Εθνικο Μετσοβιο Πολυτεχνειο
Σχολη Ηλεκτρολογων Μηχανικων και Μηχανικων Υπολογιστων
ΔΠΜΣ "Επιστημη Δεδομενων και Μηχανικη Μαθηση"

Count-based Agent Modelling in Multi-Agent Reinforcement Learning

Διπλωματικη Εργασια

των

ΚΩΝΣΤΑΝΤΙΝΟΥ ΠΑΠΑΘΑΝΑΣΙΟΥ
ΑΝΔΡΕΑ ΚΟΝΤΟΓΙΑΝΝΗ

Επιβλέπων: Γεώργιος Στάμου
Καθηγητής Ε.Μ.Π.

Αθήνα, Σεπτέμβριος 2023

Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
ΔΠΜΣ "Επιστημη Δεδομενων και Μηχανικη Μαθηση"

Count-based Agent Modelling in Multi-Agent Reinforcement Learning

Διπλωματικη Εργασια

των

ΚΩΝΣΤΑΝΤΙΝΟΥ ΠΑΠΑΘΑΝΑΣΙΟΥ
ΑΝΔΡΕΑ ΚΟΝΤΟΓΙΑΝΝΗ

Επιβλέπων: Γεώργιος Στάμου
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 7η Σεπτεμβρίου 2023.

(Υπογραφή)	*(Υπογραφή)*	*(Υπογραφή)*
.....	.....	.....
Γεώργιος Στάμου	Αθανάσιος Βουλόδημος	Ανδρέας Σταφυλοπάτης
Καθηγητής Ε.Μ.Π.	Επ. Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.

Αθήνα, Σεπτέμβριος 2023

(Υπογραφή)

.....
ΚΩΝΣΤΑΝΤΙΝΟΣ ΠΑΠΑΘΑΝΑΣΙΟΥ
ΑΝΔΡΕΑΣ ΚΟΝΤΟΓΙΑΝΝΗ

Διπλωματούχοι Ηλεκτρολόγοι Μηχανικοί και Μηχανικοί Υπολογιστών Ε.Μ.Π.

Ευχαριστίες

Θα θέλαμε να ευχαριστήσουμε τον κύριο Γεώργιο Στάμου, Καθηγητή της Σχολής Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών του Εθνικού Μετσόβιου Πολυτεχνείου και επιβλέποντα της παρούσας διπλωματικής εργασίας, για την εμπιστοσύνη που μας έδειξε αναθέτοντάς μας αυτό το θέμα, αλλά και για την άριστη επικοινωνία μας καθ' όλη την διάρκεια των μεταπτυχιακών μας σπουδών.

Θα θέλαμε επίσης να ευχαριστήσουμε θερμά τον κύριο Μιχάλη Ζαβλανό, Καθηγητή του Πανεπιστημίου Duke για την συνεπίβλεψη της παρούσας διπλωματικής, την εξαιρετική συνεργασία μας κατά την διάρκειά της, όπως και για την συνεχή καθογήση, η οποία ήταν καθοριστικής σημασίας για την ολοκλήρωσή της. Ακόμα, ευχαριστούμε τον Διδακτρορικό φοιτητή του Πανεπιστημίου Duke, Yi Shen για την προθυμία του και τις συμβουλές του σε τεχνικό επίπεδο.

Τέλος, ευχαριστούμε ιδιαίτερα τον Καθηγητή του Πανεπιστημίου Πειραιώς, κύριο Γεώργιο Βούρο, για την συνεπίβλεψη της διπλωματικής αυτής, καθώς και για τις πολύτιμες συμβουλές, όπως και για την άψογη συνεργασία μας.

Περίληψη

Σε αυτή τη διπλωματική, εξετάζουμε το πρόβλημα της Πολυπρακτορικής Ενισχυτικής Μάθησης (MARL) πάνω σε μερικώς παρατηρήσιμα συνεργατικά περιβάλλοντα. Λόγω της καλής τους επίδοσης σε πολλά περιβάλλοντα και της αποδοτικότητας της on-policy μάθησής τους, δίνουμε έμφαση σε αλγορίθμους πολιτικής κλίσης, όπως ο αλγόριθμος Multi-Agent Actor-Critic (MAA2C). Ακολουθώντας την πρόσφατη πληθώρα προσεγγίσεων που είτε (α) υιοθετούν το πρότυπο Κεντρικής Εκπαίδευσης - Αποκεντρωμένης Εκτέλεσης (CTDE), είτε (β) χρησιμοποιούν την επικοινωνία πρακτόρων κατά τη διάρκεια της εκτέλεσης για βελτιωμένη επίδοση, θέτουμε το ακόλουθο ερώτημα: πώς να συνδυάσουμε το πρότυπο CTDE με τα οφέλη των μεθόδων επικοινωνίας, για την εκπαίδευση πρακτόρων που είναι ικανοί να λύνουν δύσκολα περιβάλλοντα, συμπεριλαμβανομένων αυτών με αραιή επιβράβευση. Για τον σκοπό αυτό, σε αυτή τη διπλωματική προτείνουμε τον Count-based Agent Modelling (CAM), ένα πλαίσιο MARL που βασίζεται στον MAA2C και χρησιμοποιεί τεχνικές μοντελοποίησης πρακτόρων, μεταβαλλόμενη αναπαράσταση και αυτοεπιβλεπόμενη μάθηση για τη δημιουργία διαμοιρασμού της πληροφορίας ανάμεσα στους πράκτορες σε μορφή κρυφών νευρωνικών αναπαραστάσεων. Το CAM χρησιμοποιεί τις δημιουργημένες αναπαραστάσεις για την αποτελεσματική ενίσχυση των πολιτικών των πρακτόρων και για συντονισμένη αυτόνομη εξερεύνηση βασισμένη σε ενδογενή κινητοποίηση. Πειραματικά, δείχνουμε ότι το CAM υπερτερεί των πιο προηγμένων αλγορίθμων MARL σε δύσκολα περιβάλλοντα από τα σετ περιβαλλόντων προσομείωσης Multi-Agent Particle Environment (MPE) και Level-based Foraging (LBF).

Λέξεις Κλειδιά

Ενισχυτική Μάθηση, Πολυπρακτορική Μάθηση, Μεταβαλλόμενη Αναπαράσταση, Μοντελοποίηση Πρακτόρων, Ενδογενής Εξερεύνηση

Abstract

In this thesis, we consider the problem of Multi-Agent Reinforcement Learning (MARL) in partially-observable cooperative tasks. Due to their good performance on multiple benchmark MARL testbeds and the efficiency of their on-policy learning, we emphasize on policy gradient algorithms, such as the Multi-Agent Actor-Critic (MAA2C) algorithm. Following the recent surge of approaches that either (a) adopt the Centralized-Training-Decentralized Execution (CTDE) schema, or (b) utilize agent communication also during execution for improved performance, we address the question of how to combine the CTDE schema with the benefits of communication methods, in order to train agents able to perform better in difficult tasks, including those with sparse reward settings. To this aim, in this thesis, we propose *C*ount-based *A*gent *M*odelling *(CAM)*, a MARL framework, built on top of MAA2C, that utilizes agent modelling techniques, variational inference and self-supervised learning for generating information sharing among the agents as latent neural representations. CAM uses the generated information sharing representations for explicitly enhancing the agents' policies, and also for encouraging the agents towards coordinated exploration based on intrinsic motivation. Experimentally, we show that CAM outperforms state-of-the-art MARL algorithms on difficult tasks, with and without sparse rewards, from the Multi-Agent Particle Environment (MPE) and Level-based Foraging (LBF) benchmark testbeds.

