}\Big)$ by keeping track of hidden state $h^{(t)}$ that captures the context. At every time step t, vanilla RNNs use both input $x^{(t)}$ and $h^{(t-1)}$ to predict the output. Parameters in vanilla RNNs are shared across the different time steps. Vanilla RNNs are known to suffer from *vanishing* or *exploding* gradient problems and hence are not effective at learning temporal patterns over long sequence lengths. Several variants of RNNs have been proposed to tackle this problem primarily by introducing cell states and specific gates that selectively transmit information, most popular ones being Long Short Term Memory (LSTM) (Hochreiter and Schmidhuber, 1997), Gated Recurrent Unit (GRU) (Chung et al., 2014) and Non-saturating Recurrent Unit (NRU) (Chandar et al., 2019).

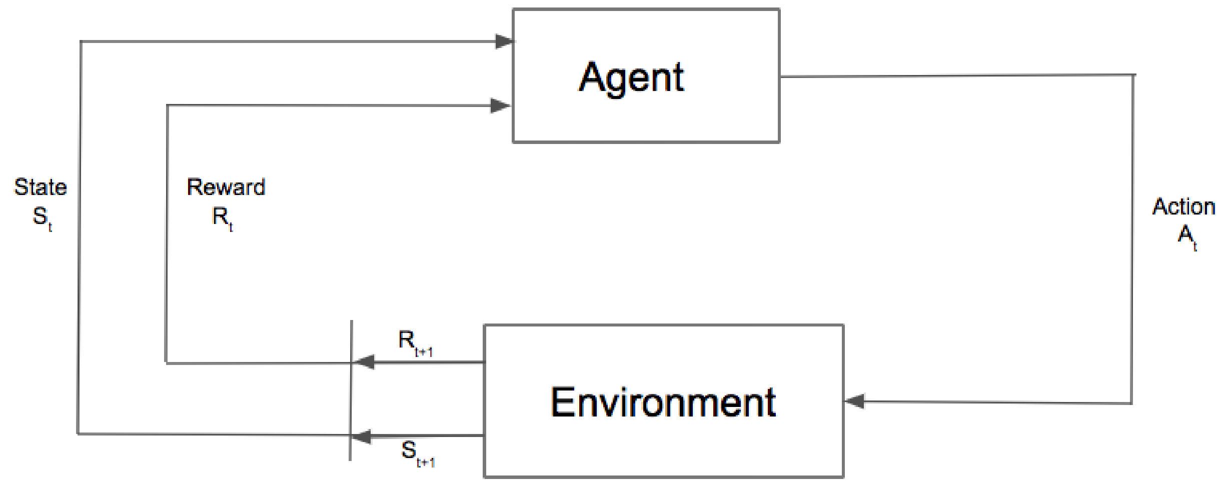


Fig. 2.4. Interaction of an agent with the environment. The agent takes action A_t based on state S_t, receives reward R_{t+1} and the state of the environment itself changes to S_{t+1}.

2.2. Reinforcement Learning

Reinforcement Learning (RL) studies the interaction of an agent in an environment (usually unknown) by attempting to learn the mapping of states to actions with the goal of the agent being to maximize its cumulative rewards. In contrast to supervised learning, the agent is not told what actions to take at each state, eg. a bot playing a game (say chess) to achieve high scores. Delayed reward signals and learning through trial-and-error are two aspects that make RL challenging. The RL problem is usually formulated using ideas borrowed from dynamical systems and control theory, especially from the optimal control of Markov Decision Processes (MDPs).

2.3. Markov Decision Process

Figure 2.4 shows an example of an **agent** interacting with an **environment**. The agent is the learner and everything outside the agent is the environment. The agent can be present in one of the many states ($s \in S$), takes action ($a \in A$) and the environment presents the agent with new states and rewards. How the environment reacts to certain actions is defined by a **model** which the agent may or may not have access to. The agent and the environment interact actively through a sequence of actions and rewards that are observed at time $t = 1,2,\ldots T$. At each time step t, the environment outputs an observation which represents the state $s_t \in S$, where S is the set of all possible states. The agent takes an action $a_t \in A$, where A is the set of all possible actions. Subsequently, the environment returns a numerical reward $r_{t+1} \in \mathbb{R}$, and the agent transitions to the next state s_{t+1}. This sequence produces a trajectory τ of one episode: $\tau = s_0 a_0 r_1 s_1 a_1 r_2 \ldots s_t$.

An observation o_t is a noisy version of the state. A fully observable environment is one in which the agent can observe the full state of the world. On the other hand, in a partially observable environment, the agent only observes $o_t \neq s_t$.

Most RL problems can be mathematically formulated as a Markov Decision Process (MDP), which lays the theoretical foundation through which some guarantees can be made. r_{t+1} and s_{t+1} are defined as random variables drawn from a probability distribution conditioned on the preceding state and action, $P\Big(s_{t+1}, r_{t+1} \mid s_t = s, a_t = a\Big)$. In a finite MDP, the sets of states and actions are finite. All states in an MDP has "Markov" property which means that the current state captures the information from the entire history, relevant to predict the next state and reward distribution, and hence:

$$P\left(s_{t+1} \mid s_t, s_{t-1}, \ldots, s_1\right) = P\left(s_{t+1} \mid s_t\right) \tag{2.3.1}$$

The initial state s_0 is sampled from start-state distribution ρ_0, i.e. $s_0 \sim \rho_0(.)$ An interaction between the agent and the environment at every time step t leads to a transition to a new state. The state transition function can be either stochastic, i.e. $s_{t+1} \sim P\left(. \mid s_t, a_t\right)$ or deterministic, i.e. $s_{t+1} = f(s_t, a_t)$, where f is a deterministic function.

In environments with a discrete action space A, there are a finite number of actions that the agent can take. On the other hand, in environments with continuous action space, actions are represented as real-valued vectors.

To summarize, an MDP consists of five elements $M = \langle S, A, P, R, \gamma \rangle$, where S is a set of states, A is a set of actions, P is a transition probability function, R is reward function and γ is discount factor for future rewards. In an unknown environment, we do not have perfect knowledge about P and R.

2.4. Return

The reward is the scalar value that the agent usually receives at every time step that is dependent on the current state, current action, and sometimes the next state (typically,

$r_t = R(s_t, a_t)$). The return is a cumulative function of the rewards obtained over a trajectory τ and is denoted $G(\tau)$. The horizon T is the number of time steps in the trajectory. The return for a finite horizon trajectory is $G(\tau) = \sum_{t=0}^{T-1} \gamma^t r_t$; whereas for an infinite horizon is $G(\tau) = \sum_{t=0}^{\infty} \gamma^t r_t$.

The discount factor $\gamma \in (0,1)$ is used to discount more recent rewards, γ close to 0 indicates a myopic and shortsighted agent that cares only about the immediate reward, whereas $\gamma = 1$ indicates that an agent assigns equal importance to rewards achieved both in immediate and farther future. In an infinite-horizon setting, the return may not have a finite value if $\gamma = 1$, hence discounting (i.e. $0 < \gamma < 1$) is required.

2.5. Policy

A policy π tells the agent what actions to take in any given state s_t. It is a mapping from state s_t to action a_t that can either be deterministic or stochastic. If the policy is deterministic, then the action a_t at any time step t is computed as $a_t = \mu_\theta(s_t)$, where μ is a deterministic function with parameters θ. For stochastic policies, the action a_t is sampled from a distribution conditioned on the state $a_t \sim \pi_\theta\,(. \mid s_t)$, where π is a conditional probability distribution parameterized by θ.

Without any function approximation, the policies would be represented as a big table with one row for every state containing the probability with which each action should be taken in that state. However, for large and/or continuous state and action spaces, it is not possible to maintain this explicit table, so function approximators are used. In such cases, the policy function can be modeled using different function approximators, most commonly a deep neural network. Changing the parameters of this policy function influences the agent's behavior. A policy π is better than policy π' if its expected return is greater than that of π' for all states. When using function approximation, stochastic policies are typically modeled as categorical distributions for discrete action spaces and diagonal Gaussian distributions for continuous action spaces. In order to apply these policies, we should be able to sample from the associated probability distributions.

The objective of RL agents is to learn a policy such that its expected return is maximized. If both the transition function and policy are stochastic, then the probability of a T-horizon trajectory is given by:

$$P(\tau) = \rho_0(S_0) \prod_{t=0}^{T-1} P(s_{t+1} \mid s_t, a_t) \, \pi(a_t \mid s_t) \tag{2.5.1}$$

which gives an expected return of:

$$J(\pi) = \int_{\tau} P(\tau) \, G(\tau) = \mathbb{E}_{\tau \sim \pi}[G(\tau)] \tag{2.5.2}$$

The optimal policy is given by the following equation:

$$\pi^* = \arg\max_{\pi} J(\pi). \tag{2.5.3}$$

2.6. Value functions

The value function measures how rewarding a state or an action is by predicting the expected future reward. The **return** (G_t) starting from time t is defined as:

$$G_t = R_{t+1} + \gamma R_{t+2} + \ldots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{2.6.1}$$

The state-value of a state s is the expected return if the agent is in this state at time t, $s_t = s$ and takes actions from a particular policy.

$$V_\pi(s) = \mathbb{E}_\pi[G_t \mid s_t = s] \tag{2.6.2}$$

The value of a terminal state, if it exists, is always zero. V_π is called the state-value function for any policy π.

The action-value ("Q-value") of a state-action pair is defined as:

$$Q_\pi(s,a) = \mathbb{E}_\pi[G_t \mid s_t = s, a_t = a] \tag{2.6.3}$$

which is the expected return starting at state s and taking an action a, followed by taking actions as per policy π. Q_π is the action-value function for policy π.

In addition, since we follow the target policy π, we can make use of the probability distribution over possible actions and the Q values to recover the state-value.

$$V_\pi(s) = \sum_{a \in A} Q_\pi(s, a) \pi(a \mid s) \tag{2.6.4}$$

The objective of an RL problem is to learn the policy that gets the agent the optimal expected return. Therefore, it is useful to estimate the value of each state under the optimal policy, known as the optimal value function.

$$V_*(s) = \max_{\pi} V_\pi(s) \tag{2.6.5}$$

which is the expected value of starting at state s and always acting according to the optimal policy. The optimal action-value function is defined as:

$$Q_*(s,a) = \max_{\pi} Q_\pi(s,a) \tag{2.6.6}$$

which is the expected value of starting at state s and take an action a, and taking actions from the optimal policy thereafter.

These value functions are related through the following relationships

$$V_\pi(s) = \mathbb{E}_{a\sim\pi}[Q_\pi(s,a)] \tag{2.6.7}$$

$$V_*(s) = \max_{a} Q_*(s,a) \tag{2.6.8}$$

The optimal value functions and optimal policy are also related as follows:

$$\pi_*(s) = \arg\max_{\pi} Q_\pi(s,a) \tag{2.6.9}$$

$$\pi_*(s) = \arg\max_{\pi} V_\pi(s) \tag{2.6.10}$$

The advantage function measures how much favorable is it to take one action relative to other actions on average (or the “advantage” of taking that action):

$$A_\pi(s,a) = Q_\pi(s,a) - V_\pi(s) \tag{2.6.11}$$

It is often used in policy gradient methods (see Section 2.8).

2.7. Bellman equations

Value estimation is one of the main methods for solving an RL problem. One way to estimate the value of a state is by averaging the returns we get from that state. This method is known as *Monte Carlo* (MC) estimation. However, by using the Markov property, one can obtain value function estimates that have lower variance. This is achieved by expanding the value function equation as below:

$$\begin{aligned} V_\pi(s) &= \mathbb{E}_\pi[G_t \mid s_t = s] && (2.7.1) \\ &= \mathbb{E}_\pi[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots \mid s_t = s] && (2.7.2) \\ &= \mathbb{E}_\pi[r_{t+1} + \gamma(r_{t+2} + \gamma r_{t+3} + \ldots) \mid s_t = s] && (2.7.3) \\ &= \mathbb{E}_\pi[r_{t+1} + \gamma G_{t+1} \mid s_t = s] && (2.7.4) \\ &= \mathbb{E}_\pi[r_{t+1} + \gamma V_\pi(s_{t+1}) \mid s_t = s] && (2.7.5) \end{aligned}$$

where r_{t+1} is the expected reward from state s_t and G_{t+1} is the return starting from state $s_{t+1} = s'$. This expression above is the *Bellman equation* for V_π (Bellman, 1956). The Bellman equation decomposes the value function into its immediate reward plus the discounted future value. $Q_\pi(s,a)$ obeys a similar equation, conditioning on (s,a).

$$\begin{aligned} Q_\pi(s,a) &= \mathbb{E}_\pi[r_{t+1} + \gamma V_\pi(s_{t+1}) \mid s_t = s, a_t = a] && (2.7.6) \\ &= \mathbb{E}_\pi[r_{t+1} + \gamma \mathbb{E}_{a\sim\pi} Q_\pi(s_{t+1}, a) \mid s_t = s, a_t = a] && (2.7.7) \end{aligned}$$

The above equations can be further broken down as follows:

$$V_\pi(s) = \sum_{a \in A} \pi(a \mid s) Q_\pi(s,a) \quad (2.7.8)$$

$$Q_\pi(s,a) = R(s,a) + \gamma \sum_{s' \in S} P^a_{ss'} V_\pi(s') \quad (2.7.9)$$

$$V_\pi(s) = \sum_{a \in A} \pi(a \mid s)\Big(R(s,a) + \gamma \sum_{s' \in S} P^a_{ss'} V_\pi(s')\Big) \quad (2.7.10)$$

$$Q_\pi(s,a) = R(s,a) + \gamma \sum_{s' \in S} P^a_{ss'} \sum_{a' \in A} \pi(a' \mid s') Q_\pi(s', a') \quad (2.7.11)$$

These are called as *Bellman expectation* equations.

The optimal state value function and optimal action-value function also obey a set of recursive relationships, called *Bellman optimality* equations:

$$V_*(s) = \max_{a \in A} Q_*(s, a) \quad (2.7.12)$$

$$Q_*(s,a) = R(s,a) + \gamma \sum_{s' \in S} P^a_{ss'} V_*(s') \quad (2.7.13)$$

$$V_*(s) = \max_{a \in A} \Big(R(s,a) + \gamma \sum_{s' \in S} P^a_{ss'} V_*(s')\Big) \quad (2.7.14)$$

$$Q_*(s,a) = R(s,a) + \gamma \sum_{s' \in S} P^a_{ss'} \max_{a' \in A} Q_*(s', a') \quad (2.7.15)$$

As shown above, the main difference between the *Bellman expectation* equations and *Bellman optimality* equations is the max operation over actions, implying that the agent has to choose the one that yields the highest return. Whenever complete information of the environment is available, this can be reduced to a planning problem, solvable by Dynamic Programming. However, in most scenarios, we do not know $P^a_{ss'}$ or $R(s,a)$, so we cannot solve MDPs by applying Bellman equations directly.

2.7.1. Dynamic Programming

If the agent has access to the model of the environment (i.e. the model is fully known), Dynamic programming (DP) can be used to find the optimal policy through iteratively evaluating value function and improving policy. DP requires enormous computational resources and hence is not very efficient.

Policy iteration (Howard, 1960) constantly improves the policy to obtain the optimal state value function. In this algorithm, $V(s)$ is initialized to 0 and $\pi(s)$ is initialized to random for all states. $V(s)$ is then iteratively updated for all states using the following update rule derived from the Bellman equation, until convergence:

$$\begin{aligned} V_{t+1}(s) &= \mathbb{E}_\pi[r + \gamma V_t(s') \mid s_t = s] && (2.7.16) \\ &= \sum_a \pi(a|s) \sum_{s',r} P(s', r \mid s,a)[r + \gamma V_t(s')] && (2.7.17) \end{aligned}$$

This is known as the *policy evaluation* stage. The policy $\pi(s)$ is then updated for all the states using the following equation:

$$\pi'(s) = \arg\max_{a \in A} \sum_{s',r} P(s', r \mid s,a)[r + \gamma V_t(s')] \tag{2.7.18}$$

This is known as *policy improvement.* If policy improvement results in a better policy ($\pi' \geq \pi$) by acting greedily, then the policy evaluation and policy improvement steps are repeated until there is no change in the policy.

The Generalized Policy Iteration (GPI) is an iterative process that improves the policy by altering between *policy evaluation* and *policy improvement.* Policy evaluation tries to find the true values of all the states given the current policy. Policy improvement greedily chooses the action that maximizes the value of a given state resulting in a better policy (π').

Value Iteration (Howard, 1960) tries to perform both the above steps at once. Specifically, $V(s)$ is initialized to random values for all states. $V(s)$ is then updated iteratively for all states, using the following update rule until convergence:

$$V_{t+1}(s) \leftarrow \max_a \sum_{s',r} p(s', r \mid s,a)[r + \gamma V_t(s')] \tag{2.7.19}$$

The final policy can be computed by using argmax:

$$\pi'(s) \leftarrow \arg\max_a \sum_{s',r} p(s', r \mid s,a)[r + \gamma V_t(s')] \tag{2.7.20}$$

Value iteration is guaranteed to converge to the optimal values. Both value-iteration and policy iteration are *offline planning* algorithms where the agent assumes that the MDP model is known. GPI is guaranteed to converge to the optimal policy and usually takes fewer iterations to converge than the value-iteration algorithm, although each iteration itself is computationally more expensive.

2.7.2. Monte-Carlo methods

Monte-Carlo (MC) methods learn directly from episodes of experiences (model-free) without modeling the environment dynamics and computes the mean return from the observation. Consider complete episodes (that terminate): $s_0 a_0 r_1 s_1 a_1 r_2 \ldots s_T$, MC methods computes return $G_t{=}\sum_{k=0}^{T-t-1} \gamma^k R_{t+k+1}$, i.e. the return of the trajectory from time t onward. The empirical mean return for state s can now be obtained as:

$$V(s) = \frac{\sum_{t=1}^{T} 1[s_t = s] G_t}{\sum_{t=1}^{T} 1[s_t = s]} \tag{2.7.21}$$

where $1[S_t = s]$ is a binary indicator function. The visit of state s can be counted every time ("every-visit") or only the first time the state is encountered in one episode ("first-visit"). An action-value function can be obtained similarly:

$$Q(s,a) = \frac{\sum_{t=1}^{T} 1[s_t = s, a_t = a] G_t}{\sum_{t=1}^{T} 1[s_t = s, a_t = a]} \tag{2.7.22}$$

By following an idea similar to GPI, we can obtain the optimal policy through MC. The MC value estimates converge to that state's true value as the number of visits of that state tends to infinity. The MC estimate can have high variance, especially if that state is visited by only one trajectory (very less data) and if the policy or environment is stochastic.

2.7.3. Temporal Difference learning

Temporal Difference (TD) learning (Sutton, 1988) tries to combine the best of MC and dynamic programming. TD is model-free similar to MC and uses the current estimate of the value function as an approximation of future return similar to DP, an approach called *bootstrapping.*

MC requires complete episodes to update the value estimates based on the return (G_t) as follows

$$V(s_t) \leftarrow V(s_t) + \alpha\Big(G_t - V(s_t)\Big) \tag{2.7.23}$$

where G_t is the return from time t onward and $\alpha \in (0,1)$ is the step size. The main difference between MC and TD learning is that TD learning can learn from incomplete episodes. The TD learning update is given by the following equation:

$$V(s_t) \leftarrow V(s_t) + \alpha\Big(r_{t+1} + \gamma V(s_{t+1}) - V(s_t)\Big) \tag{2.7.24}$$

The difference between $(r_{t+1} + \gamma V(s_{t+1}))$ and $V(s_t)$ is called the **TD-error**.

Action-value estimation can be done similarly,

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\Big(r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)\Big) \tag{2.7.25}$$

Optimal policy in TD learning (known as “TD control”) can be learned through either *On-Policy SARSA* or *Off-Policy Q-Learning.*

2.7.4. SARSA

SARSA (Sutton and Barto, 2018) is an on-policy TD control method. We start with state s_0 and choose action $a_0 = \arg\max_{a \in A} Q(s_0, a)$ (ϵ-greedy is commonly applied). At time t, after applying action a_t, the agent gets reward r_{t+1} and transitions into the next state s_{t+1}. Then, the next action is selected in the same way, i.e. $a_{t+1} = \arg\max_{a \in A} Q(s_{t+1}, a)$. Q-values are then updated based on:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\Big(r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)\Big) \tag{2.7.26}$$

In every step of SARSA, the next action is also chosen according to the current policy.

2.7.5. Q-Learning

Q-Learning (Watkins and Dayan, 1992) is an off-policy TD control method in which the optimal action value function is learned from sampled transitions. We start with state s_0, at each time step t, action is selected according to the Q values i.e. $a_t = \arg\max_{a \in A} Q(s_t, a)$ and ϵ-greedy is applied commonly. Q-values are then updated based on:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\Big(r_{t+1} + \gamma \max_{a \in A} Q(s_{t+1}, a) - Q(s_t, a_t)\Big) \tag{2.7.27}$$

where $\alpha \in$ (0,1) is the step size. It assumes that from s_{t+1} onward, the agent will follow the policy which maximizes its current value estimates.

2.7.6. Deep Q-Networks (DQN)

Although memorizing the optimal Q values (Q_*) for all state-action pairs is theoretically possible, it becomes computationally infeasible with large state and action spaces. Q values can then be approximated using function approximation $Q(s, a; \theta)$. The Deep Q-Network (DQN) algorithm (Mnih et al., 2013) extends the Q-Learning algorithm with neural networks as function approximators to approximate $Q(s,a)$ and achieve impressive results. However, Q-learning can suffer from instability and divergence when combined with non-linear function approximators and bootstrapping. To tackle this issue, the DQN paper proposes two innovative solutions.

- **Experience Replay**: All the transitions in an episode are stored in one replay buffer (RB). Replay buffer has experiences over many episodes. Batches are drawn at random from this replay buffer for updating the Q-values, hence samples could be reused multiple times. This trick helps improve data efficiency and removes correlations in the sequences observed.
- **Target network**: Using the same deep neural network to learn and compute target action-values makes the problem highly non-stationary. DQN solves this issue by using a separate target network whose weights are frozen to compute the target values, which makes the learning stable. The weights of the action-value network are copied onto the target network after every k updates, where k is a hyper-parameter.

The loss function is as below:

$$L(\theta) = \mathbb{E}_{(s,a,r,s')\sim U(RB)}\Big[\Big(r + \gamma \max_{a'} Q(s', a'; \theta^-) - Q(s,a; \theta)\Big)^2\Big] \tag{2.7.28}$$

where $U(RB)$ is a uniform distribution over the replay buffer RB; θ^- are the parameters of the target Q-network (frozen).

2.8. Policy Gradient

Policy optimization is a family of RL algorithms that try to learn $\pi_\theta(a|s)$ through either directly optimizing $J(\pi_\theta) = \mathbb{E}_{\tau\sim\pi_\theta}[G(\tau)]$ using gradient ascent or indirectly maximizing local approximations of $J(\pi_\theta)$. This optimization can be done through either *on-policy* or *off-policy.* In *on-policy*, the rollouts are generated by acting in the world using the latest version of the policy, whereas in *off-policy* the data used for learning comes from a different policy than the one we are trying to optimize. The most popular algorithms of this family that have achieved impressive results are Policy Gradient (Williams, 1992), Actor Critic methods (Sutton and Barto, 2018), Trust Region Policy Optimization (Schulman et al., 2015), and Proximal Policy Optimization (Schulman et al., 2017).

Policy gradient methods directly optimize the policy $\pi_\theta(a|s)$ through gradient ascent. Let $J(\theta)$ be the expected return and our objective is to maximize this expected return.

$$J(\theta) = \sum_{s\in S} d_{\pi_\theta}(s) V_{\pi_\theta}(s) \tag{2.8.1}$$

$$= \sum_{s\in S} \Big(d_{\pi_\theta}(s) \sum_{a\in A} \pi(a|s,\theta) Q_\pi(s,a)\Big) \tag{2.8.2}$$

where $d_{\pi_\theta}(s)$ is the stationary distribution of Markov Chain for π_θ. Intuitively, the goal is to increase the probability of the actions with a higher expected return. The aim is to use gradient ascent to optimize the parameters θ of the policy as follows:

$$\theta_{k+1} = \theta_k + \alpha \nabla_\theta J(\theta) \tag{2.8.3}$$

$$
\begin{aligned}
J(\theta) &= \mathbb{E}_{\pi_\theta}[r] && (2.8.4)\\
&= \sum_{s\in S}\Big(d_{\pi_\theta}(s)\sum_{a\in A}\pi(a|s;\theta)Q_\pi(s,a)\Big) && (2.8.5)\\
&\propto \sum_{s\in S} d(s)\sum_{a\in A}\pi(a|s;\theta)Q_\pi(s,a) && (2.8.6)\\
\nabla J(\theta) &= \sum_{s\in S} d(s)\sum_{a\in A}\nabla_\theta\pi(a|s;\theta)Q_\pi(s,a) && (2.8.7)\\
&= \sum_{s\in S} d(s)\sum_{a\in A}\pi(a|s;\theta)\frac{\nabla_\theta\pi(a|s;\theta)}{\pi(a|s;\theta)}Q_\pi(s,a) && (2.8.8)\\
&= \sum_{s\in S} d(s)\sum_{a\in A}\pi(a|s;\theta)\nabla_\theta\ln\pi(a|s;\theta)Q_\pi(s,a) && (2.8.9)\\
&= \mathbb{E}_{\pi_\theta}[\nabla_\theta\ln\pi(a|s;\theta)Q_\pi(s,a)] && (2.8.10)
\end{aligned}
$$

The above result is the **Policy Gradient Theorem** and it lays the foundation for various policy gradient algorithms. Vanilla PG also known as REINFORCE relies on $Q_\pi(s,a)$, an estimated return by MC methods using episodic trajectories to update the policy parameter θ.

2.8.1. Policy Gradient with Baseline

In Equation 2.8.10, $Q_\pi(s,a)$ is used (more generally any return G_t can be used). This can also be implemented by comparing the action value (or return) with an arbitrary baseline $b(s)$ that is independent of action, as follows:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}[\sum_{t=0}^{T}\nabla_\theta\log\pi_\theta(a_t|s_t)(G_t - b(s_t))] \tag{2.8.11}$$

The baseline b is chosen such that the variance of gradient estimation is reduced and it is not dependent on actions. In many scenarios, the baseline is simply the moving average of the rewards observed up to the current time step which encourages the agent to take actions that are at least as good as the average value of the actions taken thus far. A commonly used baseline is state-value $V(s)$, in which case we would have the *advantage* $A(s,a) = Q(s,a) - V(s)$.

2.8.2. Actor-Critic methods

If state-value $V(s)$ function is learned in addition to the policy π, we get Actor-Critic algorithm. This has two components: i) A **Critic** that updates value function parameters w (either action-value $Q(s,a)$ or state-value $V(s)$), and ii) An **Actor** that updates policy parameters θ based on critic's suggestion. In the standard actor-critic setup, the actor and critic update equations are as below.

Actor update (policy parameters):

$$\theta \leftarrow \theta + \alpha_\theta Q(s_t,a_t;w)\nabla_\theta \ln \pi(a_t|s_t;\theta). \tag{2.8.12}$$

Critic update:

$$w \leftarrow w + \alpha_w \delta_t \nabla_w Q(s_t,a_t;w) \tag{2.8.13}$$

where $\delta_t = r_t + \gamma Q(s_{t+1}, a_{t+1}; w) - Q(s_t,a_t;w)$ and α_θ, α_w are learning rates of actor and critic respectively.

Other variants have been proposed that vary in how the actor and critic are implemented and/or updated.

This concludes the primer on machine learning and reinforcement learning. In the next chapter, we present topics in lifelong learning and MARL.

Chapter 3

Lifelong Learning and Multi-agent Reinforcement Learning

In this chapter, we will introduce the lifelong learning problem and also provide the necessary background on MARL required for understanding the work to be presented in this thesis.

3.1. Lifelong Learning

In recent years, ML algorithms and in particular Deep Learning (Goodfellow et al., 2016), a subfield of ML that uses neural networks to automatically extract features from data has achieved significant breakthroughs across different domains. This success can be partly attributed to the availability of large datasets and faster compute. Deep learning models have achieved great performance in computer vision in a variety of tasks : object classification (He et al., 2016, Krizhevsky et al., 2012, Simonyan and Zisserman, 2014), object detection (Bochkovskiy et al., 2020, Sandler et al., 2018), semantic segmentation (Badrinarayanan et al., 2017, He et al., 2017, Jégou et al., 2017), object tracking (Danelljan et al., 2017, Wang et al., 2019), etc. In natural language processing, deep learning models are dominating in various applications like machine translation (Bahdanau et al., 2014, Lample et al., 2017, Vaswani et al., 2017), dialogue systems (Gao et al., 2018, Zhang et al., 2019), text classification (Conneau et al., 2016, Kumar et al., 2016, Yang et al., 2016), etc. Large scale models that learn general purpose representations in an unsupervised manner from massive datasets like BERT, GPT-2, GPT-3 (Devlin et al., 2018, Radford et al., 2019), which can then be used for several down stream tasks, have also become very popular. With regard to game play, deep learning models have achieved superhuman performance in perfect information games such as Chess, Go or Shogi (Silver et al., 2018) and even in games involving mix of betrayal and teamwork such as Diplomacy (Anthony et al., 2020, Paquette et al., 2019). Other areas such as medical imaging (Gulshan et al., 2016), drug discovery (Gottipati et al., 2020), robotics (Schulman et al., 2015), creativity (Karras et al., 2020) has also seen deep learning flourish. Thus far, *supervised learning* is the paradigm that has turned out to be most successful in deep learning. It typically involves collecting a large training dataset of input/output pairs which is both expensive and laborious task. Annotating these large datasets is another cumbersome task, for eg. it could take about four minutes on average to annotate a single image for semantic segmentation tasks (Bearman et al., 2016). Hence, they are highly **sample inefficient**. These supervised learning models also typically have large training times (slow convergence) and are prone to changes in the training environment, often requiring *re-training* from scratch. Thus, these models have poor adaptability.

Lifelong learning (Thrun, 1998) attempts to break the learning problem into smaller tasks and learn from these tasks in an incremental manner. This paradigm can potentially solve the main drawbacks of the supervised deep learning models highlighted above which are **sample inefficiency**, **slow convergence** and **poor adaptability**.

In contrast, humans can quickly and continually learn new tasks while maintaining the skills to solve previously learned tasks. The ability of an AI system to effectively update new information over time is known as lifelong learning or continual learning, and one can postulate this as one of the fundamental ingredients of general AI. Balancing between learning from recent experiences while not forgetting the knowledge acquired from the past is a well-studied problem known as the **stability-plasticity dilemma** (Carpenter and Grossberg, 1987) in neural networks. The *stability* refers to preserving the previously learned knowledge while *plasticity* refers to updating the model to learn new tasks. Increasing the plasticity of the model leads to more forgetting i.e. less stability and on a similar note, increasing the stability of the model can prevent the network from learning new tasks. *Catastrophic forgetting* is a phenomenon in which training the model with new information obstructs previously learned knowledge. This is a common failure case in training neural networks to adapt to new tasks or learning from non-stationary data streams (i.e. non-iid) (McCloskey and Cohen, 1989). Alleviating catastrophic-forgetting is crucial to enable real-world applications where input distributions can shift and where retraining on past data or from scratch is infeasible. While lifelong learning has been identified as an important and challenging problem decades ago (Ring, 1998, Thrun, 1998), it has recently seen a surge of interest (Aljundi et al., 2018, Chaudhry et al., 2018, 2019, Kirkpatrick et al., 2017, Lopez-Paz and Ranzato, 2017) with the success of deep learning.

A typical supervised ML setup assumes the availability of a labeled dataset $D = \{x_i, y_i\}_{i=1}^{N}$, that is sampled independent and identically distributed (i.i.d) from a joint probability distribution $P(X,Y)$ that is usually fixed. The objective is to learn a model $f_\theta : X \to Y$, that is parametrized by θ which predicts a target vector ($\bar{y} \in Y$) for any input ($\bar{x} \in X$). This is usually done through the principle of *Empirical Risk Minimization* (ERM) (Vapnik, 2013).

$$\mathbb{E}[l(f(x), y)] = \int l(f(x), y) dP(X,Y) \approx \frac{1}{N} \sum_{i=1}^{N} l(f(x_i), y_i) \tag{3.1.1}$$

where l is a loss function or objective function. The objective of the neural network optimizer is to minimize this loss function.

In a lifelong learning setup, the learning problem is divided into a sequence of tasks $T = \{T_1, T_2, \ldots\}$, where the number of tasks may not be known upfront. Each task t consists of N_t data points drawn from a probability distribution $(x_i, y_i) \sim P_t$, where x, y are the input and output respectively. Formally, the multitask objective function is

$$\frac{1}{T} \sum_{t=1}^{T} \frac{1}{N_t} \sum_{i=1}^{N_t} l(f(x_i^t), y_i^t) \tag{3.1.2}$$

However, the constraint imposed in lifelong learning is that all tasks are not observed at once but in a sequence. Specifically, when accessing the dataset of task t, the datasets of all other tasks ($i \neq t$) are not available. This makes the problem **non-iid** and equation 3.1.2 is no longer valid, as it assumes that the data samples are drawn i.i.d from a fixed probability distribution.

3.1.1. Characteristics of a good lifelong learning system

An *ideal* lifelong learner should have the below desiderata:

- **Less forgetting/Backward transfer:** The lifelong learner should not forget the knowledge from previous tasks *catastrophically* as it experiences new tasks. It exhibits

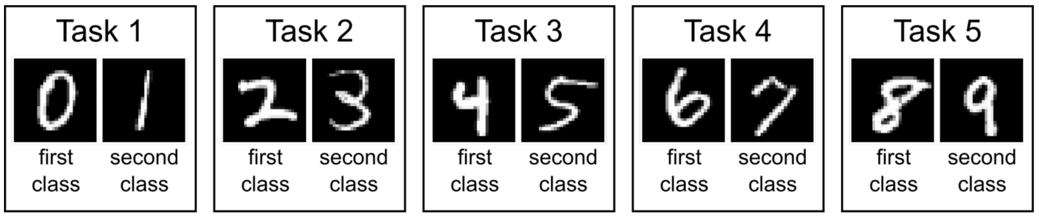


Fig. 3.1. Split MNIST protocol - The MNIST dataset is divided into 5 disjoint subsets, each subset having 2 randomly sampled classes without replacement. Image source: reproduced from the paper (Van de Ven and Tolias, 2019).

negative backward transfer if it forgets, while an ideal lifelong learner should exhibit positive backward transfer, i.e. performance on previous tasks should improve as it experiences similar new tasks.

- **Forward transfer**: As the learner encounters new tasks, it should be able to leverage knowledge gathered from previous tasks to learn new tasks quickly.
- **Limited memory**: The memory footprint of a lifelong learner should not grow linearly with the number of tasks.
- **Fast adaptation**: The lifelong learner should learn from a stream of data that is not randomly shuffled. Hence, it should adapt almost in real time to facilitate online learning.

3.1.2. Lifelong learning scenarios

When considering the problem of a single neural network adapting sequentially to a series of tasks, there could be three scenarios that occur of **increasing** difficulty (Van de Ven and Tolias, 2019). This categorization also helps to standardize evaluation enabling better comparison among methods.

- **Task-incremental** - In Task-incremental learning (Task-IL), model has the access to task-ID. Hence, common network architecture in this scenario has a “multi-headed” output layer with each task having its own output units but the rest of the network is usually shared.
- **Domain-incremental** - In Domain-incremental learning (Domain-IL), models are only required to solve the task at hand and task-ID is not available at test time.

- **Class-incremental** - In Class-incremental learning (Class-IL), the models must infer task-ID at test time.

For eg. in the Split MNIST protocol (Figure 3.1), we can think of the three scenarios as follows: a) Task-IL : Is it 1'st or 2'nd class given the task-ID? (e.g. 0 or 1), b) Domain-IL: Is it 1'st or 2'nd class with task-ID unknown, (e.g. [0,2,4,6,8] or [1,3,5,7,9]), c) Class-IL: Which class is it with task-ID unknown (e.g. choice from 0 to 9).

3.1.3. Lifelong learning methods

Existing algorithms for lifelong learning can be broadly classified into a) *Replay-based* methods, b) *Regularization-based* methods, and c) *Parameter isolation* methods.