Keywords

Reinforcement Learning, Multi-Agent Learning, Variational Inference, Agent Modelling, Intrinsic Exploration

Contents

List of Figures

List of Tables

Chapter 1

Εκτεταμένη Περίληψη στα Ελληνικά

1.1 Πολυπρακτορική Ενισχυτική Μάθηση

Σε αυτή τη διπλωματική, στοχεύουμε στην αντιμετώπιση πολύπλοκων περιορισμένων μερικώς παρατηρήσιμων περιβαλλόντων πολλαπλών πρακτόρων που απαιτούν αποτελεσματική συνεργασία από τους πράκτορες προκειμένου να λυθούν. Η συνεργασία μπορεί να εκδηλωθεί ως συντονισμός της γνώσης των πρακτόρων σχετικά με τη δυναμική του περιβάλλοντος και των ομαδικών τους στόχων. Από τη δομή των περιβαλλοντικών πειραμάτων που αντιμετωπίζουμε στα πειράματά μας, όλοι οι πράκτορες λαμβάνουν το ίδιο σήμα ανταμοιβής σε κάθε χρονική στιγμή της προσομοίωσης του περιβάλλοντος, το οποίο περιγράφει τον κοινό στόχο(υς) της ομάδας κατά μήκος ενός παιχνιδιού (πχ εάν οι παίκτες είναι κοντά στη νίκη του παιχνιδιού ή αν φτάνουν στα αγαπημένα τους σημεία αναφοράς). Το θεωρητικό πλαίσιο που μοντελοποιεί το παραπάνω πρόβλημα είναι το DecPOMDP. Επιπλέον, ακολουθούμε το σχήμα Κεντρικής Εκπαίδευσης - Αποκεντρωμένης Εκτέλεσης (CTDE), το οποίο έχει ευρέως χρησιμοποιηθεί για τον έλεγχο των αλγορίθμων Αποκεντρωμένης Ενισχυτικής Μάθησης (MARL) στη βιβλιογραφία. Με βάση τη δομή του DecPOMDP και την έννοια του σχήματος CTDE, εντοπίζουμε τα εξής προβλήματα που μας ενέπνευσαν έντονα για αυτήν τη διπλωματική.

Οι κοινές ανταμοιβές που λαμβάνουν οι πράκτορες σε ένα DecPOMDP μπορούν να παραπλανήσουν τη διαδικασία εκπαίδευσης των πρακτόρων, καθιστώντας πολλές εργασίες στο πλαίσιο του MARL πολύ δύσκολες να λυθούν. Δεδομένου ότι τόσο οι άμεσες ανταμοιβές όσο και οι συνολικοί στόχοι της ομάδας είναι κοινοί, είναι δύσκολο για κάθε πράκτορα να αναγνωρίσει τη συνεισφορά της δικής του πολιτικής στις ανταμοιβές που λαμβάνει. Αυτή η δυσκολία μπορεί να καθιστά ορισμένους πράκτορες "αδρανείς" στην αποτελεσματική εξερεύνηση του τοπικού χώρου παρατήρησής τους, ακόμη και αν η ανταμοιβή της ομάδας είναι σχετικά υψηλή λόγω της ύπαρξης καλών πολιτικών άλλων πρακτόρων. Συνεπώς, οι αδρανείς πράκτορες θα επιλέγουν ενέργειες που είναι μη βέλτιστες, επειδή η εκτιμώμενη τους αξία θα είναι μεγάλη κατά μέσο όρο. Με άλλα λόγια, αυτό το πρόβλημα μπορεί να οδηγήσει τους πράκτορες σε υποβέλτιστα προσεγγιστικά ισορροπία Nash, ένα πρόβλημα που έχει χαρακτηριστεί ως "Σχετική

Υπεργενίκευση" στη βιβλιογραφία του MARL. Το παραπάνω πρόβλημα επιδεινώνεται υπό την ύπαρξη ρυθμίσεων αραιών ανταμοιβών, στις οποίες οι πράκτορες λαμβάνουν καλές ανταμοιβές μόνο όταν φτάσουν σε καλές καταστάσεις, π.χ. όταν επιλύουν πλήρως το πρόβλημα. Σε τέτοιες περιπτώσεις, οι πράκτορες πρέπει να προτρέπονται να υιοθετήσουν πιο εξερευνητικές πολιτικές, με καθέναν από αυτούς να στοχεύει στην αποτελεσματική εξερεύνηση του τοπικού χώρου παρατήρησής του τόσο ατομικά όσο και συνεργατικά.

Σύμφωνα με το σχήμα CTDE, κατά τη δοκιμή, οι πράκτορες πρέπει να λύσουν την εργασία δίνοντας μόνο τις δικές τους παρατηρήσεις (καθώς η εκτέλεση είναι αποκεντρωμένη), αν και μπορούν να χρησιμοποιούν οποιεσδήποτε πληροφορίες από τους άλλους πράκτορες για συντονισμό κατά την εκμάθηση των ατομικών πολιτικών τους κατά την κεντρική εκπαίδευση. Ωστόσο, δεν είναι πάντα προφανές πώς οι πράκτορες να χρησιμοποιήσουν τις πληροφορίες των άλλων πρακτόρων κατά την κεντρική εκπαίδευση. Πρόσφατα, υπάρχει μεγάλο ενδιαφέρον για μεθόδους επικοινωνίας, οι οποίες στοχεύουν να χρησιμοποιούν αποδοτική κεντρική επικοινωνία μεταξύ των πρακτόρων μέσω μηνυμάτων και πρωτοκόλλων προκειμένου να λύσουν πολύπλοκες συνεργατικές εργασίες. Οι μέθοδοι επικοινωνίας αντιμετωπίζουν το πρόβλημα του συντονισμού των πρακτόρων ενσωματώνοντας στην πολιτική κάθε πράκτορα κάποιον τρόπο κοινοποίησης πληροφοριών (όπως κοινές παρατηρήσεις και/ή ενέργειες) μέσω της εκμάθησης ενός δικτύου επικοινωνίας των πρακτόρων που βελτιστοποιεί την επίδοση ολόκληρης της ομάδας. Με αυτόν τον τρόπο, κάθε πράκτορας μαθαίνει μια βελτιωμένη πολιτική που λαμβάνει υπόψη επίσης αυτό που παρατηρούν/κάνουν οι συνδεδεμένοι συμπαίκτες του. Ωστόσο, αυτοί οι τρόποι δεν βασίζονται στο σχήμα CTDE, διότι δεν χρησιμοποιούν αποκεντρωμένη εκτέλεση, ενώ οι πράκτορες χρησιμοποιούν πάντα κάποιες πληροφορίες από άλλους πράκτορες ως μέρος των πολιτικών τους κατά τις δοκιμές.

Ένα σημαντικό ερώτημα που ενδιαφερόμαστε να αντιμετωπίσουμε σε αυτήν τη διπλωματική είναι το εξής: *Μπορούμε να συνδυάσουμε το σχήμα CTDE με τα οφέλη των μεθόδων επικοινωνίας για να εκπαιδεύσουμε πράκτορες που είναι σε θέση να δρουν αποτελεσματικότερα σε δύσκολα προβλήματα;* Επιπλέον, λαμβάνοντας υπόψη την εντυπωσιακή επίδοση των μεθόδων επικοινωνίας σε δύσκολα περιβάλλοντα αξιολόγησης, ενδιαφερόμαστε επίσης να αντιμετωπίσουμε το ακόλουθο ερώτημα: *Μπορούμε να εκπαιδεύσουμε πράκτορες υπό το σχήμα CTDE με βελτιωμένες πολιτικές, δηλαδή εκμεταλλευόμενοι κάποιον τύπο κοινής κοινοποίησης πληροφοριών, και να χρησιμοποιήσουμε αυτήν την κοινοποίηση πληροφοριών για συντονισμένη διερεύνηση σε δύσκολα προβλήματα με αραιές ανταμοιβές;*

1.2 CAM: Learning Variational Embeddings for Opponent Modelling

Στην προτεινόμενη μέθοδο, κάθε πράκτορας υιοθετεί σύστημα μοντελοποίησης του αντιπάλου, στοχεύοντας να προβλέψει τις παρατηρήσεις των άλλων πρακτόρων σε ένα χρονικό βήμα, λαμβάνοντας υπόψη μόνο τις δικές του παρατηρήσεις. Όπως στο [54], κάθε πράκτορας i μοντελοποιεί τις παρατηρήσεις των άλλων πρακτόρων, που συμβολίζονται ως o_t^{-i}, μέσω μιας εσωτερικής πεποίθησης, $p(o_t^{-i} \mid o_t^i; \theta_i)$, παραμετροποιημένη από την τυχαία μεταβλητή Θ_i με

τιμές $\theta_i \in \mathbf{\Theta_i}$. Αυτή η πεποίθηση περιγράφει τον τρόπο με τον οποίο ο πράκτορας αντιλαμβάνεται τα μη ορατά τμήματα της πλήρους κατάστασης βασιζόμενος στις δικές του μερικές παρατηρήσεις. Υποθέτουμε (α) $p(\theta_i)$ για το θ_i, (β) την ύπαρξη ορισμένων κρυφών μεταβλητών z^i στο χώρο $\mathcal{Z}$ και (γ) ότι σε κάθε χρονικό βήμα t, οι κρυφές μεταβλητές z^i περιέχουν πληροφορίες σχετικά με την μοντελοποίηση του αντιπάλου από τον πράκτορα i.

Για να μάθουμε μια καλή αναπαράσταση (embedding) για την κρυφή μεταβλητή μοντελοποίησης του αντιπάλου z^i, χρησιμοποιούμε έναν μεταβολικό κωδικοποιητή (p_{θ_i}) - αποκωδικοποιητή (q_{ϕ_i}) για κάθε παίκτη που στοχεύει στο να αντιληφθεί τις παρατηρήσεις των άλλων παικτών (o_t^{-i}) στο χρονικό βήμα t, δεδομένων μόνο των δικών του παρατηρήσεων (o_t^i). Παράλληλα, σε κάθε χρονικό βήμα t, ενισχύουμε την πολιτική του παίκτη π_t^i προσθέτοντας την αρχική είσοδο της πολιτικής (δηλαδή o_t^i) με το z_t^i. Θα μπορούσαμε να υποθέσουμε ότι ιδανικά ελπίζουμε ότι ο κωδικοποιητής-αποκωδικοποιητής θα μπορούσε να προβλέπει τέλεια τις παρατηρήσεις των άλλων παικτών, έτσι ώστε το z_t^i να είναι όσο το δυνατόν πιο ενημερωτικό σχετικά με το τι παρατηρούν οι άλλοι παίκτες. Ωστόσο, όπως παρατηρήθηκε σε πρόσφατη έρευνα [25], η χρήση όλων των πληροφοριών της κατάστασης (ακόμα και όταν χρησιμοποιούμε μια συμπιεσμένη αναπαράσταση) για την ενίσχυση της πολιτικής του παίκτη δεν αυξάνει πάντα την επίδοση λόγω πληροφοριών της κατάστασης που είναι περιττές για τον παίκτη. Σημειώνουμε ότι παρόμοια αποτελέσματα παρατηρούνται και στα πειράματά μας σε επόμενο κεφάλαιο αυτής της διπλωματικής.

Με στόχο την εκμάθηση σημαντικών embeddings κατά την αναπαράσταση των z_t^i, αντιμετωπίζουμε τη χρήση περιττών πληροφοριών κατά την προσπάθεια πρόβλεψης από τον κωδικοποιητή-αποκωδικοποιητή του o_t^{-i}. Αντίστοιχα με το [25], χρησιμοποιούμε επιπλέον τη μάθηση βαρών, $w^i = \sigma(\phi_w(o^i))$, έχοντας ένα βάρος για κάθε χαρακτηριστικό του o^{-i}. Χρησιμοποιούμε ένα πολυεπίπεδο νευρωνικό δίκτυο ως τη συνάρτηση ϕ_w και τη σιγμοειδή συνάρτηση ενεργοποίησης σ προκειμένου να διατηρήσουμε την τιμή κάθε εισόδου μεταξύ $[0, 1]$. Στην ουσία, το w^i αντιπροσωπεύει τη σημασία κάθε χαρακτηριστικού του o^{-i} για τον παράγοντα i. Για κάθε παράγοντα i, εκπαιδεύουμε έναν κωδικοποιητή-αποκωδικοποιητή ελαχιστοποιώντας την ακόλουθη αυτο-επιβλεπόμενη συνάρτηση ανακατασκευής:

$$L_{rec}(\theta_i, \phi_i, \phi_w^i) = (\tilde{w}^i \cdot o^{-i} - w^i \cdot \hat{o}^{-i})^2 \tag{1.1}$$

Παραπάνω, η μεταβλητή $\hat{o}^{-i}$ αντιπροσωπεύει τις προβλεπόμενες παρατηρήσεις του κωδικοποιητή-αποκωδικοποιητή, ενώ η μεταβλητή $\hat{o}^{-i}$ αντιστοιχεί στις πραγματικές παρατηρήσεις στόχου των άλλων παραγόντων. Οι μεταβλητές w^i αντιπροσωπεύουν τα προβλεπόμενα βάρη φίλτρου, και το σύμβολο $\cdot$ υποδηλώνει τον element-wise πολλαπλασιασμό. Για να σταθεροποιήσουμε την εκπαίδευση των w^i, παρόμοια με τις τυπικές ενημερώσεις στο βαθύ ενισχυτικό μάθηση, όπως στο [51, 68], χρησιμοποιούμε βάρη, συμβολίζοντας τα με $\tilde{w}^i$, για να φιλτράρουμε τις προβλεπόμενες παρατηρήσεις στόχου $\hat{o}^{-i}$ με τιμές που αντιστοιχούν στις προηγούμενες τιμές των w^i και παραμένουν αμετάβλητες κατά τη διάρκεια της ενημέρωσης των w^i.

Η απώλεια ανακατασκευής στην 4.1 γενικεύει την τυπική απώλεια ανακατασκευής ενός Μεταβολικού Αυτοκωδικοποιητή (VAE) (δείτε 3.31). Συγκεκριμένα, εάν τόσο τα w^i όσο και τα $\tilde{w}^i$ είναι ίσα με τον πίνακα ταυτότητας, τότε λαμβάνουμε ακριβώς τον όρο απώλειας

ανακατασκευής του 3.31. Στη γενική περίπτωση, η απώλεια ανακατασκευής του 4.1 λαμβάνει υπόψη και το (α) αποτελεσματική ανακατασκευή του o^{-i} και το (β) τη μάθηση του w^i με έναν αυτό-εκπαιδευτικό τρόπο.

Για να διασφαλίσουμε ότι οι τιμές του w^i δεν εξαφανίζονται στο μηδέν σε όλες τις διαστάσεις (σημειώστε ότι σε αυτήν την περίπτωση, η απώλεια ανακατασκευής θα ήταν μηδέν), προσθέτουμε επίσης τον ακόλουθο όρο απώλειας L2 κανονικοποίησης:

$$L_{norm}(\phi_w^i) = -||w^i||^2 \tag{1.2}$$

Για να εξασφαλίσουμε επιπλέον ότι οι τιμές του w^i έχουν σημασία για την επίλυση της δεδομένης εργασίας και, συνεπώς, για τη μεγιστοποίηση της επίδοσης, προσθέτουμε τον παρακάτω όρο ρύθμισης για τον πράκτορα i, προερχόμενο από τον χαμένο κριτικού του τυπικού MAA2C.

$$L_{critic}(\phi_w^i, k_i) = [r_t + V_{k_i'}(\hat{s_t}) - V_{k_i}(\hat{s_t})]^2 \tag{1.3}$$

όπου r_t είναι η ληφθείσα ανταμοιβή στο χρονικό βήμα t, s_t είναι η κατάσταση του περιβάλλοντος στο χρονικό βήμα t, V_{k_i} είναι η κατάσταση αξίας παραμετροποιημένη από ένα νευρωνικό δίκτυο k_i, $V_{k'i}(s)$ είναι η κατάσταση αξίας παραμετροποιημένη από ένα δίκτυο στόχο $k'i$ και $\hat{s_t} = o_t \oplus (w^i \cdot o_t^{-1})$ είναι η προβλεπόμενη κατάσταση που δημιουργείται από την πεποίθηση του παίκτη i, όπου $\oplus$ σημαίνει διανυσματική συνένωση. Σημειώνουμε ότι το $Vk_i(\hat{s})$ (και, ως εκ τούτου, το $Vk'i(\hat{s})$) δεν εκπαιδεύεται ως βοηθητική εργασία, με την έννοια ότι το $Vk_i(\hat{s})$ χρησιμοποιείται επίσης ως κεντρικός critic του MAA2C και δεν εξαρτάται από την πραγματική κατάσταση s.