- **Replay methods** - These methods store raw samples from the dataset or generates fake samples with a generative model. The samples from the previous tasks are replayed when learning a new task, which aids in alleviating *catastrophic forgetting.* Methods such as (Chaudhry et al., 2019, Rebuffi et al., 2017) explicitly retrain on the stored samples, thus are prone to overfitting the subset of stored samples, while methods such as (Chaudhry et al., 2018) constrain the optimization using these stored samples by projecting the gradients such that the average loss over the previous tasks does not increase.
- **Regularization-based methods** - These methods add an additional regularization term that consolidates knowledge from previous tasks when learning on new tasks, instead of storing samples. Assuming parameters of a neural network are independent, the importance of all its parameters is estimated and changes to important parameters are penalized. Methods such as Elastic weight consolidation (EWC) (Kirkpatrick et al., 2017), Variational Continual Learning (VCL) (Nguyen et al., 2017) are some examples of this category drawing inspiration from Bayesian methods. Other examples include distillation methods such as (Li and Hoiem, 2017) that focus on knowledge distillation from a previous model that is trained on an older task to the current model being trained with new data.
- **Parameter isolation methods** - These methods have different model parameters to every task to help alleviate forgetting. This could be through having new branches for new tasks while the old task parameters are frozen (Rusu et al., 2016) or having

a static architecture with every task having their allocated parts. In the latter case, parts corresponding to previous tasks are masked out during new task training (Fernando et al., 2017, Mallya and Lazebnik, 2018).

3.1.4. Benchmarks

Some of the commonly used benchmarks in supervised lifelong learning are as below:

- **Permuted MNIST**: MNIST dataset consists of 60,000 train images and 10,000 test images of handwritten digits of 10 classes. Each image is of size 28×28. Every task consists of a random permutation of the input pixels applied to all images in the dataset.
- **Rotated MNIST**: Every task is defined by applying some random rotation to all images.
- **Split MNIST**: The dataset is divided into 5 disjoint subsets, each subset having 2 randomly sampled classes without replacement (Figure 3.1).
- **Split CIFAR-10**: CIFAR-10 consists of 60,000 colored images of 10 different objects, each being being of size 32×32. In Split CIFAR-10, the dataset is divided into 5 disjoint subsets, each subset consists of 2 randomly sampled classes without replacement.
- **Split CIFAR-100**: CIFAR-100 consists of 60,000 colored images of 100 different categories. Each image is of size 32×32. In Split CIFAR-100, the dataset is divided into 20 disjoint subsets, each subset consists of 5 randomly sampled classes without replacement.
- **Split CUB**: CUB contains 11,788 colored images of 200 bird classes. Each image is of size 224×224. In Split CUB, the dataset is divided into 20 disjoint subsets, each subset consists of 10 randomly sampled classes without replacement.
- **Split miniImageNet**: miniImageNet contains 60,000 colored images of 100 different objects. Each image is of size 84×84. In Split miniImageNet, the dataset is divided into 20 disjoint subsets, each subset consists of 5 randomly sampled classes without replacement.

- **Core50**: Core50 (Lomonaco and Maltoni, 2017) is a dataset consisting of 50 domestic objects belonging to 10 categories. Classification can be performed at object level (50 classes) or at category level (10 classes).
- **CRIB**: CRIB (Stojanov et al., 2019) supports existing 3D datasets and is capable of generating unlimited data for incremental instance and category learning. They also introduce a new 3D object dataset, Toys-200 that contains 200 unique toy-like object instances.
- **OpenLoris**: OpenLoris (Shi et al., 2020) provides visual, inertial and odometry data recorded with real robots in real scenes, and ground-truth robot trajectories acquired by motion capture system or high-resolution LiDARs, primarily for benchmarking lifelong SLAM algorithms.
- **Stream 51**: Stream 51 (Roady et al., 2020) is a large-scale streaming dataset having images that are temporally correlated with training instances drawn from 51 unique classes.
- **IIRC**: IIRC (Abdelsalam et al., 2020) starts training with some coarse, high-level classes and observes new, fine-grained classes as it trains over new tasks.

With regard to lifelong RL benchmarks,

- (Henderson et al., 2017) proposed 50 new variations to OpenAI Gym environments through modifying some aspects of either the environments or agents like gravity, morphology of the agent's body, or goal positions.
- (Al-Shedivat et al., 2017) introduced **RoboSumo** — a 3D environment based on MuJoCo physics simulator that allows pairs of agents to compete against each other. The robots differ in anatomy: the number of legs, their positions, and constraints on the thigh and knee joints.
- (Lomonaco et al., 2020) designed **CRLMaze** based on VizDoom (Kempka et al., 2016), an object-picking lifelong learning task that is composed of 4 scenarios (Light, Texture, Object, All) of incremental difficulty and a total of 12 maps.
- (Cobbe et al., 2019) proposed **Coinrun**, that is a procedurally generated environment having different training and testing sets to measure generalization in RL.
- Recently, **Jelly Bean World** (JBW) (Platanios et al., 2020) was introduced as a testbed to develop agents with never-ending learning capabilities. It provides support

to create non-stationary environments with wide range of tasks including multi-task and multi-modal settings. These are 2D grid worlds filled with items in which agents can navigate.

While these are interesting benchmarks, they still need **synthetic** modifications to either the environment or the agent to introduce non-stationarity.

3.2. Multi-agent RL

Progress in deep RL has mostly focused on settings where a single agent needs to solve a static task. However, many real-world applications such as self-driving cars, cyber-physical systems, etc. involve environments that contain a large number of learning agents. Multi-agent RL (MARL) addresses the sequential decision-making problem when there are multiple autonomous agents that interact in a common environment, each of which aims to optimize its own long-term return. MARL takes into account the presence of other learning agents' in the environment and hence the environment is *non-stationary* from one agent's point of view. For this reason, MARL can be also be seen as a stepping stone towards developing systems that have human-like reasoning abilities. MARL framework can potentially learn higher cognitive concepts like *reciprocity*, *theory of mind* etc. that can facilitate artificial agents' cooperation amongst themselves, as well as humans. *Theory of mind* refers to the ability of an agent to think from other agents' perspectives and amend its actions.

3.2.1. MARL framework

The evolution of the state and the reward received by each agent are influenced by the joint actions of all agents. We can categorize framework for studying theoretical aspects into a) **Markov** or stochastic games and b) **Extensive-form** games.

A *Markov game* is defined by a tuple $(\mathcal{N}, \mathcal{S}, \{\mathcal{A}^i\}_{i\in\mathcal{N}}, \mathcal{P}, \{\mathcal{R}^i\}_{i\in\mathcal{N}}, \gamma)$, where $\mathcal{N} = \{1, \ldots, N\}$ denotes the set of $N > 1$ agents, $\mathcal{S}$ is the state space observed by all the agents, $\mathcal{A}^i$ denotes the action space of agent i. Suppose $\mathcal{A} := \mathcal{A}^1 \times \mathcal{A}^2 \times \ldots \times \mathcal{A}^N$, then $\mathcal{P} : \mathcal{S} \times \mathcal{A} \rightarrow \Delta(\mathcal{S})$ denotes the transition probability from any state $s \in \mathcal{S}$ to any state $s' \in \mathcal{S}$ for any joint action $a \in \mathcal{A}$, $\mathcal{R}^i : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{R}$ is the reward function that determines the immediate reward received by agent i for a transition from (s,a) to s'; $\gamma \in [0,1]$ is the discount factor.

At every time step t, each agent $i \in \mathcal{N}$ executes an action a_t^i in state s_t, transitions to state s_{t+1}. Each agent i receives a reward $R^i(s_t, a_t, s_{t+1})$. Every agent i tries to optimize its own long-term return by finding the policy $\pi^i : \mathcal{S} \to \Delta(\mathcal{A}^i)$ such that $a_t^i \sim \pi^i(.|s_t)$. Hence, the value-function $V^i : \mathcal{S} \to \mathbb{R}$ of agent i is a function of the joint policy $\pi : \mathcal{S} \to \Delta(\mathcal{A})$ (i.e. $\pi(a|s) := \Pi_{i\in\mathcal{N}}\pi^i(a^i|s)$).

$$V^i_{\pi^i,\pi^{-i}}(s) := \mathbb{E}\Big[\sum_{t\geq 0}\gamma^t R^i(s_t, a_t, s_{t+1})\Big|a_t^i \sim \pi^i(.|s_t), s_0 = s\Big] \tag{3.2.1}$$

where $-i$ represents indices of all agents in $\mathcal{N}$ except agent i.

Most common solution concept in *Markov game* is the **Nash Equilibrium** (NE) (Başar and Olsder, 1998). A nash equilibrium of the Markov game $(\mathcal{N}, \mathcal{S}, \{\mathcal{A}^i\}_{i\in\mathcal{N}}, \mathcal{P}, \{\mathcal{R}^i\}_{i\in\mathcal{N}}, \gamma)$ is a joint policy $\pi^* = (\pi^{1,*}, \pi^{2,*}, \dots \pi^{N,*})$, such that for any $s \in \mathcal{S}$ and $i \in \mathcal{N}$

$$V^i_{\pi^{i,*},\pi^{-i,*}}(s) \geq V^i_{\pi^i,\pi^{-i,*}}(s), \tag{3.2.2}$$

for any π^i. In essence, it characterizes an equilibrium point π^* from which none of the agents has any incentive to deviate. NE always exists for a discounted MG but may not necessarily be unique in general (Filar and Vrieze, 2012).

Cooperative setting: In a fully cooperative setting, all agents share a common reward function, i.e. $R^1 = R^2 = \dots R^N = R$. Hence, the value function and Q-function are identical to all agents, thus enabling the application of single-agent RL algorithms, e.g., Q-learning (eq 2.7.27). There could also be *team-average* reward models (Doan et al., 2019, Zhang et al., 2018) in which the agents are allowed to have different reward functions (potentially private to each agent), with the cooperative goal being to optimize the average reward

$$\bar{R}(s,a,s') := \frac{1}{N}\sum_{i\in\mathcal{N}} R^i(s,a,s') \tag{3.2.3}$$

for any $(s,a,s') \in \mathcal{S} \times \mathcal{A} \times \mathcal{S}$. This *team-average* model facilitates the development of *decentralized* MARL algorithms.

Competitive setting: Fully competitive settings in MARL can be usually modeled as *zero-sum* Markov games, i.e. $\sum_{i\in\mathcal{N}} R^i(s,a,s') = 0$ for any (s,a,s'). This setting is widely applicable in game-playing (Berner et al., 2019, Silver et al., 2017).

Mixed setting: In this setting, there is no restriction imposed on the goal and relationship among the agents. Each agent is self-interested, whose reward may be conflicting with others'. It is also known as the *general-sum* setting.

Challenges in MARL: *Non-unique learning goals*, *non-stationarity* due to multiple agents learning simultaneously, *scability* issues arising due to combinatorial nature, *credit assignment* i.e. which agent contributed how much for the team performance make MARL very challenging.

Common MARL methods: In *independent learning*, the agents can only observe local action and reward. Assuming stationary environment (i.e. by ignoring other agents in the environment), (Tan, 1993) use vanilla Q-learning, henceforth called as *Independent Q-Learners* (**IQL**). Hence, *independent learning* suffer from non-convergence in general. Joint-action learners (JALs) model the strategies of the opponents explicitly by taking into account the joint-action of all the agents in the Q-learning update (the agent can observe the actions of others) (Claus and Boutilier, 1998). Most works assume having a *centralized controller* that can collect information about joint actions, joint rewards and joint observations and design local policies for all agents (Foerster et al., 2018, Rashid et al., 2018) leading to the **centralized-training-decentralized-execution** paradigm. This paradigm simplifies the analysis for *cooperative settings* through the use of single-agent RL analysis, while for *non-cooperative settings*, it does not help much. Under this paradigm, information can be exchanged freely amongst all the agents during centralized training, as long as the final policies rely only on local observations during decentralized execution. A *fully-decentralized* learning scheme is usually preferred as the assumption of a central controller is not pragmatic in general, except in cases where the agents can access a simulator such as video games and robotics.

Decentralized-POMDP (**Dec-POMDP**) (Oliehoek and Amato, 2016) is an extension of *Markov* game model previously described, to partially observed setting. It shares the reward function and the transition model similar to *Markov* game, but the agent only has access to its local observations s_i. Partially observed Markov games under the cooperative setting are usually formulated as Dec-POMDPs. Standard MARL methods for Dec-POMDPs typically focus on the self-play (SP) setting (Tesauro, 1994) where agents learn by playing the game with themselves repeatedly. In self-play, the agent (single parameter set) controls both players during training and iteratively improves both players' strategies. The agent then uses this learned strategy at test time. If it converges, self-play finds a Nash equilibrium of the game and gives rise to superhuman performance in two-player zero-sum games such as

Chess, Go (Silver et al., 2018) and Poker (Brown and Sandholm, 2018). Methods such as Value Decomposition Networks (**VDN**s) (Sunehag et al., 2017), QMIX (Rashid et al., 2018) use self-play to optimize team performance. QMIX and VDN try to exploit independence among agents through factorizing value functions. VDNs decompose a team action-value function into individual value functions across the agents such that these pieces can be easily added. Similarly, QMIX decomposes a team action-value function. However, QMIX assumes a mixing network (instead of the sum) that combines the local values in a non-linear way, which can represent monotonic action-value functions.

As seen previously in section 2.2, it is imperative for an RL agent to *explore* to discover good policies through trial and error. However, naive exploration will add noise to the policy of the agent during the training process, thus making their actions less informative to their teammates. Bayesian Action Decoder (**BAD**) (Foerster et al., 2019) explores in the space of deterministic partial policies instead of actions, and samples these policies from a distribution that conditions on a common knowledge Bayesian belief. However, since BAD requires tracking an explicit common knowledge Bayesian belief, it adds a computational burden due to the required sampling steps as well as it uses expert knowledge regarding the game dynamics. Reliance on this common knowledge also limits the generality of the method. Typically, BAD is trained using actor-critic methods which are sample inefficient and suffer from local optima. Population-based training is used in (Foerster et al., 2019) to get around this, further increasing the number of samples required. Simplified Action Decoder (**SAD**) (Hu and Foerster, 2019) was proposed to address the above-mentioned drawbacks in BAD. SAD has a different approach towards resolving this conflict between being interpretable and exploring. It also relies on the *centralized training with decentralized execution paradigm*. During training, SAD allows other agents to not only observe the chosen exploratory action, but instead, agents also observe the greedy action of their teammates. The greedy actions are not executed by the environment and are only meant to be additional information to the teammates. In essence, in SAD each agent takes two different actions at every time step: a greedy action, which is not presented to the environment but observed by the teammates at the next time step as an additional input, and the exploratory action that gets executed by the environment and is observed by the teammates as part of the environment dynamics. During greedy execution, the observed environment action can be used instead of centralized information for the additional input, since now the agent has stopped exploring. Other Play (**OP**) (Hu et al., 2020) is a learning algorithm that enhances self-play by exploiting known symmetries (ϕ) in the underlying game. OP has been shown to construct good strategies for the *zero-shot coordination* problem. *Zero-shot coordination* refers to the setting where an RL agent should coordinate with unseen novel partners at test-time without any chance to adapt its policy to the partner (for eg. fine-tuning or co-training for a few games). They assume that every *Markov* game also comes with a set of symmetries that are arbitrary relabeling of the state and/or action space, that leave trajectories unchanged up to relabeling. Agents have no way to break the symmetry.

Extensive-form games: Extensive form games (Shoham and Leyton-Brown, 2008) is another paradigm used for modeling MARL problems. This framework can handle imperfect information in MARL and is rooted in computational game theory. These are usually used to model non-cooperative settings.

This concludes the presentation of fundamental background topics required for understanding the work presented in this thesis. In the next chapter, we jump straight into our *Lifelong Hanabi* setup.

.

Chapter 4

Lifelong Hanabi setup

In this chapter, we will introduce the Hanabi game as well as the *Lifelong Hanabi* setup, which is one of our main contributions of this thesis. We also argue how the fields of MARL and Lifelong learning are closely tied, how progress in one can benefit the other.

4.1. Hanabi

Hanabi is a partially-observable, fully cooperative, 2-5 player card game in which all players work together to form piles of cards referred to as fireworks. It has recently been proposed as a new benchmark challenge for AI research (Bard et al., 2020), in particular for *theory of mind* reasoning and *ad-hoc cooperation*. Each card has a rank from 1 to 5, and a color that could be one of the following: red, green, blue, yellow and white. Each firework (one per color) starts with a 1 and is completed once the 5 has been added. There are three 1s, one 5, two 2s, two 3s and two 4s for each of the colors, adding up to a total of 50 cards in the deck. Since there is only one 5, it needs to be used very carefully. In a 2-player game, each player maintains a 5-card hand. The unique characteristic in Hanabi is that players can observe the cards held by their team mates but they cannot observe their own cards and thus need to exchange information with each other in order to understand what cards can be played. Thus the players need to find conventions that allow them to exchange information effectively from their local observations through their actions. Actions taken by a player are observed by all team mates. In each turn, a player can play or discard a card in their hand, or give a hint to their partner by spending a hint token. Players can take "hint" actions, in which they reveal the subset of a team mate's hand that matches a specific rank or color.

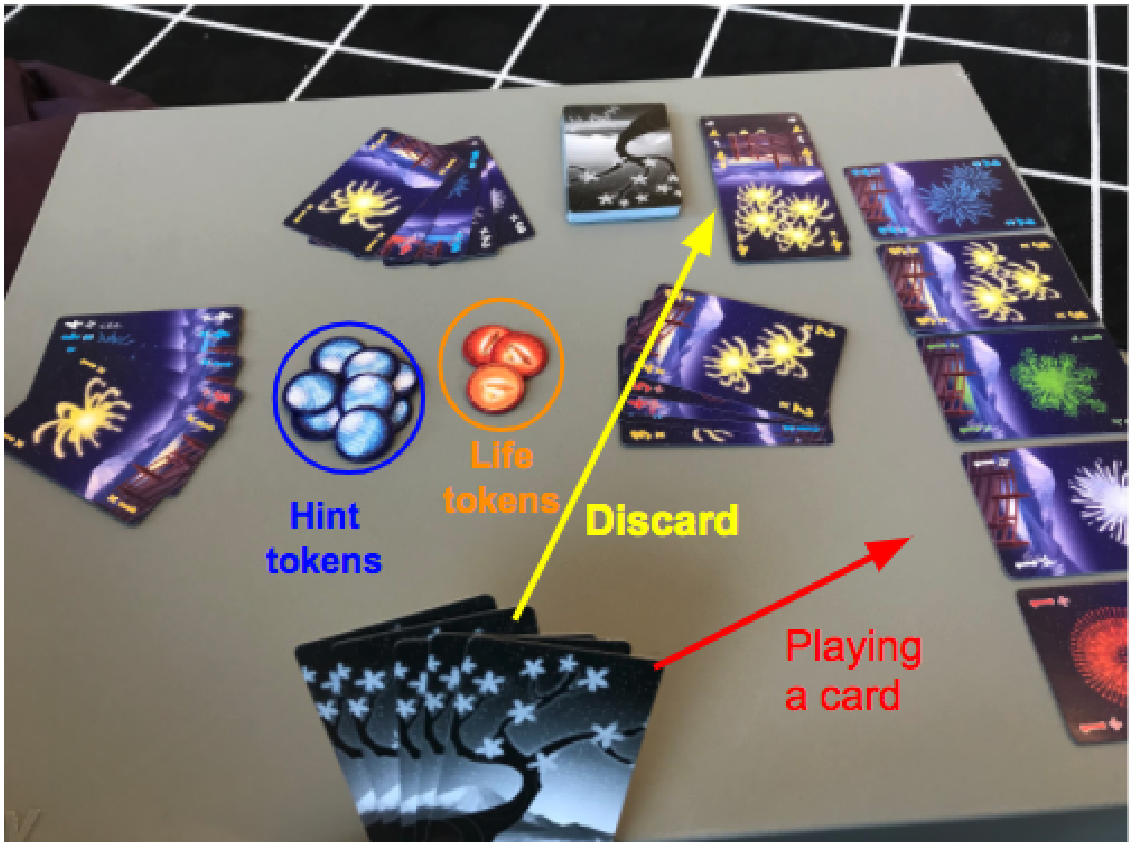


Fig. 4.1. Hanabi game state showing the hint tokens, life tokens, discard pile and playing cards.