Όπως και στο VAE [37], προσθέτουμε έναν όρο ρύθμισης που στοχεύει στην ελαχιστοποίηση της KL απόκλισης μεταξύ της μετεπεξεργασίας (posterior) και της προτελεστικής (prior) κατανομής (η οποία συνήθως υποθέτεται ως η κανονική κατανομή).

$$L_{KL}(\phi_i) = D_{KL}(q_{\phi_i}(z^i|o^i)||p_{\theta_i}(z^i) = \mathcal{N}(0,1)) \tag{1.4}$$

Συνολικά, η συνολική απώλεια του CAM για τη μάθηση προαιρετικών ενσωματώσεων για τον πράκτορα i, που σημειώνεται με L_{TOTAL}, είναι η ακόλουθη:

$$L_{TOTAL}(\theta_i, \phi_i, \phi_w^i, k_i) = L_{rec}(\theta_i, \phi_i, \phi_w^i) + L_{norm}(\phi_w^i) + L_{critic}(\phi_w^i, k_i) + L_{KL}(\phi_i) \tag{1.5}$$

Ο κωδικοποιητής-αποκωδικοποιητής κάθε πράκτορα λαμβάνει τις παρατηρήσεις του πράκτορα i, o_t^i, σε κάθε χρονικό βήμα t και στοχεύει στο να μάθει χρήσιμα embeddings στη μορφή κρυφής αναπαράστασης, έχοντας ως στόχο τις παρατηρήσεις των άλλων πρακτόρων, o_t^{-i}, πολλαπλασιασμένες (σε στοιχειώδη βάση) με το διανυσματικό διάνυσμα βαρών w^i, προκειμένου να επιτρέψει σε κάθε πράκτορα να επικεντρώνεται μόνο στις πληροφορίες που μπορεί να είναι σημαντικές. Σε αυτό το σημείο, πρέπει να τονίσουμε τη σημασία του διανυσματικού διανύσματος βαρών, καθώς χωρίς αυτό, ο στόχος του κωδικοποιητή-αποκωδικοποιητή αυξάνεται

γραμμικά με τον αριθμό των πρακτόρων και τη διαδικασία μάθησης δεν αναμένεται να οδηγήσει σε χρήσιμη ενσωμάτωση. Τέλος, σε κάθε χρονικό βήμα t, κάθε πράκτορας i χρησιμοποιεί μια βελτιωμένη πολιτική $\pi^i(\cdot \mid o_t^i, z_t^i)$ που εξαρτάται τόσο από το o_t^i όσο και από την προβλεπόμενη μεταβαριατική ενσωμάτωση z_t^i.

Για να εκπαιδευτούν οι κωδικοποιητές-αποκωδικοποιητές των πρακτόρων με ανεξάρτητα (i.i.d) δεδομένα, στην πράξη δειγματοληπτούμε μεταβάσεις από τις ληφθείσες πορείες των πρακτόρων, προκειμένου να διακόψουμε την υψηλή συσχέτιση των μεταβάσεων εντός των ίδιων τροχιών. Σημειώνουμε ότι το βασικό μοντέλο MAA2C μας εκπαιδεύεται χρησιμοποιώντας δεδομένα εντός πολιτικής (on-policy).

1.3 CAM: Count-based Intrinsic Motivation

Σε αυτήν την ενότητα, παρουσιάζουμε το δεύτερο μέρος του CAM, το οποίο στοχεύει στον σχεδιασμό αποτελεσματικών επιπλέον ανταμοιβών για καλύτερη εξερεύνηση σε πολύπλοκα προβλήμα με αραιές ανταμοιβές. Ωστόσο, όπως συζητήσαμε παραπάνω, είναι σημαντικό για τους πράκτορες να συντονίζουν την εξερεύνησή τους με αποδοτικό τρόπο, δεδομένου ότι οι τυχαίες αρχικές πολιτικές (που οφείλονται στον όρο της εντροπίας στην απώλεια πολιτικής) μπορεί να μην είναι επιτυχείς στον εντοπισμό σημαντικών καταστάσεων. Αλλιώς, οι καταστάσεις υψηλής αξίας μπορεί να απαιτούν μια ακολουθία κοινών ενεργειών από τους πράκτορες που είναι πολύ δύσκολο να βρεθούν μέσω τυχαίων ασυντόνιστων πολιτικών.

Για να ενισχύσουμε περαιτέρω τη συνδυαστική πολιτική των πρακτόρων για την επίτευξη καταστάσεων υψηλής αξίας σε περιβάλλοντα με αραιές ανταμοιβές, εκμεταλλευόμαστε την εκφραστική δύναμη των προβλεπόμενων z^i, που έχουν ενσωματώσει την πεποίθηση του πράκτορα i σχετικά με τη μερική παρατηρήσιμη πλήρη κατάσταση και σχεδιάζουμε εσωτερικές ανταμοιβές που λαμβάνουν υπόψη τόσο το *ατομικό* όσο και το *συνεργατικό* όφελος. Καθώς το z^i περιλαμβάνει πληροφορίες σχετικά με το τι βλέπουν οι άλλοι παίκτες, προτείνουμε να ενθαρρύνουμε κάθε πράκτορα i να φτάσει σε παρατηρήσεις που οδηγούν σε νέα z^i και, συνεπώς, να εξερευνήσει εκδοχές του τοπικού χώρου κατάστασης (παρατήρηση) που σχετίζονται με διάφορα z^i, για το ατομικό του όφελος. Ωστόσο, με αυτόν τον τρόπο, ο πράκτορας i προκαλεί επίσης διαφορετικούς στόχους για τη μοντελοποίηση των άλλων πρακτόρων, $-i$ δηλαδή, οι άλλοι πράκτορες, $-i$, πρέπει τώρα να μάθουν από αόρατους στόχους μοντελοποίησης πρακτόρων λόγω νέων παρατηρήσεων που λαμβάνουν από τον πράκτορα i. Συνεπώς, ο πράκτορας i αναγκάζει έμμεσα τους άλλους πράκτορες, $-i$, να εξερευνήσουν καλύτερα τον χώρο στόχων της μοντελοποίησής τους, βοηθώντας τους να βελτιώσουν τις πεποιθήσεις τους, δηλαδή το z^{-i}, σχετικά με τον πλήρη χώρο κατάστασης. Από την άλλη πλευρά, με το να κάνουν και οι άλλοι πράκτορες, $-i$, εξερευνήσεις σε περιοχές του τοπικού χώρου κατάστασης που μπορεί να οδηγήσουν σε πιο ενημερωτικά z^i, η προτεινόμενη προσέγγιση ενθαρρύνει αλληλεπίδραση για την *συντονισμένη εξερεύνηση* για συνεργατικό όφελος.