Some example hints are "Your third and fifth cards are 1s" or "Your first and fourth cards are red". These hint actions cost information tokens that are scarce, which can be replenished by either discarding a card or successfully playing a 5 (i.e. completing one firework), both of which lead to regaining one hint token. The team shares 8 hint tokens at the start of the game. Discarding a card is an action that both removes the card from the game and makes it visible to all players. Finally, players can also choose to play a card. If this card is the next card for the firework of the corresponding color, it is added to the firework and the team scores one point. Otherwise the card is removed from the game, its identity is revealed, and the team loses one of the 3 life tokens. If the team runs out of life tokens ("bomb out") before the end of the game, all points accumulated so far are lost, and the game finishes immediately. In essence, the objective in Hanabi is to form completing stacks of cards in order from 1 to 5, one for each color. These rules result in a maximum score of $5 \times 5 = 25$ points in any game, which corresponds to all five fireworks being completed with five cards per firework. Figure 4.1 illustrates an example Hanabi game state. Thus, Hanabi is a challenging game that requires the agent to possess the *theory of mind* (Premack and Woodruff, 1978, Rabinowitz et al., 2018) in order to cooperate effectively. The *theory of mind* is the ability of an agent to see the world through the lens of other agents. It is about asking what a given action by another agent implies about the state of the world and it requires understanding of why this action was taken i.e. doing counterfactual reasoning.

Ad-hoc and Zero-shot coordination: Learning in Dec-POMDPs is typically done through self-play. Optimal self-play policies usually rely on arbitrary conventions, which the whole team can jointly coordinate on during training. However, many real-world problems require agents to cooperate with other, unseen agents and humans at test time. *Ad-hoc* team play setting in Hanabi refers to developing RL agents that can coordinate well with unknown novel partners (that are independently trained agents) that they have not been trained with, without providing the agent "sufficient" chance to adapt its policy to its partner at test-time. (Hu et al., 2020) refer to the setting where the RL agent has to coordinate strictly without providing "any" chance to adapt or fine-tune at test time as the *zero-shot coordination* setting. Both *ad-hoc* and *zero-shot* settings are crucial for developing learning algorithms that can produce robust solutions that are capable of potentially coordinating even with humans. The main difference between these two settings is that in the *ad-hoc* setting, the agent can play a few games with the new partner to adapt its policy, while in the *zero-shot* setting it has to strictly coordinate on the fly.

4.2. Multi-Agent RL and Lifelong Learning

MARL for Lifelong learning: As seen before, many machine learning algorithms make the assumption that the observations in the dataset are i.i.d. However, in many real-world scenarios, this assumption is violated because the underlying data distribution is non-stationary. Lifelong learning tries to address this problem, where the non-stationarity of data is usually described as a sequence of distinct tasks. On the other hand, MARL is inherently non-stationary due to changing behavior of other agents present in the environment. Therefore, MARL is a realistic scenario for lifelong learning. Another source of non-stationarity in MARL arises when the agent has to interact with different agents over its lifetime, even if the other agents are fixed. For example, in the game of Hanabi, we are interested in designing a single agent that can learn to coordinate well with a sequence of agents it will see over its lifetime. This is a lifelong learning problem.

Lifelong learning for MARL: Standard MARL methods typically focus on the centralized training with decentralized execution paradigm where agents have access to other agents' policies and observations during training (OroojlooyJadid and Hajinezhad, 2019, Zhang et al., 2019). Self-Play (Tesauro, 1995), the most common centralized training setting involves training a single agent against itself without any extra supervision. While this strategy works very well in competitive settings like playing the game of Go (Silver et al., 2016), in cooperative settings it can produce agents that establish highly specialized conventions that do not carry over to novel partners they have not been trained with (Bard et al., 2020). In particular, Bard et al. (2020) show that even though RL agents achieve a decent score after training in the self-play setting, their performance drops off sharply in the zero-shot coordination scenario, with some agents scoring essentially zero. Therefore, self-play agents fail to learn robust strategies that facilitate cooperation with *other* agents. Lifelong learning provides a natural framework to transfer knowledge from previous experience to future scenarios. Hence, in this thesis, we consider lifelong learning as an alternative to self-play in MARL with the hope that lifelong learning algorithms can learn to coordinate well with unseen agents.

4.3. Lifelong Hanabi: A Benchmark for Lifelong Reinforcement Learning

As shown in Figure 4.2, in our Lifelong RL setup, the *learner* ($\sim p_{train}$) is trained sequentially on a set of tasks $\mathbf{M} = \{M_t\}_{t=1}^{T}$ sampled from a distribution p_{train} over diverse strategies that perform well in Hanabi. The objective of the *learner* is to learn to coordinate well with its partners during continual training, with the ultimate goal of learning to coordinate well with unseen agents at the end of the training. During the testing phase, in order to measure generalization performance, the *learner* is evaluated with some random agents sampled from p_{test}. Although we consider Hanabi, a fully cooperative game in this work, our proposed lifelong RL setup can be easily extended to other multi-agent scenarios (e.g. fully competitive, mixed cooperative-competitive, etc.). Our proposed setup consists of three

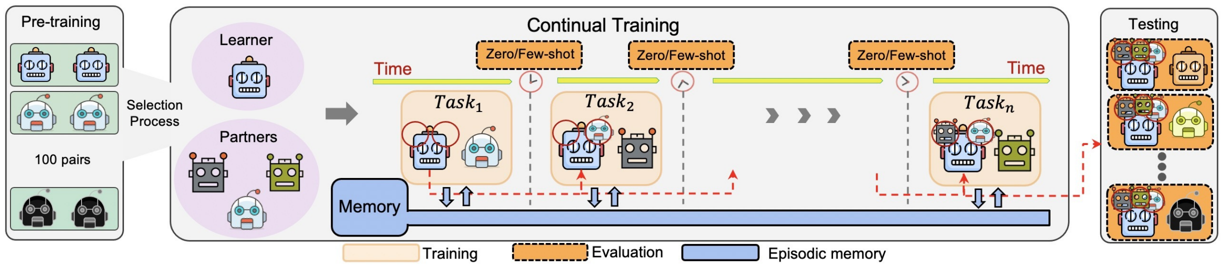


Fig. 4.2. Our Lifelong Hanabi setup consists of three phases: **1- Pre-training (Optional):** In this phase, a pool of agents are trained through SP, **2- Continual training:** The *learner* is taken from the pool ($\sim p_{train}$) and trained sequentially with some *partners* ($\sim p_{train}$) and periodically evaluated against all the partners, **3- Testing:** The *learner* is evaluated with a set of random agents excluding its partners ($\sim p_{test}$) to measure generalization.

phases: (1) Pre-training, (2) Continual-training, and (3) Testing. The detailed description of each phase is as follows:

Pre-training: In this phase, agents are trained through SP to play the game of Hanabi. We consider several recent MARL methods for training the agents (IQL/VDN/OP/SAD, and their combinations; refer sections 3.2.1) across different seeds and architectures leading to a pool of agents having diverse strategies.

Continual-training: An agent sampled from the pool is chosen as the *learner* and is trained sequentially with a set of agents (partners) for a fixed number of games per partner. The *learner* is also periodically evaluated with all its partners under both zero-shot and few-shot settings. In order to implement memory-based lifelong learning algorithms (ER, A-GEM, etc.), we also include an episodic memory that is used to store some transitions from every task, which can be then be used for replaying in the future tasks.

Testing: The *learner* is evaluated with K random agents sampled from the pool excluding its partners in order to measure the generalization performance.

4.3.1. Pool of Agents

The cross-play matrix of all the 100 agents trained with different MARL algorithms is shown in figure 4.3. There are five types of architectures with two different seeds per MARL algorithm. Auxiliary task (AUX) in Hanabi is to predict the card of player's own hand. In our experiments, it is to predict the status of a card, which can be playable, discardable, or

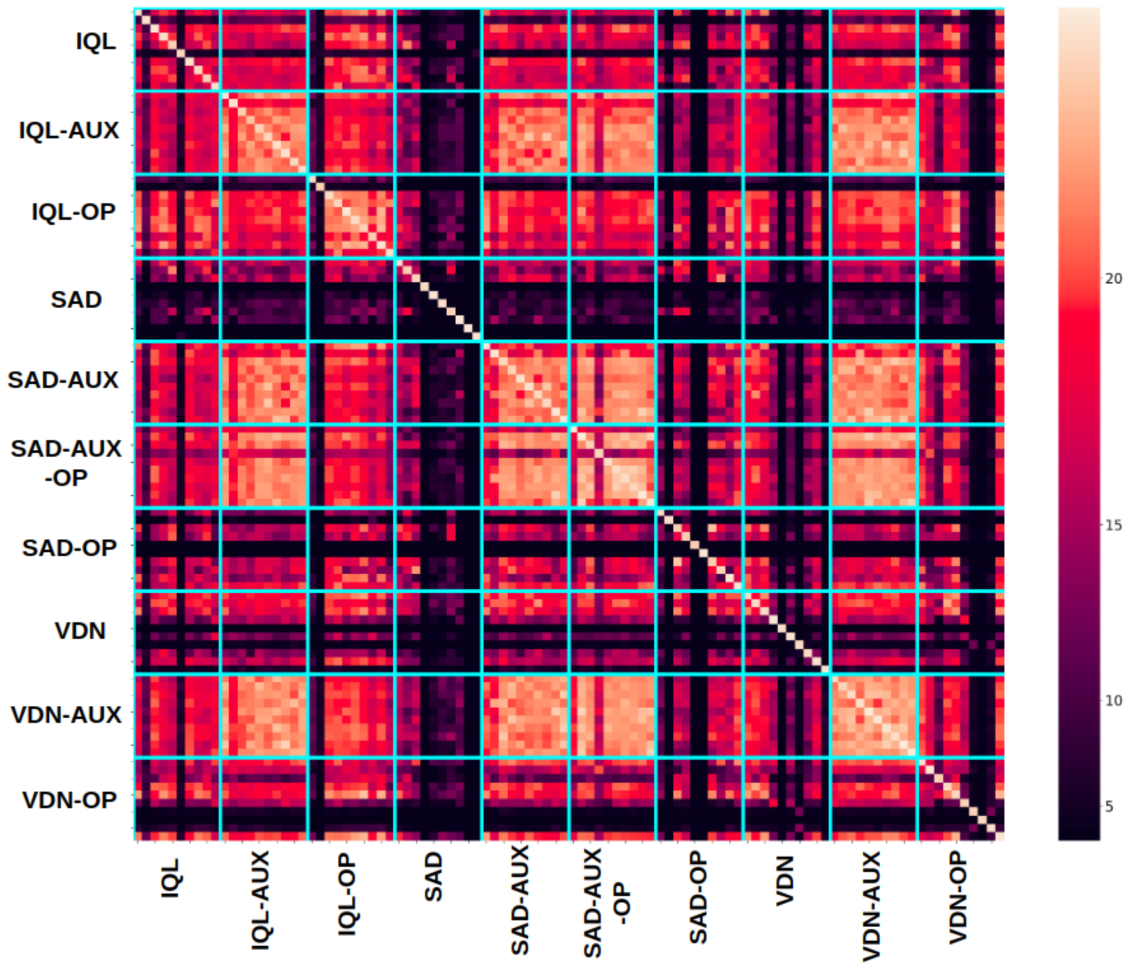


Fig. 4.3. The pool of 100 agents pre-trained through self-play using different MARL methods (IQL/VDN/OP/AUX/SAD, and their combinations). 10 agents having 5 different architectures with 2 seeds are generated with each of these MARL methods. $(i, j)_{th}$ element is the average score of agent i paired with j over 5k games. The diagonal entries indicate SP scores.

unknown. The loss between the predicted hand and ground-truth labels is the average cross entropy loss per card and is simply added to the TD-error of RL during training.

4.3.2. Evaluation methods

We consider two modes of evaluation in our setup : (a) zero-shot and (b) few-shot. In zero-shot setting, the *learner* is evaluated with another agent without providing it a chance to make updates to its own policy through its interaction with the other agent. On the other hand, in the few-shot setting, the *learner* plays a few games with the other agent, thereby adapting its policy through interaction, before being evaluated. We believe agents performing well under both these evaluation settings are crucial to developing AI systems that can adapt well to unknown partners not just in Hanabi but can also facilitate effective collaboration with humans. Few-shot evaluation setting also opens a door to explore recent advances in Meta-RL algorithms to enable fast adaptation.

4.3.3. Metrics

We measure the *learner*'s performance during continual training with some standard metrics from the lifelong learning literature such as average score (A), forgetting (F), and forward transfer (FT) (Chaudhry et al., 2018, Lopez-Paz and Ranzato, 2017). We also define a metric for measuring OOD generalization in our setup called generalization improvement score (GIS), inspired by Zhang et al. (2018). To calculate these metrics, we map Hanabi scores at the end of the game from [0, 25] to [0, 1] to have values more consistent with the literature.

Average score ($A \in [0, 1]$): Let $a_{i,j}$ be the score of the *learner* versus the j^{th} partner, after training it with the i^{th} partner in sequential training. The average score of the *learner* at task t (A_t) is defined as:

$$A_t = \frac{1}{t} \sum_{j=1}^{t} a_{t,j} \tag{4.3.1}$$

Forgetting($F \in [-1, 1]$): Let f_j^t represent the forgetting on task j after the *learner* is trained on task t and is computed as:

$$f_j^t = \max_{l \in 1,\ldots,t-1} a_{l,j} - a_{t,j} \tag{4.3.2}$$

The average forgetting at task t is then defined as:

$$F_t = \frac{1}{t-1} \sum_{j=1}^{t-1} f_j^t \tag{4.3.3}$$

Average future score ($FT \in [0, 1]$): Let $a_{i,j}$ be the score of the *learner* versus the j^{th} partner, after training it with the i^{th} partner in sequential training. The average future score (or forward transfer) of the *learner* at task t (FT_t) is defined as:

$$FT_t = \frac{1}{T-t} \sum_{j=t+1}^{T} a_{t,j} \tag{4.3.4}$$

Generalization Improvement Score ($GIS \in [0,1]$): (Zhang et al., 2018) define in-task generalization as the difference in the performance of an RL algorithm between a set of training and testing trajectories all generated by the same simulator. In this case, the only source of variation is through random seeds. However, in our work, we are more interested in the out-of-task generalization that can be defined as follows. Let $a_{0,k}$ and $a_{N,k}$ be the score of the *learner* versus the k_{th} random agent sampled from the pool (different from its partners) before the start of LLL and at the end of continual training, respectively (T is number of tasks in LLL). The GIS is computed as follows:

$$GIS = \frac{1}{K} \sum_{k=1}^{K} (a_{T,k} - a_{0,k}) \tag{4.3.5}$$

where K is the total number of unseen agents.

This concludes the presentation of our *Lifelong Hanabi* setup as well as the modes and metrics used for evaluation. In the next chapter, we benchmark several popular lifelong learning algorithms and provide experimental results.

Chapter 5

Experiments and Results

In this chapter, we understand how standard lifelong learning algorithms perform in *Lifelong Hanabi* and also provide experimental evidence on *hard* and *easy* settings that we introduce later in this chapter. In addition, we also see how well our lifelong-learned agent performs when compared to other MARL methods on the *zero-shot* coordination setting.

Figure 5.1 showcases how continual training can lead to improved scores and better zero-shot coordination. An IQL agent (pre-trained with self-play) is trained sequentially with 5 partners that correspond to the *hard* setting (ordering denoted by arrows in Figure 5.1), and then evaluated with both its partners seen during continual training as well as some unseen agents (SAD+AUX+OP, VDN+AUX). In this figure, we consider 5 agents each of VDN+AUX and SAD+AUX+OP mainly for measuring the generalization performance of our lifelong learned agent. Recall from Chapter 3 that a SAD agent allows other agents to not only observe the chosen exploratory action but also observe the greedy action of their teammates, while a VDN agent decomposes a team action-value function into individual value functions across the agents such that the pieces can be easily added. Auxiliary task (AUX) is predicting the player's own hand. Sections (C) and (E) in Figure 5.1 show the performance of the IQL agent on unseen agents before and after continual training respectively, clearly showing signs of OOD generalization. Likewise, sections (A) and (D) show the performance before and after training respectively with its partners used in continual training. For completeness, we also show the performance of MTL with partners as well as with the same unseen agents after training.

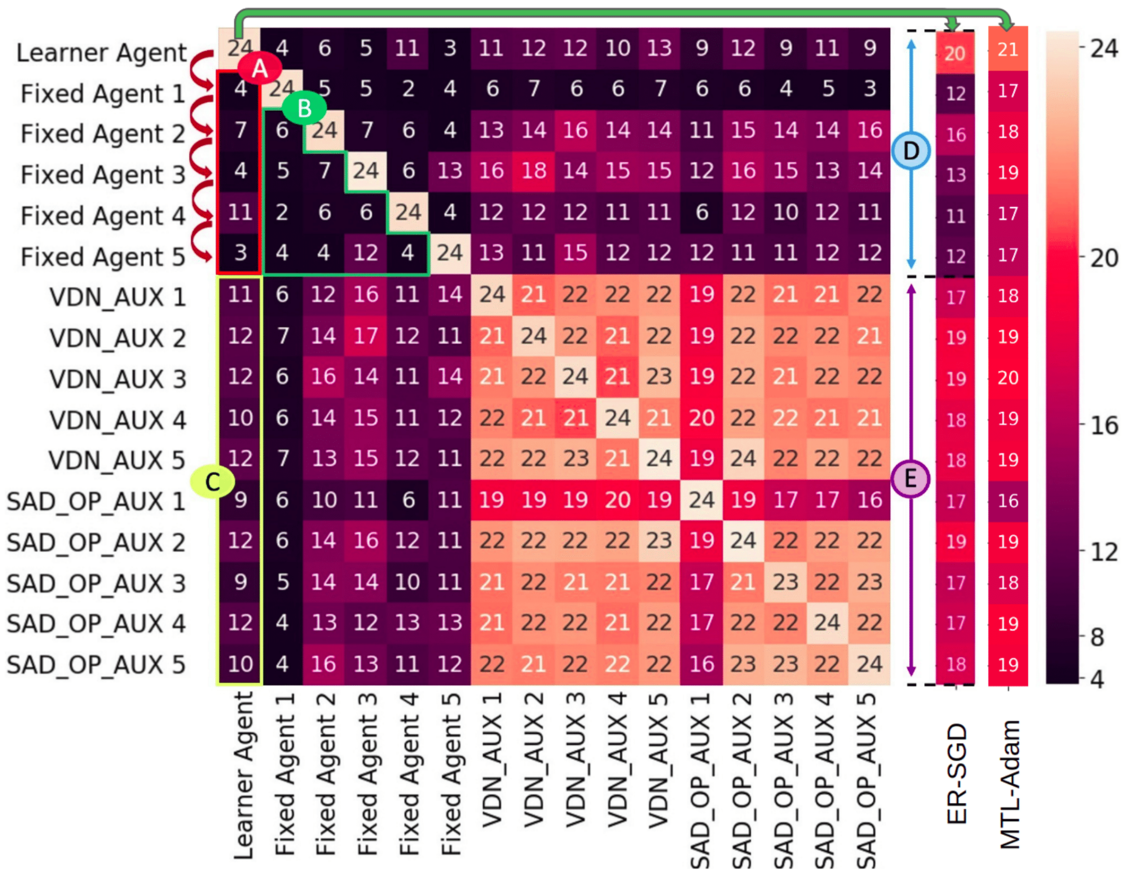


Fig. 5.1. CP scores before (left) and after continual training (right) – $(i, j)_{th}$ element is the average score of agent i paired with j. [A-C] is before continual training – (A) Initial scores of the *learner* with its partners, (B) Cross-play scores amongst the partners, low scores indicate they are far apart in the strategy space, (C) Initial generalization scores with some unseen agents. The *learner* is then trained continually with its partners following the order indicated by the arrows. [D-E] is after continual training – (D) Scores with the original learner and its partners, (E) Generalization scores with the same unseen agents.

As described in Section 4.3, we first pre-train a set of agents to play the game of Hanabi through self-play. Our RL agents are based on the distributed deep recurrent Q-learning with prioritized experience replay R2D2 architecture (Kapturowski et al., 2018). Training consists of a large number of parallel environments that invoke a deep neural network to approximate the Q function at each time step to generate trajectories, a prioritized replay buffer that is distributed and shared by N asynchronous actors to store the trajectories (Horgan et al., 2018). A training loop updates the neural network using samples from the replay buffer. In each actor thread, there are K environments run sequentially and the queries from these different environments are batched dynamically together making it efficient to run on GPUs (Espeholt et al., 2019), thus improving the throughput and efficiency. To update the model, a centralized trainer samples mini-batches from the replay buffer. The observation batch is then fed into an actor that utilizes GPUs to compute a batch of actions. All asynchronous actors share 2 GPUs leading to faster data generation and the trainer uses another GPU for gradient computation and model updates. This is different from the standard RL settings which run single actor and single environment in each thread on a CPU. This mechanism enables to run a very large number of simulations with moderate computation resources. In all Hanabi experiments, we run $N = 10$ actor threads with $K = 80$ environments. A similar setting was first used in the SAD paper by (Hu and Foerster, 2019).

Table 5.1. Exact architectures used in the pool.

Agent	RNN type	Num of feed-forward layers	Num of RNN layers	RNN hid dim
Type-1	LSTM	1	1	256
Type-2	LSTM	2	2	256
Type-3	LSTM	1	2	512
Type-4	GRU	1	2	256
Type-5	GRU	2	1	512

The diversity in strategies learned by these self-play agents are controlled by varying the seed, the MARL methods used for training (either IQL/VDN/OP/SAD and their combinations), type of recurrence (LSTM/GRU), number and dimension of recurrent layers, number and dimension of feed-forward layers before recurrence. The exact architecture details are described in the table 5.1. A pool of more than 100 self-play agents is created this way. A subset of 100 agents with 10 agents from each of these 10 MARL methods are used to generate the *cross-play* (CP) matrix as shown in Figure 4.3 in chapter 4. The entries in this matrix (diagonal entries indicate self-play scores) are obtained through the gameplay of agents with each other for 5k games, and then averaging the scores.

We propose two levels of tasks based on these scores that have different difficulty levels: *easy* and *hard*. In both the settings, one of the agents is used as *learner* and the rest as its *partners* that are fixed during continual training i.e. they represent different tasks since these agents have different strategies. Both the *learner* and its *partners* are initialized with weights of the pre-trained agents as we found pre-training to be crucial to learn some basic knowledge of Hanabi since training Hanabi agents from scratch in a decentralized manner is extremely hard. Our preliminary experiments suggested that fully decentralized learning with current deep MARL methods can be extremely challenging. However, we found that pre-training is not essential for simpler games such as the lever game (Hu et al., 2020). The names of these tasks are self-explanatory in the sense the *learner* in the *hard* version has to start from a much lower *cross-play* score with its partners and learn to achieve a good final score (out of 25). During continual training, the *learner* is trained sequentially with every *partner* for a fixed number of epochs and evaluated periodically with all its *partners* under both zero-shot and few-shot settings. In the zero-shot setting, the *learner* is directly evaluated with all its *partners*, while in the few-shot setting, the *learner* is fine-tuned with its *partner* for a few gradient steps before evaluating against the same partner. In both these settings, scores are reported based on an average of over 5k gameplay. The complete set of hyperparameters used in our experiments are listed in Tables 6.1 and 6.2.

Although the *hard* and *easy* settings consist of five tasks, our setup as such can be effortlessly extended to any number of tasks by selecting a different number of partners from the pre-trained pool and the pre-trained pool can itself be expanded by training more agents through SP. For instance, we report results on 10 tasks in section 5.4 and section 5.6. Note that to choose the partners, we excluded all the agents using either SAD or AUX from the pre-trained pool as we wanted to compare the continually trained learner with them in terms of zero-shot coordination. We also wanted to select partners that have low CP scores so that the tasks are diverse.

R2D2 agents keep recent game transitions in a fixed-size prioritized replay buffer (Schaul et al., 2015). At the end of every task, the replay-buffer is sliced and stored in an episodic memory, which is then used for replay in different memory-based lifelong learning algorithms that we consider in our benchmark. The *learner* can also start with random parameters (i.e. without pre-training), albeit, as previously mentioned this setting is very hard for the game of Hanabi. Our setup also contains support to restrict training based on the number of environment steps taken by the *learner* instead of a pre-determined number of epochs.

We aim to answer the following questions through our experiments : (1) How well standard LLL algorithms perform in our setup (section 5.2), (2) How well do these LLL algorithms fare under constrained memory and compute settings (section 5.3), (3) How lifelong RL methods perform in our setup (section 5.4), (4) How well the agents trained in our setup do in zero-shot and few-shot coordination scenarios in Hanabi (section 5.5) when compared to other recent methods such as OP (Hu et al., 2020).

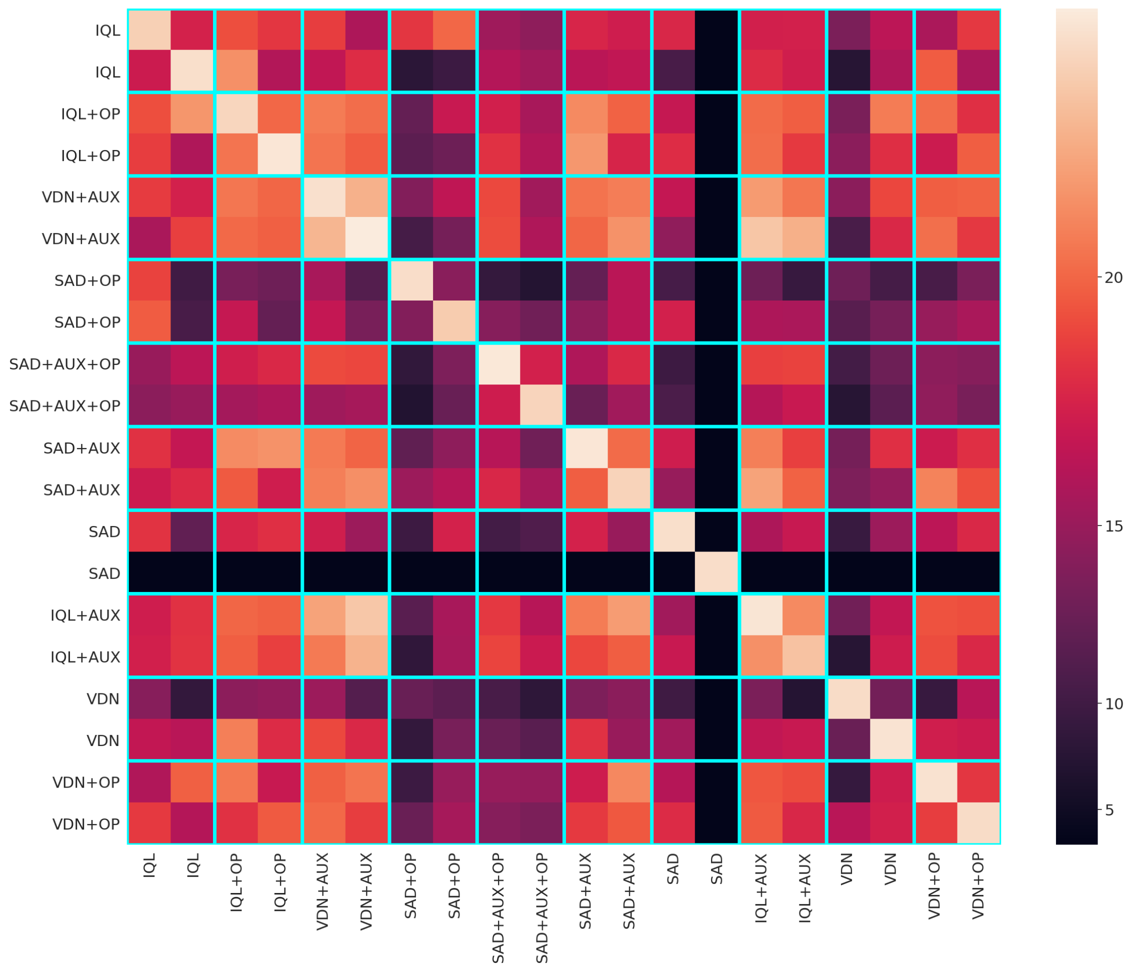


Fig. 5.2. Diverse agents used in our Inter-CP set for evaluating across-method performance of agents trained in our setup.

5.1. List of agents

In this section, we present the exact type of agents that we use as the *learner* and its *partners* in both *easy* and *hard* settings, as well as the set of 10 partners used in section 5.4

and section 5.6. All these settings have an IQL agent of Type-2 as the *learner* and a sequence of 5/10 agents (can be extended to any number of agents) as its *partners*. Recall that the table 5.1 has details of the exact architectures corresponding to these *Types*.

- **Easy** : *Learner* — $\Big\{$ IQL (Type-2) $\Big\}$
  Partners — $\Big\{$ IQL (Type-1), VDN (Type-3), VDN (Type-5), IQL+OP (Type-2), VDN+OP (Type-5) $\Big\}$.
- **Hard** : *Learner* — $\Big\{$ IQL (Type-2) $\Big\}$
  Partners — $\Big\{$ VDN+OP (Type-3), VDN (Type-4), VDN (Type-5), IQL+OP (Type-3), VDN (Type-3) $\Big\}$.
  The partner agents in the Figure 5.1 are these *hard* agents.
- **10 agents**: *Learner* — $\Big\{$ IQL (Type-2) $\Big\}$
  Partners — $\Big\{$ VDN (Type-2), VDN (Type-3), IQL+OP (Type-2), VDN+OP (Type-5), IQL (Type-4), VDN+OP (Type-1), VDN (Type-4), IQL+OP (Type-3), VDN+OP (Type-1), VDN (Type-5) $\Big\}$.

The below are the set of 20 held-out agents that we use for across method evaluation in Tables 5.3 and 5.4. Figure 5.2 shows the cross-play matrix of Inter-CP agents.

Inter-CP : $\Big\{$ IQL (Type-1), IQL (Type-3), IQL+OP (Type-4), IQL+OP (Type-5), VDN+AUX (Type-2), VDN+AUX (Type-3), SAD+OP (Type-3), SAD+OP (Type-1), SAD+OP+AUX (Type-3), SAD+OP+AUX (Type-1), SAD+AUX (Type-3), SAD+AUX (Type-1), SAD (Type-3), SAD (Type-2), IQL+AUX (Type-3), IQL+AUX (Type-1), VDN (Type-4), VDN (Type-2), VDN+OP (Type-5), VDN+OP(Type-4) $\Big\}$.

5.2. Lifelong learning benchmarking

We implement some standard lifelong learning algorithms that are both replay-based and regularization-based.

Naive: This is the simplest algorithm in which the *learner* is trained sequentially on subsequent tasks, starting with the learned parameters at the end of the previous task, without any episodic memory or regularization.

Experience Replay (ER): We follow the procedure described in (Chaudhry et al., 2019) for implementing ER. We sample a mini-batch B_k from the current task (in which the *learner* plays with partner k), and a mini-batch B_m that consists of an equal number of samples from all previous tasks collectively. These mini-batches are stacked and a single gradient step is used to update the *learner*. Our implementation closely resembles the ring buffer strategy described in (Chaudhry et al., 2019), as there's equal representation from all previous tasks when sampling B_m, although the samples within each task itself are prioritized.

Averaged Gradient Episodic Memory (A-GEM): Minibatches B_k and B_m are sampled as described in (Chaudhry et al., 2018) and similar to ER. The gradients corresponding to these mini-batches are first computed denoted by g and g_{ref} respectively. If $g^T g_{ref} \geq 0$, then the gradient of the current task g is directly used to update the *learner*'s parameters, whereas if $g^T g_{ref} < 0$, g is first projected such that $g^T g_{ref} = 0$ before updating the *learner*. This projection ensures that the average loss over the previous tasks does not increase.

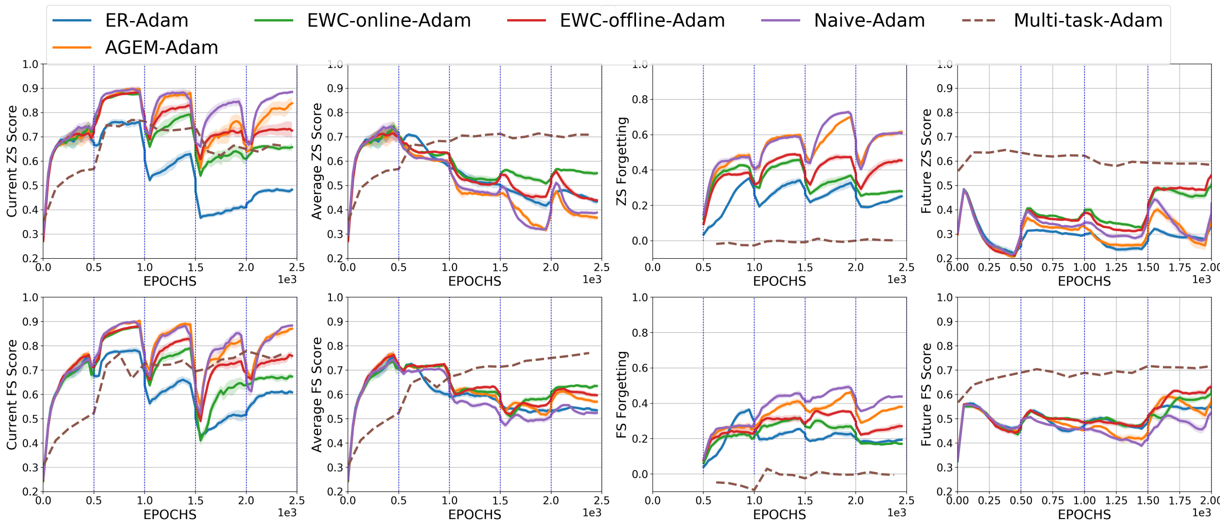


Fig. 5.3. Zero-shot (top row) and Few-shot (bottom row) performance of different LLL algorithms with Adam optimizer on *hard* task. From left to right: current score (↑), average score (↑), forgetting (↓), and average future score (↑) respectively. (↑ = higher better, ↓ = lower better).

Table 5.2. Benchmarking LLL methods on Hanabi. Average accuracy and forgetting of LLL algorithms on *hard* task averaged over 5 runs with 5000 games. ($\uparrow$ = higher better, $\downarrow$ = lower better)

Method	Zero-shot		Few-shot	
	$A_T \uparrow$	$F_T \downarrow$	$A_T \uparrow$	$F_T \downarrow$
Naive Adam	0.39	0.60	0.52	0.44
EWC off. Adam	0.45	0.45	0.60	0.27
EWC on. Adam	**0.55**	0.28	**0.63**	0.17
ER Adam	0.44	0.26	0.53	0.20
AGEM Adam	0.38	0.62	0.57	0.38
Naive SGD	0.52	0.15	0.52	0.14
EWC off. SGD	0.49	0.12	0.50	0.11
EWC on. SGD	0.47	0.12	0.48	0.11
ER SGD	0.50	**0.05**	0.50	**0.04**
AGEM SGD	0.50	0.13	0.51	0.12
Multi-Task Adam	0.70	0	0.77	0
Multi-Task SGD	0.50	0	0.51	0

Elastic Weight Consolidation (EWC): EWC is a regularization-based technique proposed to alleviate catastrophic forgetting by selectively reducing the plasticity of weights drawing inspiration from Bayesian methods (Kirkpatrick et al., 2017). EWC uses Fisher information matrix as a surrogate for the importance of learned weights and uses that for gradient updates. Offline EWC uses one Fisher matrix per task, therefore the number of regularization terms increases linearly with the number of tasks whereas Online EWC (Schwarz et al., 2018) uses only one Fisher matrix that is computed based on all the previous tasks. We consider both these variants in our benchmark.