Για να εξασφαλίσουμε τους στόχους μας, υιοθετούμε το σχήμα ανταμοιβής ενσωματωμένης συνάρτησης καταμέτρησης με χρήση του αλγορίθμου SimHash που προτάθηκε στην εργασία [66]. Η περιγραφή του αλγορίθμου SimHash μπορεί να βρεθεί στον Αλγόριθμο 8. Γενικά,

Algorithm 1 Count-based exploration through static hashing, using SimHash

Define state preprocessor $g : S \to \mathbb{R}^D$

Initialize $A \in \mathbb{R}^{k \times D}$ with entries drawn i.i.d. from the standard Gaussian $\mathcal{N}(0, 1)$

Initialize a hash table with values $n(\cdot) \equiv 0$

For each iteration j **do**

Collect a set of state-action samples $\{(s_m, a_m)\}_{m=0}^{M}$ with policy π

Compute hash codes using SimHash, $\phi(s_m) = \text{sgn}(Ag(s_m))$

Update the hash table counts $\forall m : 0 \leq m \leq M$ as $n(H(s_m)) \leftarrow n(H(s_m)) + 1$

Update the policy π using rewards $\left\{ r(s_m, a_m) + \frac{\beta}{\sqrt{n(H(s_m))}} \right\}_{m=0}^{M}$ with any RL algorithm

End for

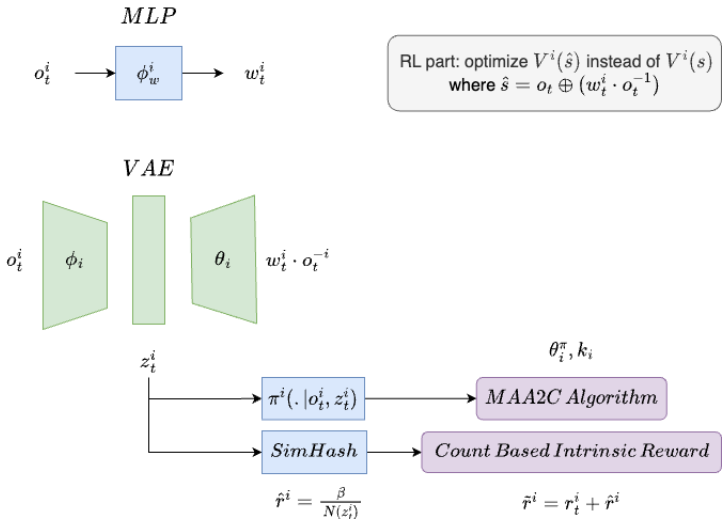

Figure 1.1: The Count-Based Agent Modelling (CAM) framework

παρουσιάζουμε την επισκόπηση του προτεινόμενου πλαισίου στο Σχήμα 4.1. Ο αλγόριθμος SimHash διακριτοποιεί τις αναπαραστάσεις (ή το χώρο καταστάσεων γενικά) χρησιμοποιώντας μια συνάρτηση κατακερματισμού $H : S \to \mathbb{Z}$. Στη συνέχεια, ένα μπόνους εξερεύνησης $\hat{r} : S \to \mathbb{R}$ υπολογίζεται ως εξής:

$$\hat{r} = \frac{\beta}{\sqrt{n(H(s_m))}} \tag{1.6}$$

Ο όρος $\beta \in \mathbb{R}_{\geq 0}$ περιγράφει το συντελεστή μπόνους, ενώ οι αρχικές μετρήσεις $n(\cdot)$ ορίζονται στο μηδέν για όλο το εύρος του H. Κατά τη διάρκεια της εκπαίδευσης, για κάθε επισκεφθείσα ενσωμάτωση o_t^i στη χρονική στιγμή t, το $n(H(o_t^i))$ αυξάνεται κατά ένα, και ο παίκτης i εκπαιδεύεται με ανταμοιβές $\tilde{r}_t^i = (r_t^i + \hat{r}_t^i)$, όπου r_t^i είναι η αρχική ανταμοιβή του παίκτη i (που είναι ίδια για όλους τους παίκτες σε συνεργατικά παιχνίδια) και $\hat{r}_t^i$ είναι η εγγενής ανταμοιβή του παίκτη i.

Η επίδοση του αλγορίθμου εξαρτάται σε μεγάλο βαθμό από την επιλογή της συνάρτησης κατακερματισμού H, η οποία θα έπρεπε ιδανικά να αντιστοιχεί διαφορετικά απομακρυσμένα embeddings και να συγχωνεύει παρόμοια. Η απόφαση λαμβάνεται από τον αλγόριθμο SimHash, ο οποίος μετρά την ομοιότητα με βάση τη γωνιακή απόσταση χρησιμοποιώντας την ακόλουθη συνάρτηση:

$$H(s_m) = \text{sgn}(Ag(s_m)) \in \{-1, 1\}^k, \tag{1.7}$$

όπου $g : S \to \mathbb{R}^D$ αποτελεί μια συνάρτηση προεπεξεργασίας και το A είναι ένας πίνακας $k \times D$ με ανεξάρτητες καταχωρήσεις από την κανονική κατανομή $\mathcal{N}(0, 1)$.

Επιλέξαμε τον αλγόριθμο SimHash, ο οποίο στην πράξη εκτελεί μια απλή μαθηματική μετασχηματιση της εισόδου αντί για μια πιο λεπτομερή προσέγγιση, λόγω της απλότητάς του και της ικανότητάς του να επιτρέπει σε κοντινές εισόδους να μετατρέπονται σε κοντινές τιμές κατακερματισμού.

Για να καταστήσουμε τις εσωτερικές ανταμοιβές πιο σταθερές και να καθοδηγήσουμε αποτελεσματικά την εξερεύνηση καθ' όλη τη διάρκεια της διαδικασίας εκπαίδευσης του κάθε πράκτορα, πρέπει να καταστήσουμε το z^i πιο σταθερό. Δηλαδή, οι παράμετροι των κωδικοποιητών-αποκωδικοποιητών να μην αλλάζουν δραματικά μεταξύ διαδοχικών επεισοδίων εκπαίδευσης. Για τον σκοπό αυτό, επιλέγουμε να πραγματοποιούμε περιοδικές ενημερώσεις των κωδικοποιητών-αποκωδικοποιητών χρησιμοποιώντας ένα σταθερό μεγάλο αριθμό επεισοδίων εκπαίδευσης ως περίοδο ενημέρωσης, η οποία αποτελεί μια υπερπαράμετρο του setting μας. Παραθέτουμε την επισκόπηση του αλγορίθμου CAM στον Αλγόριθμο 9.

Algorithm 2 CAM algorithm

Initialize n actor networks with random parameters $\theta_1^\pi, \ldots, \theta_n^\pi$
Initialize n critic network with random parameters $k_1, \ldots, k_n$
Initialize n VAE networks with random parameters $\theta_1, \ldots, \theta_n$ and $\phi_1, \ldots, \phi_n$
Initialize n weight networks with random parameters $\phi_w^1, \ldots, \phi_w^n$
Collect environment observations $o_1^1, \ldots, o_1^n$
for time step $t = 1, \ldots,$ EpisodeHorizon **do**
  for agent $i = 1, \ldots, n$ **do**
    Sample embedding z_i^t from variational encoder q_{ϕ_i}
    Sample actions a_i^t from $\pi(a_t^i \mid h_t^i, z_t^i; \theta_i^\pi)$
Execute actions and collect states s_{t+1}, observations o_{t+1}^i and rewards r_t^i.
for time step $t = 1, \ldots,$ EpisodeHorizon **do**
  for agent $i = 1, \ldots, n$ **do**
    Calculate the count-based intrinsic reward $\hat{r}_t^i$ based on algorithm 8
    and use $\tilde{r}_i = r_t^i + \hat{r}_t^i$ as the reward of each agent
for agent $i = 1, \ldots, n$ **do**
  Update parameters θ_i^π by minimizing the loss of the equation 3.8
  Update parameters k_i by minimizing the loss of the equation 4.3
for agent $i = 1, \ldots, n$ **do**
  Update parameters $\phi_w^i, \theta_i, \phi_i$ by minimizing the loss of the equation 4.5

1.4 Πειραματικό Μέρος

Σε αυτό το κεφάλαιο συζητούμε την αξιολόγηση του προτεινόμενου πλαισίου (CAM) χρησιμοποιώντας (α) το multi-agent particle environment (MPE) που προτάθηκε από το [45], (β) το level-based Foraging (LBF) που προτάθηκε από το [56]. Και τα δύο περιβάλλοντα περιλαμβάνουν συνεργατικές και ανταγωνιστικά προβλήματα, ενώ το δεύτερο παρέχει επίσης προβλήματα με αραιές ανταμοιβές. Στοχεύουμε να κάνουμε μια σύγκριση της επίδοσης του CAM σε προβλήματα από τα προαναφερθέντα settings και να συγκρίνουμε την επίδοσή του με γνωστούς αλγορίθμους του πεδίου.

Αρχικά, ξεκινάμε με τη συζήτηση του πειραματικού περιβάλλοντος που ακολουθούμε στα πειράματά μας. Χρησιμοποιούμε δύο δοκιμαστικά περιβάλλοντα που έχουν χρησιμοποιηθεί ευρέως για την αξιολόγηση της επίδοσης των αλγορίθμων και των πλαισίων εργασίας στη Πολυπρακτορική Ενισχυτική Μηχανική Μάθηση (MARL). Σε όλες τις εργασίες των δύο δοκιμαστικών περιβάλλοντων, αντί για οπτικές πληροφορίες, οι πράκτορες λαμβάνουν υψηλού επιπέδου χαρακτηριστικά ως παρατηρήσεις, τα οποία λειτουργούν ως τοπικές καταστάσεις. Στην ουσία, η πλήρης κατάσταση αποτελείται από την συνένωση των παρατηρήσεων των πρακτόρων, και κάθε πράκτορας έχει μερική παρατηρησιμότητα σχετικά με την πλήρη κατάσταση.

1.5 Αποτελέσματα

Σε αυτή την ενότητα συζητούμε τα αποτελέσματα του προτεινόμενου πλαισίου στις δοκιμές MPE και LBF MARL, στις οποίες καταφέραμε να επιτύχουμε καλή επίδοση σε σχέση με τις επιλεγμένες μεθόδους αναφοράς. Στην ακόλουθη πειραματική ανάλυση, επισημαίνουμε την επίδοση των αξιολογημένων αλγορίθμων MARL στα παρακάτω διαγράμματα. Σε όλα τα διαγράμματα, ο οριζόντιος άξονας δείχνει την πρόοδο του χρόνου (σε timesteps) της εκπαίδευσης των αλγορίθμων και ο κατακόρυφος άξονας μετρά το συνολικό ανταμοιβή επεισοδίου στις δοκιμές που λαμβάνεται κατά τη διάρκεια ενός σταθερού χρονικού διαστήματος εκπαίδευσης. Συγκεκριμένα, για τη μέτρηση της συνολικής ανταμοιβής επεισοδίου, κάθε 50000 timesteps εκτιμούμε τον αλγόριθμο MARL σε 10 παράλληλες τυχαίες περιβαλλοντικές περιπτώσεις (δηλαδή 10 περιβάλλοντα που δημιουργούνται από 10 τυχαίους σπόρους) και λαμβάνουμε το μέσον συνολικό ανταμοιβή επεισοδίου που προκύπτει από την εκτέλεση της πολιτικής σε αυτά τα περιβάλλοντα. Σε όλα τα πειράματα, ορίζουμε τον χρονικό ορίζοντα εκπαίδευσης (δηλαδή τον οριζόντιο άξονα κάθε διαγράμματος) ίσο με 10 εκατομμύρια timesteps. Επίσης, εμφανίζουμε το μέσον συνολικό ανταμοιβή επεισοδίου χωρίς καμία κανονικοποίηση, ώστε τα αποτελέσματα να μπορούν να αναπαραχθούν εύκολα.

1.5.1 Αποτελέσματα στο MPE

Ξεκινώντας από το πειραματικό σύνολο δοκιμών MPE, συγκρίνουμε τον CAM με τα ακόλουθα αναφορικά μοντέλα: COMA, MAA2C και ATM (τα οποία βασίζονται όλα σε ενημερώσεις πολιτικής κλίσης, όπως παρουσιάζονται στο κεφάλαιο 2). Χρησιμοποιούμε τέσσερις ρυθμίσεις περιβαλλοντικών μετρητών ως benchmark, και συγκεκριμένα το Spread (με το N να παίρνει τιμές $3, 4, 6$) και το SpeakerListener. Δεδομένου ότι όλα τα περιβάλλοντα δεν έχουν ρυθμίσεις αραιών ανταμοιβών, δεν χρησιμοποιούμε καμία εσωτερική κίνητρα στο CAM (δηλαδή χρησιμοποιούμε $\beta = 0$). Αυτό επιτρέπει μια δίκαιη σύγκριση με τα αναφορικά μοντέλα, τα οποία δεν χρησιμοποιούν ούτε σχήματα που βασίζονται στην εξερεύνηση αλλά βασίζονται στους στόχους της πολιτικής και στη συνολική αρχιτεκτονική τους. Ορίζουμε τον χρονικό ορίζοντα για τις εργασίες MPE σε 25 χρονικά βήματα, όπως ορίζεται στην επίσημη υλοποίηση.

Στο Σχήμα 1.2, εμφανίζουμε τη σύγκριση του CAM με τα αναφορικά μοντέλα στις ρυθμίσεις Spread. Παρατηρούμε ότι το CAM (σε κόκκινο) υπερτερεί σε όλα τα περιβάλλοντα Spread σε σχέση με τα άλλα μοντέλα. Το CAM καταφέρνει συνεχώς να επιτυγχάνει υψηλότερες συνολικές αναμονές επεισοδίων σε σημαντικά λιγότερα χρονικά βήματα. Συγκεκριμένα, εκτός από το Spread-3 όπου το MAA2C καταφέρνει επίσης να επιτύχει αντίστοιχο σκορ με το CAM, χρειάστηκε περισσότερα από 5 εκατομμύρια βήματα εκπαίδευσης για να το πετύχει, ενώ σε άλλες ρυθμίσεις Spread, το CAM υπερτερεί σημαντικά σε σχέση με τα αναφορικά μοντέλα. Σημειώνουμε ότι το ATM εμφανίζει καλύτερη επίδοση από το MAA2C, το οποίο αποτελεί τη βάση του (όπως και για το CAM) στο Spread-4 και Spread-5, όπου το MAA2C αποτυγχάνει εντελώς στην επίλυση της εργασίας. Όσον αφορά το COMA, αποτυγχάνει επίσης εντελώς να επιλύσει την εργασία σε όλα τα περιβάλλοντα Spread. Παρόμοια αποτελέσματα παρατηρούμε και στο SpeakerListener (βλ. Σχήμα 5.5); Το CAM χρειάζεται