Stable naive/ER/A-GEM/EWC: Mirzadeh et al. (2020) show that catastrophic forgetting can be mitigated through careful design of training regimes such as learning rate decay, batch size, dropout, and optimizer that can widen the tasks' local minima. The resulting model with these optimal choices is referred to as "stable". In particular, we consider if using larger batches, exponential learning rate decay, dropout (either in feed-forward

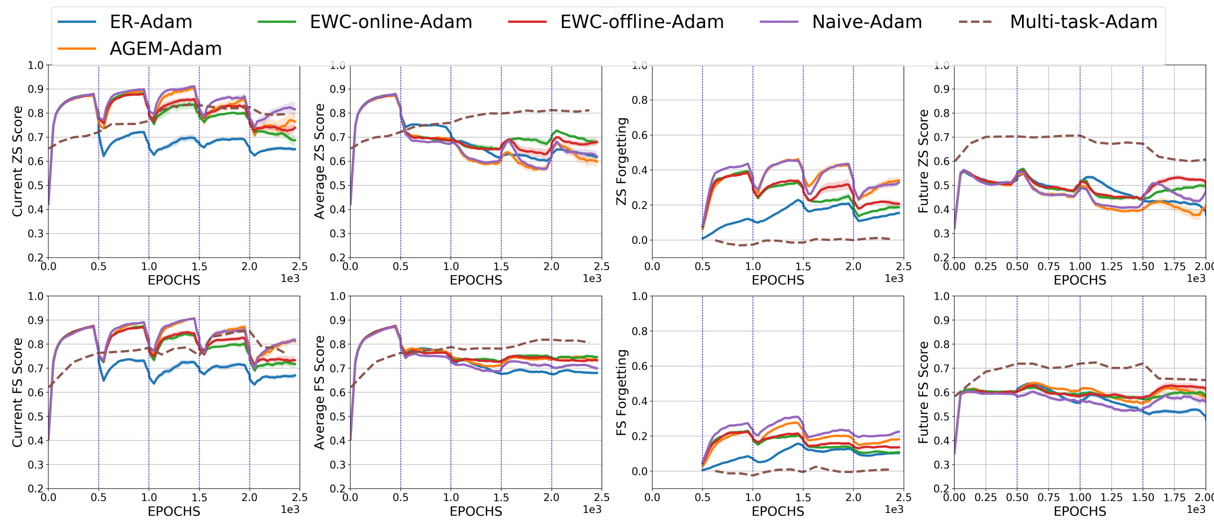


Fig. 5.4. Zero-shot (top row) and Few-shot (bottom row) performance of different LLL algorithms with Adam optimizer on *easy* task. From left to right: current score (↑), average score (↑), forgetting (↓), and average future score (↑) respectively. (↑ = higher better, ↓ = lower better).

or recurrent layers in R2D2), and SGD optimizer help improve continual training, thereby leading to better final performance as well as generalization to unseen agents.

Multi-Task Learning (MTL): In this setting, there's a common replay-buffer that contains the experiences of the *learner* interacting with all its partners. Mini-batches sampled from this common replay buffer are used for training the *learner*. This serves as an upper-bound on the achievable performance in our benchmark.

Some key observations from Figures 5.3, 5.4, 5.5, 5.6 and Table 5.2:

- We can observe from Figure 5.3 and Table 5.2 that online EWC with Adam has the best average score in both the zero-shot and few-shot setting among the lifelong learning algorithms.
- Forgetting is least for ER with SGD.
- Table 5.2 also shows the effect of the optimizer on different lifelong learning algorithms. Lifelong learning algorithms with SGD tend to have comparatively less forgetting and better zero-shot performance on average.

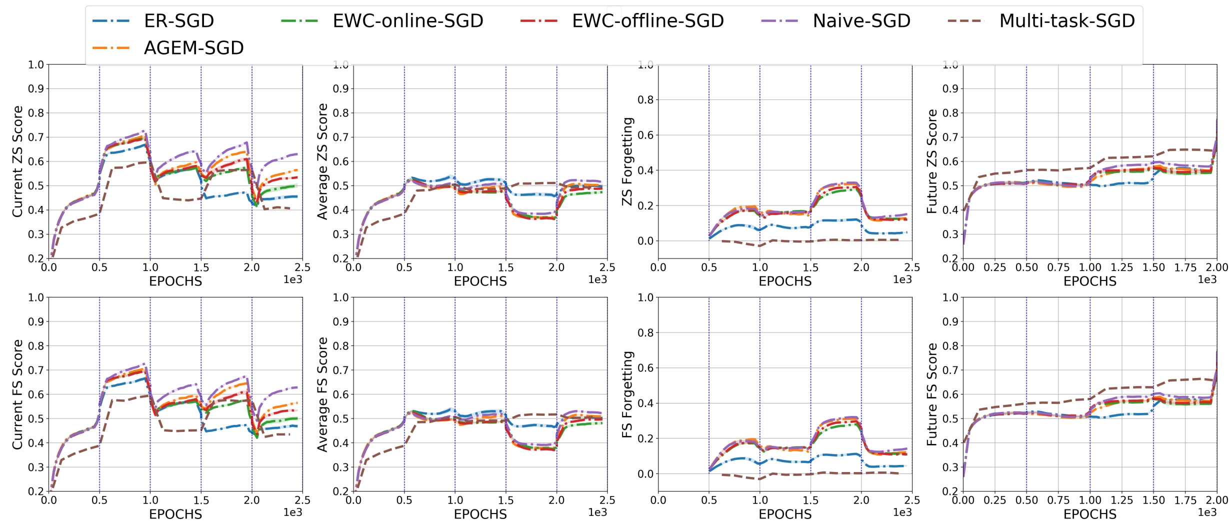


Fig. 5.5. Zero-shot (top row) and Few-shot (bottom row) performance of different LLL algorithms with SGD optimizer on *hard* task. From left to right: current score (↑), average score (↑), forgetting (↓), and average future score (↑) respectively. (↑ = higher better, ↓ = lower better).

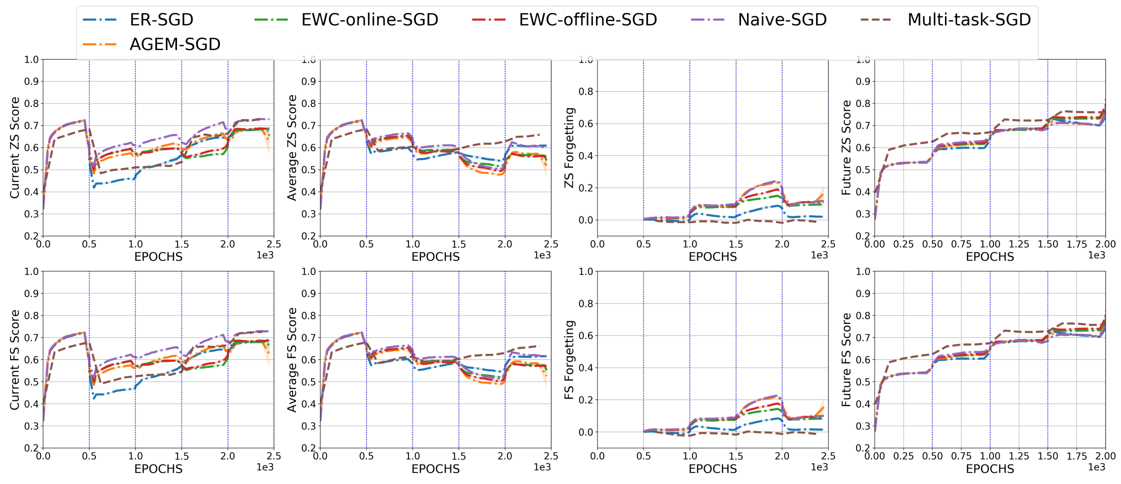


Fig. 5.6. Zero-shot (top row) and Few-shot (bottom row) performance of different LLL algorithms with SGD optimizer on *easy* task. From left to right: current score (↑), average score (↑), forgetting (↓), and average future score (↑) respectively. (↑ = higher better, ↓ = lower better).

- We can infer from Figure 5.7 that using Adam helps in fast adaptation to current task albeit at the expense of greater forgetting. This effect can also be seen in higher average few shot scores (Table 5.2).

- As one might expect, MTL with Adam achieves the highest average score in both zero-shot and few-shot as MTL has access to all the partners during training, which is not a feasible option in general.
- From the last column of Figure 5.3, we can observe that when the *learner* start playing with a new partner, there is an increase in average future score suggesting it has learned some useful skills from previous tasks that is transferable to the other partners. However, with more training, the *learner* possibly overfits to its current partner, leading to a drop in average future scores.

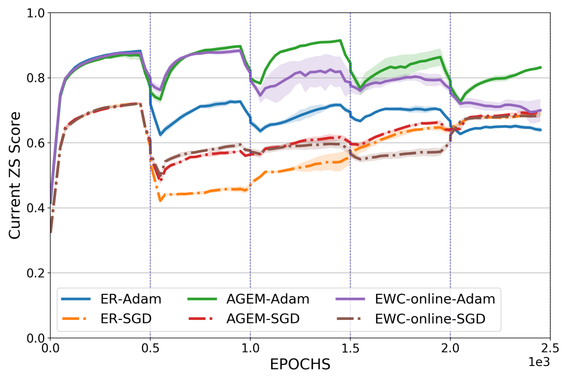
To explore the effect of different training regimes on continual training, we study the effect of larger batches (128 vs 32), learning rate decay with high initial rates (0.2/0.02), and dropout.

Our experiments suggest that *Lifelong Hanabi* does not benefit greatly from the use of large batches as the gain in scores is negligible. This observation aligns with the “stable” networks (Mirzadeh et al., 2020) that suggests using small batches. In the case of EWC, we find that there is a stark difference in performance with different λ values (the weight assigned to the Fischer term). Our experiments indicate larger λ is beneficial. Figures 5.4, 5.5, 5.6 contains the training curves of lifelong learning algorithms for other settings — *easy* task with Adam, *hard* task with SGD and *easy* task with SGD respectively. For all these figures, the *learner* is pre-trained with IQL method and is continually trained with either *easy* or *hard* agents mentioned in section 5.1. The sequential order of partners were chosen at random from the pre-trained pool in both *easy* and *hard* setting. Careful curation of the partner ordering and its effect on lifelong learning is left as future work.

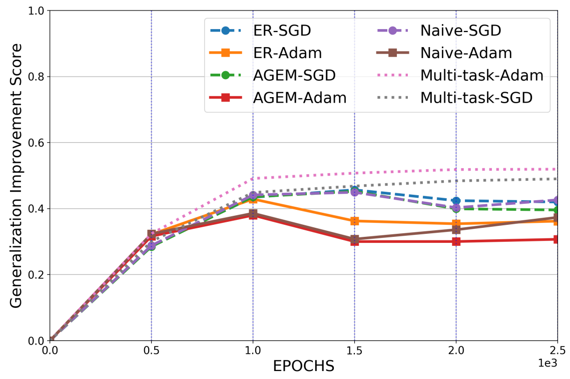
5.3. Lifelong learning under constrained memory and compute

5.3.1. Episodic memory size

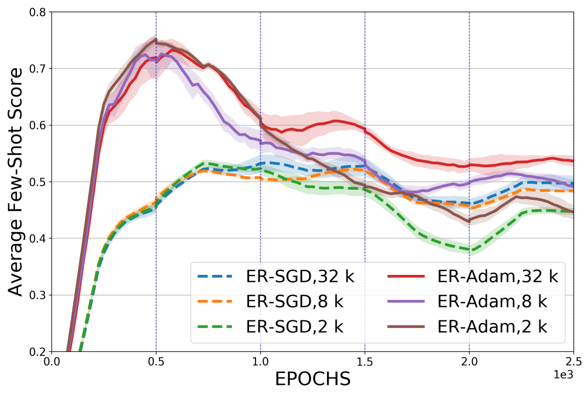
In order to understand the effect of episodic memory on the performance of memory-based lifelong learning algorithms, we vary the episodic memory size ({2k, 8k, 32k} $\times$ number of tasks) in the case of ER with both SGD and Adam as shown in Figure 5.7. In both these cases, a larger episodic memory size results in a better final performance.



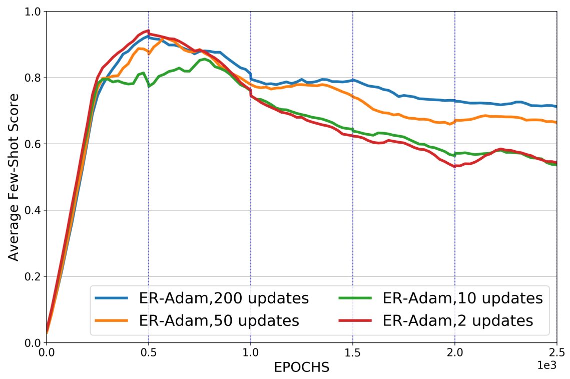
(a) Adam vs SGD



(b) GIS



(c) Episodic memory size



(d) Few-shot gradient steps

Fig. 5.7. More experiments: Generalization score with Inter-CP agents at the end of every task during continual training. LLL algorithms using SGD as an optimizer have better generalization performance compared to Adam. ER SGD has the highest GIS.

5.3.2. Gradient updates for few-shot evaluation

To better understand the ability of the *learner* trained with different lifelong learning algorithms to adapt quickly to all its partners, we vary the number of gradient steps used to update the *learner* in the few-shot evaluation scenario. As it can be seen from Figure 5.7, there is a considerable difference between 10 and 50 gradient updates on the final performance, whereas the benefit reaped beyond 50 updates is minimal.

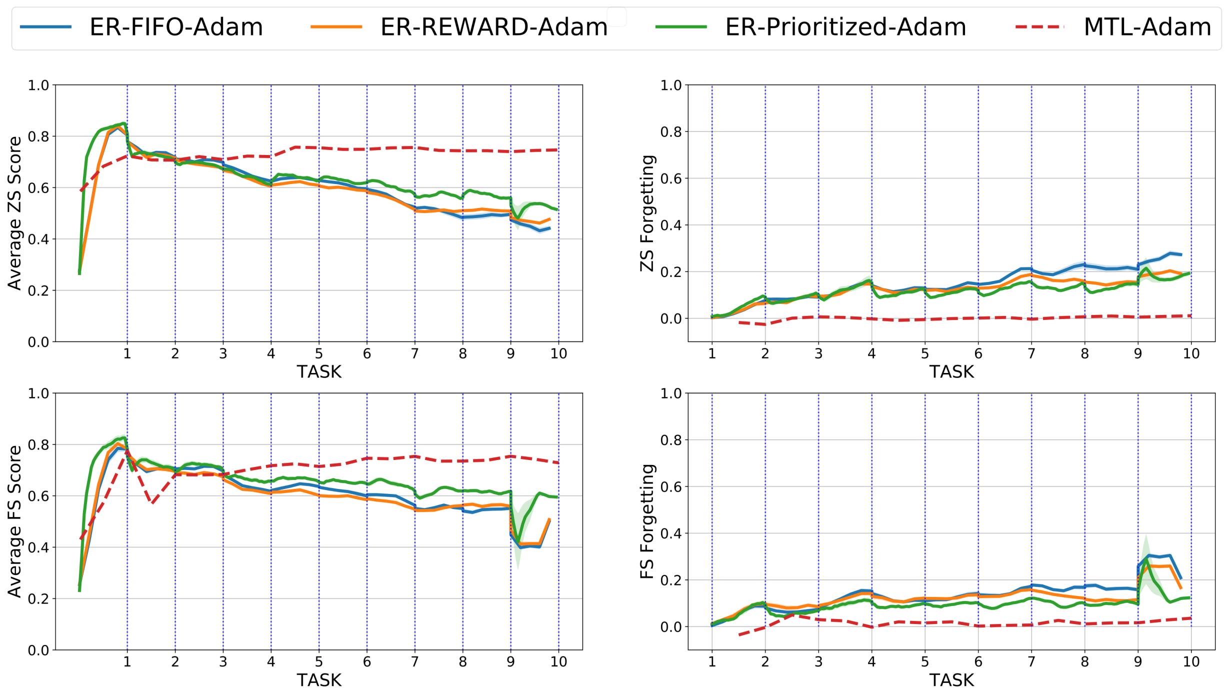


Fig. 5.8. Zero-shot (top row) and Few-shot (bottom row) performance of ER methods with different types of episodic memory designed for lifelong RL Isele and Cosgun (2018) with Adam optimizer on 10 tasks.

5.4. Lifelong RL methods

Isele and Cosgun (2018) propose some strategies for storing experiences in the replay-buffer that have been shown to reduce catastrophic forgetting in RL. All our methods use prioritized replay buffer that resembles the *surprise* strategy (Isele and Cosgun, 2018). In addition, we also compare this with *FIFO* and *Reward* strategies. For *FIFO*, we set the prioritization exponent α to 0 (Schaul et al., 2015), which is equivalent to uniformly sampling. In case of *Reward*, we do prioritized sampling that favors experiences based on the absolute value of reward instead of TD-error as done in our default case. As can be seen in Figure 5.8, ER with prioritized sampling performs best compared to *Reward* and *FIFO* strategies in terms of both average score and average forgetting. Implementing other sampling strategies such as *Global Distribution Matching* and *Coverage Maximization* are left for future work.

5.5. Zero-shot coordination

We compare our best-performing lifelong learning algorithms with recent MARL methods that have shown good performance on Hanabi (Table 5.3). In addition to reporting self-play evaluation scores, we evaluate each training method with two sets of unseen partners under zero-shot coordination scenario: (1) *Intra-CP* - a set consisting of agents that are trained with the same MARL method as the training method. For example, the SAD+OP agent is evaluated with other SAD+OP agents only, but with different architectures and seeds. Similarly, in order to evaluate agents trained in our setup, we evaluate them with other agents that are trained with the same MARL method as the *learner*, (2) *Inter-CP* (section 5.1) - a set containing 20 agents across all the MARL methods we consider.