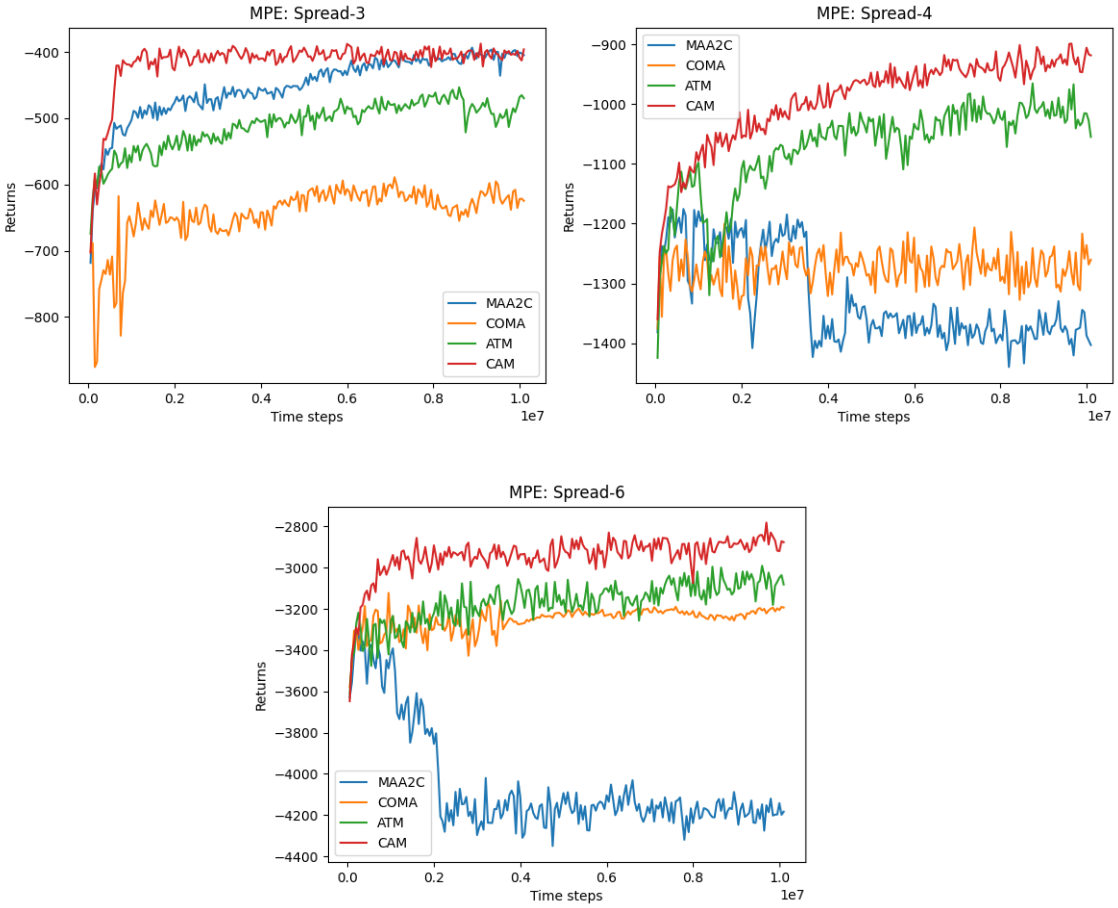

Figure 1.2: Αποτελέσματα στο Spread

περίπου 2 εκατομμύρια χρονικά βήματα για να επιλύσει την εργασία, ενώ το ATM και το MAA2C χρειάζονται περισσότερα από 4 εκατομμύρια χρονικά βήματα. Και πάλι, το COMA αποτυγχάνει να επιλύσει την εργασία MPE, αλλά εμφανίζει κάποια βελτίωση σε σχέση με το αρχικό του σημείο.

Επομένως, δεδομένου ότι το CAM είναι σε θέση να υπερτερεί σε αυτά τα περιβάλλοντα σε σχέση με όλες τις άλλες αξιολογημένες μέθοδοι πολιτικής κλίσης βασισμένες σε ομαδική ενίσχυση, ενδέχεται επίσης να μπορεί να μάθει αποτελεσματικές αναπαραστάσεις λόγω της μοντελοποίησης του παίκτη, που βοηθούν στη δραματική βελτίωση της επίδοσης της βάσης του (MAA2C). Αν και το CAM δεν ανακατασκευάζει πλήρως τις παρατηρήσεις των άλλων παικτών (λόγω της μερικής παρατηρησιμότητας του υποκείμενου περιβάλλοντος και της χρήσης των αυτο-εποπτευόμενων βαρών), κάθε παίκτης μοντελοποιεί τους άλλους με έναν σημαντικό τρόπο που ενισχύει τη συνολική επίδοση της ομάδας. Όπως τονίζουμε στη μελέτη μας για τις αποτομήσεις, αυτή η βελτίωση οφείλεται κυρίως στη χρήση των αυτο-εποπτευόμενων εκμάθησης βαρών, τα οποία στοχεύουν στην αναλογική επιλογή ποιων χαρακτηριστικών των άλλων παικτών είναι πιο σημαντικά για τη διευκόλυνση της διαδικασίας εκπαίδευσης κάθε παίκτη και για την ενίσχυση της επίδοσης της ομάδας.

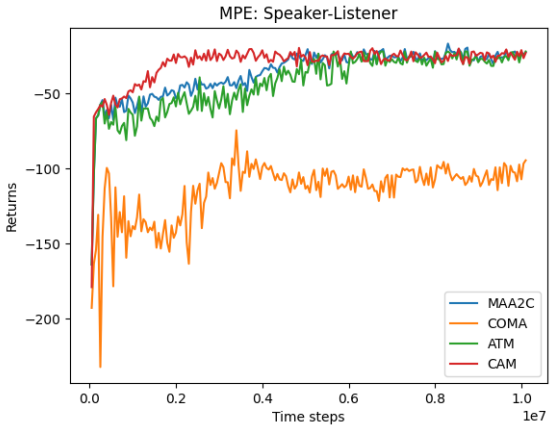

Figure 1.3: Results on MPE: SpeakerListener

1.5.2 Αποτελέσματα στο LBF

Συνεχίζουμε τη δοκιμή του LBF, όπου συγκρίνουμε τον αλγόριθμο CAM με τις προηγούμενες βάσεις (baselines) και επίσης με τον MASER, ένα πλαίσιο που βασίζεται στην εσωτερική εξερεύνηση (intrinsic motivation) και έχει καταγράψει πολύ καλές επιδόσεις σε διάφορα περιβάλλοντα με αραιές ανταμοιβές. Δοκιμάζουμε αυτούς τους αλγορίθμους σε τέσσερα περιβάλλοντα LBF, συγκεκριμένα: (α) $7 \times 7, 3p, 2f$, (β) $8 \times 8, 3p, 2f$, (γ) $11 \times 11, 4p, 3f$ και (δ) $12 \times 12, 2p, 2f$, τα οποία όλα αποτελούν περιβάλλοντα με αραιές ανταμοιβές. Σημειώνουμε ότι η επίλυση ενός περιβάλλοντος LBF αντιστοιχεί στην επίτευξη συνολικής επεισοδιακής ανταμοιβής ίσης με 1. Σημειώνουμε επίσης ότι χρησιμοποιούμε τώρα την πλήρη έκδοση του CAM, συμπεριλαμβανομένων των εσωτερικών ανταμοιβών. Ορίζουμε τον χρονικό ορίζοντα για τα MPE (Multi-Agent Partially Observable) καθήκοντα ίσο με 50 χρονικά βήματα, όπως καθορίζεται στην επίσημη υλοποίηση.