Table 5.3. BEST : Comparison with other MARL algorithms on self-play (SP), cross-play evaluation scores within method (*Intra-CP*), and across different methods (*Inter-CP*). C: centralized training, GA: agents share their greedy action along with their standard action, L: true labels of cards needed, SYM: symmetries of the game needed upfront, P: require access to some pre-trained agents in sequence, UP: Having access to all the fixed pre-trained agents at the same time. ($\uparrow$ / $\downarrow$ = Difference in score after continual training, red: pre-trained with MARL method, blue: trained continually with LLL method, * : results obtained using models released by (Hu et al., 2020))

Training Method	SP	Intra-CP	Inter-CP	Limitations
SAD	23.85 ± 0.03	7.70 ± 0.69	14.60 ± 0.24	C + GA
SAD + AUX	23.57 ± 0.03	20.97 ± 0.80	18.51 ± 0.23	C + GA + L
SAD + OP	24.14 ± 0.03	10.10 ± 0.87	16.09 ± 0.25	C + Sym + GA
SAD + AUX + OP	23.40 ± 0.04	21.23 ± 0.25	17.77 ± 0.23	C + Sym + L + GA
SAD*	23.97 ± 0.04	$2.52 \pm 0.0.34$	11.46 ± 0.35	C + GA
SAD + AUX*	24.09 ± 0.03	17.65 ± 0.69	17.60 ± 0.42	C + GA + L
SAD + OP*	23.93 ± 0.02	15.32 ± 0.65	17.50 ± 0.34	C + Sym + GA
SAD + AUX + OP*	24.06 ± 0.02	22.07 ± 0.11	17.45 ± 0.38	C + Sym + L + GA
IQL + ER	20.91 ± 0.05 ($\downarrow$ 2.98)	15.73 ± 0.39 ($\uparrow$ **7.06**)	16.32 ± 0.21 ($\uparrow$ **8.09**)	P
IQL + AUX + ER	22.34 ± 0.06 ($\downarrow$ 1.46)	20.90 ± 0.06 ($\downarrow$ 0.15)	$\mathbf{19.17 \pm 0.22}$ ($\uparrow$ **1.33**)	L + P
IQL + Multi-task	20.93 ± 0.09 ($\downarrow$ 2.96)	16.05 ± 0.30 ($\uparrow$ **7.38**)	$\mathbf{17.88 \pm 0.17}$ ($\uparrow$ **9.65**)	UP

As we can observe from Table 5.3, although recent MARL methods can achieve good scores in SP and *Intra-CP* evaluations, they fail to achieve high scores in *Inter-CP* highlighting their inability to coordinate effectively with other MARL methods in the zero-shot scenario. We can observe that agents trained in our setup have significant improvement in both *Inter-CP* and *Intra-CP* compared to the agent at the start of continual learning, however, their SP scores are lower than at the start. The difference in scores due to continual training is indicated in brackets. It is also worth mentioning that IQL+AUX+ER achieve a better *Inter-CP* score than even other MARL methods, although this comes at a cost of slight reduction to *Intra-CP* score.

All the training methods in Table 5.3 have some limitations that we highlight now. During training, SAD allows agents to have access to the greedy action of their team mates in addition to the actual exploratory action chosen (GA). AUX refers to having an auxiliary task that predicts the *learner*'s own hand and hence requires ground truth labels for this (L). OP requires symmetries of the game to be known beforehand (SYM). IQL+ER and IQL+AUX+ER require pre-trained agents in sequence for lifelong learning (P), while IQL + Multi-Task requires access to all pre-trained agents simultaneously (UP). Figure 5.7 shows the progression in generalization performance (*Inter-CP*) after every task during continual training for several LLL algorithms. MTL (with Adam) and ER (with SGD) have the best scores with the *Inter-CP* agents at the end of continual training. However, MTL needs to interact with all the partners' at the same time which is not always a realistic assumption.

In order to obtain the *Intra-CP* scores for the existing MARL methods in the Table 5.3 (referenced as BEST in caption), we take the agent from each training method that performs best with the **Inter-CP** agents listed above in section 5.1 and evaluate them with the other 9 agents of the same method from the pretrained pool (Figure 4.3). However, in order to obtain the *Intra-CP* scores for each MARL method in the Table 5.4 (referenced as AVG in caption), we pick one agent, evaluate it with the rest (barring itself) and repeat the same for all other agents. The average of these scores are reported. A similar process is followed for reporting *Inter-CP* scores. The method of evaluating our LLL methods remains consistent in all the Tables (5.3, 5.4). For IQL+ER, we start with the IQL agent that has the least cross-play score and train it with *hard* agents sequentially using ER algorithm. In the case of IQL+AUX+ER, we start with an IQL agent that is pre-trained with AUX and is continually trained with the *hard* agents using ER algorithm. This continually trained agent is then evaluated with 9 other agents in either IQL or IQL+AUX respectively in order to obtain *Intra-CP* scores. However, please note that the auxiliary task is used only during pre-training and is not used during continual training. Note that the middle row in the Table 5.3 is generated using the latest models released by Hu et al. (2020).

Table 5.4. AVG : Comparison with other MARL algorithms on self-play (SP), cross-play evaluation scores within method (*Intra-CP*), and across different methods (*Inter-CP*). C: centralized training, GA: agents share their greedy action along with their standard action, L: true labels of cards needed, SYM: symmetries of the game needed upfront, P: require access to some pre-trained agents in sequence, UP: Having access to all the fixed pre-trained agents at the same time. (↑ / ↓ = Difference in score after continual training red: pre-trained with MARL method, blue: trained continually with LLL method). Refer text for difference between AVG and BEST.

Training method	SP	Intra-CP	Inter-CP	Limitations
SAD	23.78 ± 0.03	4.38 ± 0.66	8.40 ± 0.23	C+GA
SAD+AUX	23.82 ± 0.02	21.15 ± 0.26	17.01 ± 0.22	C+GA+L
SAD+OP	23.67 ± 0.03	12.00 ± 0.86	12.79 ± 0.24	C+Sym+GA
SAD+AUX+OP	23.88 ± 0.03	22.01 ± 0.03	17.08 ± 0.22	C+Sym+L+GA
IQL + ER	20.91 ± 0.05 (↓ 2.98)	15.73±0.39 (↑ **7.06**)	16.32±0.21 (↑ **8.09**)	P
IQL+AUX + ER	22.34± 0.06 (↓ 1.46)	20.90± 0.06 (↓0.15)	**19.17±0.22**(↑**1.33**)	L+P
IQL + Multi-task	20.93±0.09 (↓ 2.96)	16.05± 0.30(↑ **7.38**)	**17.88**±0.17 (↑ **9.65**)	UP

5.6. Additional results on 10 tasks

Figures 5.9, 5.10 shows the performance of ER, EWC online and offline, AGEM and Naive when the learner is trained with 10 partners (exact agent types previously mentioned in section 5.1). We can again observe that Adam EWC online and offline have the best average scores in both zero-shot and few-shot settings while the forgetting is least for ER SGD. This is consistent with what we observed in case of 5 tasks for both *hard* and *easy* agents.

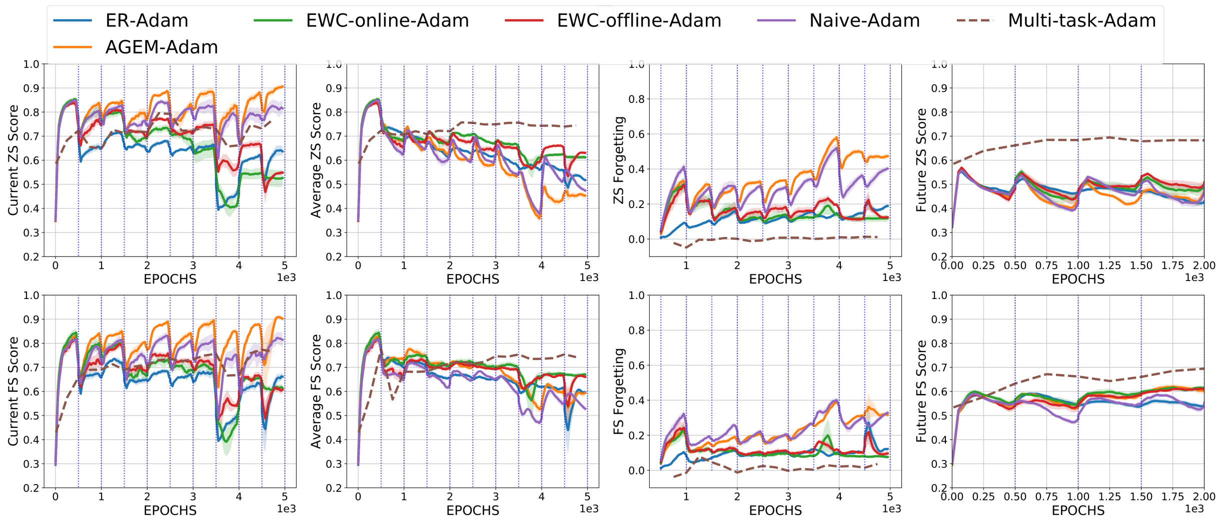


Fig. 5.9. Zero-shot (top row) and Few-shot (bottom row) performance of different LLL algorithms with Adam optimizer on 10 tasks. From left to right: current score, average score, forgetting and average future score respectively.

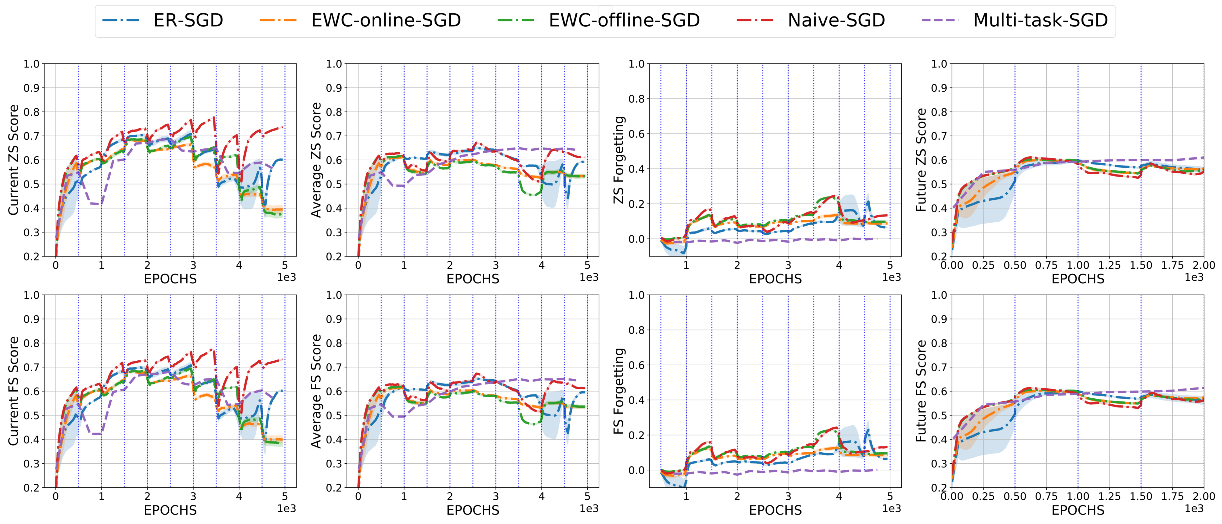


Fig. 5.10. Zero-shot (top row) and Few-shot (bottom row) performance of different LLL algorithms with SGD optimizer on 10 tasks. From left to right: current score, average score, forgetting and average future score respectively.

Chapter 6

Conclusions and Future work

Conclusions: In this thesis, we proposed *Lifelong Hanabi*, a new challenging benchmark for lifelong RL in particular, and lifelong learning in general, as the boundaries between supervised lifelong learning and lifelong RL are somewhat blurry. This benchmark tries to address some of the major drawbacks of previous lifelong learning benchmarks such as simplicity of tasks and lack of a measure of task correlation. The non-stationarity in our benchmark was introduced through agents having different strategies instead of synthetic modifications to the environment or the agent, while *cross-play* score served as an easy metric to quantify the similarity between tasks. By benchmarking several popular lifelong learning methods, we hope that this serves as a starting point in understanding drawbacks with existing methods and enabling researchers to come up with better lifelong learning algorithms. In Hanabi, we also showed that our lifelong learned agent generalizes well to unseen agents trained with the same MARL method as well as across different MARL methods, performing comparably to previous state-of-the-art methods that require explicit additional knowledge of the environment. We hope that the lifelong RL community adopts this as a standard benchmark for evaluating algorithmic advances due to its ease of use.

Future work: *Lifelong Hanabi* aims to facilitate the development of novel algorithms for lifelong learning specific to RL (i.e. lifelong RL). This framework also serves as a step towards thinking beyond centralized training in MARL. Some interesting future directions are to understand the kind of policies learned by the agents trained in our setup through policy visualization to see what kind of conventions (if any) emerges. It is also valuable to evaluate our trained agents with humans, as developing artificial agents capable of coordinating effectively with humans is an important long-term goal of modern AI. Moreover, exploiting recent advances in Meta-RL such as (Zintgraf et al., 2019) for faster adaptation in the *few-shot* evaluation setting, instead of naively fine-tuning can lead to agents that adapt well in the *ad-hoc* scenario. We believe studying the effect of the order of *partners* that the *learner* encounters and its effect on final performance is an interesting next step. Currently, the non-stationarity across different strategies is what we only exploit to design LLL tasks. This already resulted in an interesting trade-off for the learner between adapting to new partners and not forgetting to coordinate well with previous partners. Our preliminary experiments suggest that extending our framework to learning partners is extremely difficult for current methods, however, this could be an exciting future research direction.

References

[1] Mohamed Abdelsalam, Mojtaba Faramarzi, Shagun Sodhani, and Sarath Chandar. Iirc: Incremental implicitly-refined classification. *arXiv preprint arXiv:2012.12477*, 2020.

[2] Maruan Al-Shedivat, Trapit Bansal, Yuri Burda, Ilya Sutskever, Igor Mordatch, and Pieter Abbeel. Continuous adaptation via meta-learning in nonstationary and competitive environments. *arXiv preprint arXiv:1710.03641*, 2017.

[3] Rahaf Aljundi, Francesca Babiloni, Mohamed Elhoseiny, Marcus Rohrbach, and Tinne Tuytelaars. Memory aware synapses: Learning what (not) to forget. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 139–154, 2018.

[4] Thomas Anthony, Tom Eccles, Andrea Tacchetti, János Kramár, Ian Gemp, Thomas C Hudson, Nicolas Porcel, Marc Lanctot, Julien Pérolat, Richard Everett, et al. Learning to play no-press diplomacy with best response policy iteration. *arXiv preprint arXiv:2006.04635*, 2020.

[5] Antreas Antoniou, Massimiliano Patacchiola, Mateusz Ochal, and Amos Storkey. Defining benchmarks for continual few-shot learning. *arXiv preprint arXiv:2004.11967*, 2020.

[6] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE transactions on pattern analysis and machine intelligence*, 39(12):2481–2495, 2017.

[7] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[8] Nolan Bard, Jakob N Foerster, Sarath Chandar, Neil Burch, Marc Lanctot, H Francis Song, Emilio Parisotto, Vincent Dumoulin, Subhodeep Moitra, Edward Hughes, et al. The hanabi challenge: A new frontier for ai research. *Artificial Intelligence*, 280:103216,

2020.

[9] Tamer Başar and Geert Jan Olsder. *Dynamic noncooperative game theory*. SIAM, 1998.

[10] Amy Bearman, Olga Russakovsky, Vittorio Ferrari, and Li Fei-Fei. What's the point: Semantic segmentation with point supervision. In *European conference on computer vision*, pages 549–565. Springer, 2016.

[11] Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013.

[12] Richard Bellman. Dynamic programming. Technical report, RAND CORP SANTA MONICA CA, 1956.

[13] James Bergstra, Dan Yamins, David D Cox, et al. Hyperopt: A python library for optimizing the hyperparameters of machine learning algorithms. In *Proceedings of the 12th Python in science conference*, volume 13, page 20. Citeseer, 2013.

[14] Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemysław Dębiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, et al. Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*, 2019.

[15] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. Yolov4: Optimal speed and accuracy of object detection. *arXiv preprint arXiv:2004.10934*, 2020.

[16] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[17] Noam Brown and Tuomas Sandholm. Superhuman ai for heads-up no-limit poker: Libratus beats top professionals. *Science*, 359(6374):418–424, 2018.

[18] Gail A Carpenter and Stephen Grossberg. A massively parallel architecture for a self-organizing neural pattern recognition machine. *Computer vision, graphics, and image processing*, 37(1):54–115, 1987.

[19] Sarath Chandar, Chinnadhurai Sankar, Eugene Vorontsov, Samira Ebrahimi Kahou, and Yoshua Bengio. Towards non-saturating recurrent units for modelling long-term dependencies. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 3280–3287, 2019.

[20] Arslan Chaudhry, Puneet K Dokania, Thalaiyasingam Ajanthan, and Philip HS Torr. Riemannian walk for incremental learning: Understanding forgetting and intransigence. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 532–547, 2018.

[21] Arslan Chaudhry, Marc'Aurelio Ranzato, Marcus Rohrbach, and Mohamed Elhoseiny. Efficient lifelong learning with a-gem. *arXiv preprint arXiv:1812.00420*, 2018.

[22] Arslan Chaudhry, Marcus Rohrbach, Mohamed Elhoseiny, Thalaiyasingam Ajanthan, Puneet K Dokania, Philip HS Torr, and Marc'Aurelio Ranzato. On tiny episodic memories in continual learning. *arXiv preprint arXiv:1902.10486*, 2019.

[23] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

[24] Caroline Claus and Craig Boutilier. The dynamics of reinforcement learning in cooperative multiagent systems. *AAAI/IAAI*, 1998(746-752):2, 1998.

[25] Karl Cobbe, Oleg Klimov, Chris Hesse, Taehoon Kim, and John Schulman. Quantifying generalization in reinforcement learning. In *International Conference on Machine Learning*, pages 1282–1289. PMLR, 2019.

[26] Alexis Conneau, Holger Schwenk, Loïc Barrault, and Yann Lecun. Very deep convolutional networks for text classification. *arXiv preprint arXiv:1606.01781*, 2016.

[27] Martin Danelljan, Goutam Bhat, Fahad Shahbaz Khan, and Michael Felsberg. Eco: Efficient convolution operators for tracking. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 6638–6646, 2017.

[28] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[29] Thinh Doan, Siva Maguluri, and Justin Romberg. Finite-time analysis of distributed td (0) with linear function approximation on multi-agent reinforcement learning. In *International Conference on Machine Learning*, pages 1626–1635. PMLR, 2019.

[30] Lasse Espeholt, Raphaël Marinier, Piotr Stanczyk, Ke Wang, and Marcin Michalski. Seed rl: Scalable and efficient deep-rl with accelerated central inference. *arXiv preprint arXiv:1910.06591*, 2019.

[31] Mojtaba Faramarzi, Mohammad Amini, Akilesh Badrinaaraayanan, Vikas Verma, and Sarath Chandar. Patchup: A regularization technique for convolutional neural networks. *arXiv preprint arXiv:2006.07794*, 2020.

[32] Chrisantha Fernando, Dylan Banarse, Charles Blundell, Yori Zwols, David Ha, Andrei A Rusu, Alexander Pritzel, and Daan Wierstra. Pathnet: Evolution channels gradient descent in super neural networks. *arXiv preprint arXiv:1701.08734*, 2017.

[33] Jerzy Filar and Koos Vrieze. *Competitive Markov decision processes*. Springer Science & Business Media, 2012.

[34] Jakob Foerster, Gregory Farquhar, Triantafyllos Afouras, Nantas Nardelli, and Shimon Whiteson. Counterfactual multi-agent policy gradients. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[35] Jakob Foerster, Francis Song, Edward Hughes, Neil Burch, Iain Dunning, Shimon Whiteson, Matthew Botvinick, and Michael Bowling. Bayesian action decoder for deep multi-agent reinforcement learning. In *International Conference on Machine Learning*, pages 1942–1951, 2019.

[36] Jianfeng Gao, Michel Galley, and Lihong Li. Neural approaches to conversational ai. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*, pages 1371–1374, 2018.

[37] Golnaz Ghiasi, Tsung-Yi Lin, and Quoc V Le. Dropblock: A regularization method for convolutional networks. *arXiv preprint arXiv:1810.12890*, 2018.

[38] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.

[39] Ian J Goodfellow, Mehdi Mirza, Da Xiao, Aaron Courville, and Yoshua Bengio. An empirical investigation of catastrophic forgetting in gradient-based neural networks. *arXiv preprint arXiv:1312.6211*, 2013.

[40] Sai Krishna Gottipati, Boris Sattarov, Sufeng Niu, Yashaswi Pathak, Haoran Wei, Shengchao Liu, Simon Blackburn, Karam Thomas, Connor Coley, Jian Tang, et al. Learning to navigate the synthetically accessible chemical space using reinforcement learning. In *International Conference on Machine Learning*, pages 3668–3679. PMLR, 2020.

[41] Varun Gulshan, Lily Peng, Marc Coram, Martin C Stumpe, Derek Wu, Arunachalam Narayanaswamy, Subhashini Venugopalan, Kasumi Widner, Tom Madams, Jorge Cuadros, et al. Development and validation of a deep learning algorithm for detection of diabetic retinopathy in retinal fundus photographs. *Jama*, 316(22):2402–2410, 2016.

[42] Hongyu Guo, Yongyi Mao, and Richong Zhang. Mixup as locally linear out-of-manifold regularization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 3714–3722, 2019.

[43] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[44] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017.

[45] Peter Henderson, Wei-Di Chang, Florian Shkurti, Johanna Hansen, David Meger, and Gregory Dudek. Benchmark environments for multitask learning in continuous domains. *arXiv preprint arXiv:1708.04352*, 2017.

[46] Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal processing magazine*, 29(6):82–97, 2012.

[47] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[48] Dan Horgan, John Quan, David Budden, Gabriel Barth-Maron, Matteo Hessel, Hado Van Hasselt, and David Silver. Distributed prioritized experience replay. *arXiv preprint arXiv:1803.00933*, 2018.

[49] Ronald A Howard. Dynamic programming and markov processes. 1960.

[50] Hengyuan Hu and Jakob N Foerster. Simplified action decoder for deep multi-agent reinforcement learning. *arXiv preprint arXiv:1912.02288*, 2019.

[51] Hengyuan Hu, Adam Lerer, Alex Peysakhovich, and Jakob Foerster. " other-play" for zero-shot coordination. *arXiv preprint arXiv:2003.02979*, 2020.

[52] David Isele and Akansel Cosgun. Selective experience replay for lifelong learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[53] Simon Jégou, Michal Drozdzal, David Vazquez, Adriana Romero, and Yoshua Bengio. The one hundred layers tiramisu: Fully convolutional densenets for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pages 11–19, 2017.

[54] Christos Kaplanis, Murray Shanahan, and Claudia Clopath. Continual reinforcement learning with complex synapses. In *International Conference on Machine Learning*, pages 2497–2506. PMLR, 2018.

[55] Christos Kaplanis, Murray Shanahan, and Claudia Clopath. Policy consolidation for continual reinforcement learning. *arXiv preprint arXiv:1902.00255*, 2019.

[56] Steven Kapturowski, Georg Ostrovski, John Quan, Remi Munos, and Will Dabney. Recurrent experience replay in distributed reinforcement learning. In *International conference on learning representations*, 2018.

[57] Tero Karras, Samuli Laine, Miika Aittala, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. Analyzing and improving the image quality of stylegan. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8110–8119, 2020.

[58] Michał Kempka, Marek Wydmuch, Grzegorz Runc, Jakub Toczek, and Wojciech Jaśkowski. Vizdoom: A doom-based ai research platform for visual reinforcement learning. In *2016 IEEE Conference on Computational Intelligence and Games (CIG)*, pages 1–8. IEEE, 2016.

[59] Khimya Khetarpal, Matthew Riemer, Irina Rish, and Doina Precup. Towards continual reinforcement learning: A review and perspectives. *arXiv preprint arXiv:2012.13490*, 2020.

[60] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 114(13):3521–3526, 2017.

[61] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25:1097–1105, 2012.

[62] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, 2017.

[63] Ankit Kumar, Ozan Irsoy, Peter Ondruska, Mohit Iyyer, James Bradbury, Ishaan Gulrajani, Victor Zhong, Romain Paulus, and Richard Socher. Ask me anything: Dynamic memory networks for natural language processing. In *International conference on machine learning*, pages 1378–1387. PMLR, 2016.

[64] Guillaume Lample, Alexis Conneau, Ludovic Denoyer, and Marc'Aurelio Ranzato. Unsupervised machine translation using monolingual corpora only. *arXiv preprint arXiv:1711.00043*, 2017.

[65] Zhizhong Li and Derek Hoiem. Learning without forgetting. *IEEE transactions on pattern analysis and machine intelligence*, 40(12):2935–2947, 2017.

[66] Vincenzo Lomonaco and Davide Maltoni. Core50: a new dataset and benchmark for continuous object recognition. *arXiv preprint arXiv:1705.03550*, 2017.

[67] Vincenzo Lomonaco, Karan Desai, Eugenio Culurciello, and Davide Maltoni. Continual reinforcement learning in 3d non-stationary environments. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pages 248–249, 2020.

[68] David Lopez-Paz and Marc'Aurelio Ranzato. Gradient episodic memory for continual learning. In *Advances in neural information processing systems*, pages 6467–6476, 2017.

[69] Marlos C Machado, Marc G Bellemare, Erik Talvitie, Joel Veness, Matthew Hausknecht, and Michael Bowling. Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents. *Journal of Artificial Intelligence Research*, 61:523–562, 2018.

[70] Arun Mallya and Svetlana Lazebnik. Packnet: Adding multiple tasks to a single network by iterative pruning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7765–7773, 2018.

[71] Michael McCloskey and Neal J Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of learning and motivation*,

volume 24, pages 109–165. Elsevier, 1989.

[72] Seyed Iman Mirzadeh, Mehrdad Farajtabar, Razvan Pascanu, and Hassan Ghasemzadeh. Understanding the role of training regimes in continual learning. *Advances in Neural Information Processing Systems*, 33, 2020.

[73] Tom M Mitchell. Machine learning. 1997. *Burr Ridge, IL: McGraw Hill*, 45(37): 870–877, 1997.

[74] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[75] Arvind Neelakantan, Luke Vilnis, Quoc V Le, Ilya Sutskever, Lukasz Kaiser, Karol Kurach, and James Martens. Adding gradient noise improves learning for very deep networks. *arXiv preprint arXiv:1511.06807*, 2015.

[76] Hadi Nekoei, Akilesh Badrinaaraayanan, Aaron Courville, and Sarath Chandar. Continuous coordination as a realistic scenario for lifelong learning. *arXiv preprint arXiv:2103.03216*, 2021.

[77] Cuong V Nguyen, Yingzhen Li, Thang D Bui, and Richard E Turner. Variational continual learning. *arXiv preprint arXiv:1710.10628*, 2017.

[78] Frans A Oliehoek and Christopher Amato. *A concise introduction to decentralized POMDPs*. Springer, 2016.

[79] Afshin OroojlooyJadid and Davood Hajinezhad. A review of cooperative multi-agent deep reinforcement learning. *arXiv preprint arXiv:1908.03963*, 2019.

[80] Philip Paquette, Yuchen Lu, Steven Bocco, Max O Smith, Satya Ortiz-Gagné, Jonathan K Kummerfeld, Satinder Singh, Joelle Pineau, and Aaron Courville. No press diplomacy: Modeling multi-agent gameplay. *arXiv preprint arXiv:1909.02128*, 2019.

[81] Emmanouil Antonios Platanios, Abulhair Saparov, and Tom Mitchell. Jelly bean world: A testbed for never-ending learning. *arXiv preprint arXiv:2002.06306*, 2020.

[82] Ameya Prabhu, Philip HS Torr, and Puneet K Dokania. Gdumb: A simple approach that questions our progress in continual learning. In *Proceedings of the European Conference on Computer Vision*, 2020.

[83] David Premack and Guy Woodruff. Does the chimpanzee have a theory of mind? *Behavioral and brain sciences*, 1(4):515–526, 1978.

[84] Neil Rabinowitz, Frank Perbet, Francis Song, Chiyuan Zhang, SM Ali Eslami, and Matthew Botvinick. Machine theory of mind. In *International conference on machine learning*, pages 4218–4227. PMLR, 2018.

[85] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *OpenAI blog*, 1 (8):9, 2019.

[86] Tabish Rashid, Mikayel Samvelyan, Christian Schroeder, Gregory Farquhar, Jakob Foerster, and Shimon Whiteson. Qmix: Monotonic value function factorisation for deep multi-agent reinforcement learning. In *International Conference on Machine Learning*, pages 4295–4304. PMLR, 2018.

[87] Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, Georg Sperl, and Christoph H Lampert. icarl: Incremental classifier and representation learning. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 2001–2010, 2017.

[88] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *IEEE transactions on pattern analysis and machine intelligence*, 39(6):1137–1149, 2016.

[89] Mark B Ring. Child: A first step towards continual learning. In *Learning to learn*, pages 261–292. Springer, 1998.

[90] Ryne Roady, Tyler L. Hayes, Hitesh Vaidya, and Christopher Kanan. Stream-51: Streaming classification and novelty detection from videos. In *The IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2020.

[91] Andrei A Rusu, Neil C Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks. *arXiv preprint arXiv:1606.04671*, 2016.

[92] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520, 2018.

[93] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.

[94] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897. PMLR, 2015.

[95] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[96] Jonathan Schwarz, Wojciech Czarnecki, Jelena Luketina, Agnieszka Grabska-Barwinska, Yee Whye Teh, Razvan Pascanu, and Raia Hadsell. Progress & compress: A scalable framework for continual learning. In *International Conference on Machine Learning*, pages 4528–4537. PMLR, 2018.

[97] Xuesong Shi, Dongjiang Li, Pengpeng Zhao, Qinbin Tian, Yuxin Tian, Qiwei Long, Chunhao Zhu, Jingwei Song, Fei Qiao, Le Song, Yangquan Guo, Zhigang Wang, Yimin Zhang, Baoxing Qin, Wei Yang, Fangshi Wang, Rosa H. M. Chan, and Qi She. Are we ready for service robots? the openloris-scene datasets for lifelong slam, 2020.

[98] Yoav Shoham and Kevin Leyton-Brown. *Multiagent systems: Algorithmic, game-theoretic, and logical foundations.* Cambridge University Press, 2008.

[99] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

[100] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *nature*, 550(7676):354–359, 2017.

[101] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018.

[102] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[103] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[104] Stefan Stojanov, Samarth Mishra, Ngoc Anh Thai, Nikhil Dhanda, Ahmad Humayun, Chen Yu, Linda B Smith, and James M Rehg. Incremental object learning from contiguous views. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8777–8786, 2019.

[105] Peter Sunehag, Guy Lever, Audrunas Gruslys, Wojciech Marian Czarnecki, Vinicius Zambaldi, Max Jaderberg, Marc Lanctot, Nicolas Sonnerat, Joel Z Leibo, Karl Tuyls, et al. Value-decomposition networks for cooperative multi-agent learning. *arXiv preprint arXiv:1706.05296*, 2017.

[106] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks, 2014.

[107] Richard S Sutton. Learning to predict by the methods of temporal differences. *Machine learning*, 3(1):9–44, 1988.

[108] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[109] Ming Tan. Multi-agent reinforcement learning: Independent vs. cooperative agents. In *Proceedings of the tenth international conference on machine learning*, pages 330–337, 1993.

[110] Gerald Tesauro. Td-gammon, a self-teaching backgammon program, achieves master-level play. *Neural computation*, 6(2):215–219, 1994.

[111] Gerald Tesauro. Temporal difference learning and td-gammon. *Communications of the ACM*, 38(3):58–68, 1995.

[112] Sebastian Thrun. Lifelong learning algorithms. In *Learning to learn*, pages 181–209. Springer, 1998.

[113] Gido M Van de Ven and Andreas S Tolias. Three scenarios for continual learning. *arXiv preprint arXiv:1904.07734*, 2019.

[114] Vladimir Vapnik. *The nature of statistical learning theory*. Springer science & business media, 2013.

[115] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017.

[116] Vikas Verma, Alex Lamb, Christopher Beckham, Amir Najafi, Ioannis Mitliagkas, David Lopez-Paz, and Yoshua Bengio. Manifold mixup: Better representations by

interpolating hidden states. In *International Conference on Machine Learning*, pages 6438–6447. PMLR, 2019.

[117] Qiang Wang, Li Zhang, Luca Bertinetto, Weiming Hu, and Philip HS Torr. Fast online object tracking and segmentation: A unifying approach. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1328–1338, 2019.

[118] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4): 279–292, 1992.

[119] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

[120] Mengdi Xu, Wenhao Ding, Jiacheng Zhu, Zuxin Liu, Baiming Chen, and Ding Zhao. Task-agnostic online reinforcement learning with an infinite mixture of gaussian processes, 2020.

[121] Zichao Yang, Diyi Yang, Chris Dyer, Xiaodong He, Alex Smola, and Eduard Hovy. Hierarchical attention networks for document classification. In *Proceedings of the 2016 conference of the North American chapter of the association for computational linguistics: human language technologies*, pages 1480–1489, 2016.

[122] Friedemann Zenke, Ben Poole, and Surya Ganguli. Continual learning through synaptic intelligence. *Proceedings of machine learning research*, 70:3987, 2017.

[123] Amy Zhang, Nicolas Ballas, and Joelle Pineau. A dissection of overfitting and generalization in continuous reinforcement learning. *arXiv preprint arXiv:1806.07937*, 2018.

[124] Hongyi Zhang, Moustapha Cisse, Yann N Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. *arXiv preprint arXiv:1710.09412*, 2017.

[125] Kaiqing Zhang, Zhuoran Yang, Han Liu, Tong Zhang, and Tamer Basar. Fully decentralized multi-agent reinforcement learning with networked agents. In *International Conference on Machine Learning*, pages 5872–5881. PMLR, 2018.

[126] Kaiqing Zhang, Zhuoran Yang, and Tamer Başar. Multi-agent reinforcement learning: A selective overview of theories and algorithms. *arXiv preprint arXiv:1911.10635*, 2019.

[127] Yizhe Zhang, Siqi Sun, Michel Galley, Yen-Chun Chen, Chris Brockett, Xiang Gao, Jianfeng Gao, Jingjing Liu, and Bill Dolan. Dialogpt: Large-scale generative pre-training for conversational response generation. *arXiv preprint arXiv:1911.00536*, 2019.

[128] Luisa Zintgraf, Kyriacos Shiarli, Vitaly Kurin, Katja Hofmann, and Shimon Whiteson. Fast context adaptation via meta-learning. In *International Conference on Machine Learning*, pages 7693–7702. PMLR, 2019.

6.1. All hyperparameters and experiment details

In this section, we present list of common and important hyperparameters used in our experiments along with their description, while also presenting algorithm-specific hyperparameters.

Table 6.1. All common hyperparameters and their description.

HYPERPARAMETERS	VALUE	DESCRIPTION
batchsize	32	BATCHSIZE USED FOR BOTH TRAINING AND FEW-SHOT EVALUATION
max_train_steps	200M	MAXIMUM NUMBER OF TRAINING STEPS PER TASK
max_eval_steps	500K	MAXIMUM NUMBER OF TRAINING STEPS DURING FEW-SHOT EVALUATION
burn_in_frames	10K	NUMBER OF SAMPLES USED TO WARM-UP REPLAY BUFFER
eval_burn_in_frames	1K	NUMBER OF SAMPLES USED TO WARM-UP EVALUATION REPLAY BUFFER
replay_buffer_size	32768	REPLAY BUFFER SIZE DURING CONTINUAL TRAINING
eval_replay_buffer_size	10000	REPLAY BUFFER SIZE FOR FEW-SHOT EVALUATION
epoch_len_size	200	NUMBER OF GRADIENT UPDATES PER EPOCH
eval_epoch_len_size	50	NUMBER OF GRADIENT UPDATES FOR FEW-SHOT EVALUATION
eval_freq	25	LEARNER IS EVALUATED AFTER EACH 25 EPOCHS
num_thread	10	NUMBER OF THREADS USED FOR R2D2 ACTORS
num_game_per_thread	80	NUMBER OF GAME PER THREADS USED FOR R2D2 ACTORS
eval_num_thread	10	NUMBER OF THREADS USED FOR R2D2 ACTORS DURING FEW-SHOT EVALUATION
eval_num_game_per_thread	10	NUMBER OF GAMES PER THREADS USED FOR R2D2 ACTORS DURING FEW-SHOT EVALUATION
sgd_momentum	0.8	MOMENTUM FOR SGD OPTIMIZER

Table 6.2. Specific hyperparameters to each algorithm and their description

hyperparameters	value	Description
ewc_lambda	50000	EWC
ewc_gamma	1	EWC
$replay\_buffer\_size$	163840	Multi-task