Στο Σχήμα 5.6, απεικονίζουμε την επίδοση των αξιολογηθέντων αλγορίθμων στα προαναφερθέντα περιβάλλοντα LBF. Παρατηρούμε ότι ο CAM είναι σε θέση να επιλύσει όλα τα καθήκοντα, επιτυγχάνοντας πάντοτε την βέλτιστη συνολική επεισοδιακή ανταμοιβή. Από την άλλη πλευρά, εκτός από το $8 \times 8, 3p, 2f$, οι ATM και MAA2C εμφανίζουν κάποια βελτίωση σε σχέση με τον COMA, ο οποίος αποτυγχάνει πλήρως να επιλύσει τα προβλήματα. Ο MAA2C λύνει τα $7 \times 7, 3p, 2f$ και $12 \times 12, 2p, 2f$ (όπως και ο ATM), αλλά χρειάστηκε πολύ περισσότερα χρονικά βήματα από τον CAM για να φτάσει σε αυτήν την επίδοση (ο CAM χρειάστηκε λιγότερα από το μισό χρονικό βήμα για να συγκλίνει σε μια καλή πολιτική). Αξιοσημείωτο είναι ότι ο CAM κατάφερε επίσης να επιλύσει το $8 \times 8, 3p, 2f$, το οποίο είναι το πιο δύσκολο περιβάλλον που δοκιμάσαμε, καθώς όλοι οι άλλοι αλγόριθμοι αποτύγχαναν πλήρως να αυξήσουν τη συνολική επεισοδιακή ανταμοιβή τους. Όσον αφορά τον MASER, παρόμοια με τον COMA, αποτύγχανε πλήρως να αυξήσει την συνολική επεισοδιακή ανταμοιβή του. Αυτό το αποδίδουμε στο γεγονός ότι ο MASER είναι ένας εκτός πολιτικής (off-policy) αλγόριθμος που βασίζεται στον QMIX και απαιτεί περισσότερη εκπαίδευση για την εύρεση καλών πολιτικών και πιθανώς σχεδιάστηκε κυρίως για την επίλυση περιβαλλόντων με αραιές

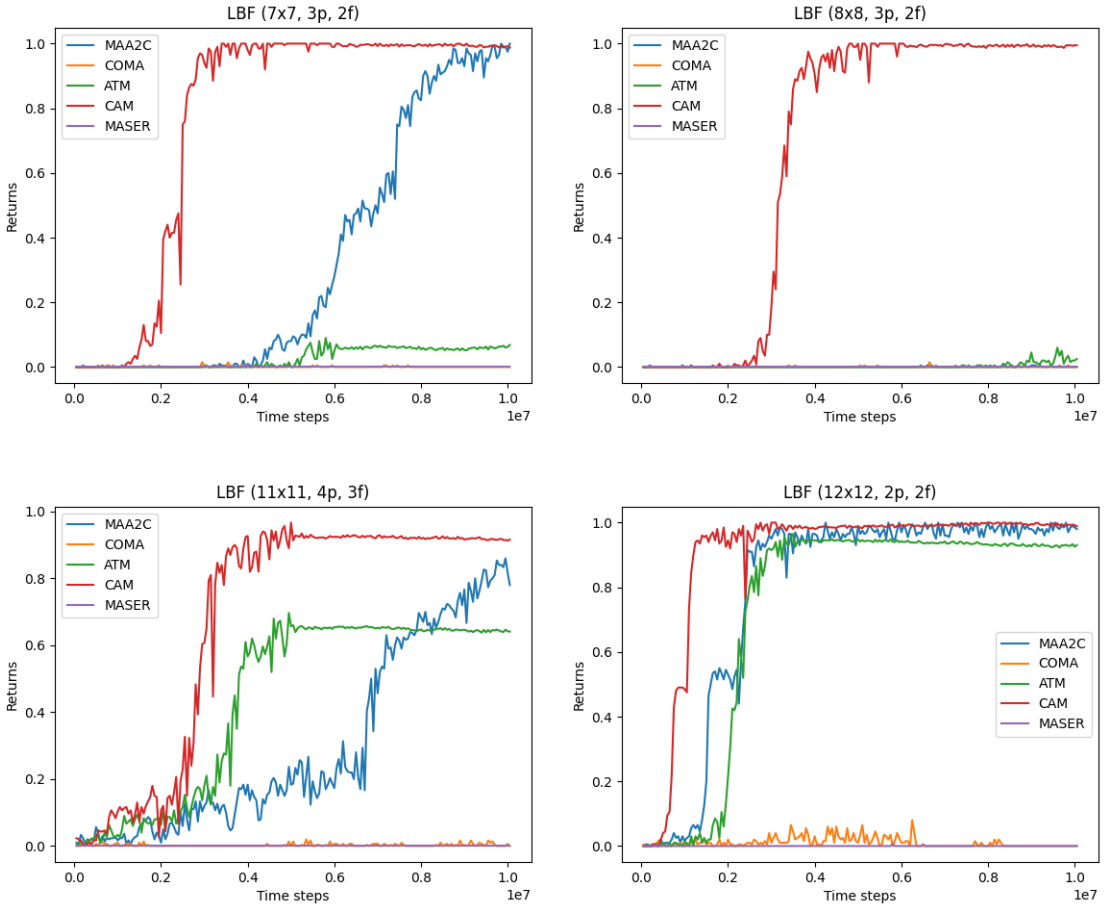

Figure 1.4: Results on LBF tasks

ανταμοιβές στο SMAC setting.

1.6 Συμπεράσματα και Συζήτηση

Αυτή η διατριβή ξεκίνησε μια σφαιρική εξερεύνηση διαφόρων αλγορίθμων ενισχυτικής μάθησης πολλαπλών πρακτόρων (Multi-Agent Reinforcement Learning - MARL) για την αντιμετώπιση συνεργατικών περιβαλλοντικών παιχνιδιών. Η έρευνα επικεντρώθηκε στην εξέταση προηγμένων αλγορίθμων MARL στο πλαίσιο της επίλυσης συνεργατικών ομαδικών παιχνιδιών, με έμφαση σε δύο βασικά στοιχεία της διαδικασίας μάθησης: (α) η ικανότητα των αλγορίθμων MARL να διευκολύνουν την μάθηση αναπαράστασης, επιτρέποντας τη δημιουργία εννοιών διανυσματικών ερμηνειών του υποκείμενου περιβάλλοντος. Αυτές οι ερμηνείες προάγουν τον αποτελεσματικό συντονισμό μεταξύ των πρακτόρων στις ενέργειές τους, και (β) η ικανότητα των πρακτόρων να αντιλαμβάνονται και να διατυπώνουν σημαντικούς στόχους πρακτόρων, συμβάλλοντας στη συλλογική προσπάθεια ενίσχυσης της συνολικής επίδοσης.

Από την μεριά μας, εισαγάγαμε το καινοτόμο πλαίσιο του Count-based Agent Modelling (CAM), το οποίο είναι εφαρμόσιμο σε οποιονδήποτε αλγόριθμο MARL που λειτουργεί υπό το

παραδεκτό παράδειγμα Κεντρικής Εκπαίδευσης και Αποκεντρωμένης Εκτέλεσης (Centralized-Training-Decentralized-Execution - CTDE). Το CAM συνδυάζει αρχές από τον τομέα της μοντελοποίησης πρακτόρων με την εσωτερική εξερεύνηση, επιδιώκοντας δύο βασικούς στόχους. Καταρχάς, το CAM χρησιμοποιεί νόημες, εκμαθήσιμες αναπαραστάσεις για να καθοδηγήσει τον συντονισμό των πρακτόρων και να ενισχύσει τις πολιτικές τους. Δεύτερον, αυτές οι αναπαραστάσεις χρησιμοποιούνται για νέα εσωτερική εξερεύνηση, που είναι ιδιαίτερα πολύτιμη για την αντιμετώπιση των προκλήσεων που παρουσιάζουν οι εργασίες με λίγες ανταμοιβές. Διεξήχθησαν εκτεταμένες πειραματικές αναλύσεις χρησιμοποιώντας δύο δημοφιλείς περιβάλλοντα δοκιμών MARL, και συγκεκριμένα το Πολυπρακτοριακό Περιβάλλον Σωματιδίων (Multi-Agent Particle Environment) και το Level-based Foraging. Τα αποτελέσματα δείχνουν ότι το CAM ξεπερνάει τους αλγορίθμους MARL όσον αφορά την συνολική ανταμειβή.

Τέλος, αυτή η διατριβή προσφέρει μια επιπλέον επιβεβαίωση της ικανότητας του αλγορίθμου "actor-critic" εντός του πλαισίου CTDE στην αντιμετώπιση στο MARL. Αυτό επιβεβαιώνει την καταλληλότητα και την αποτελεσματικότητα του αλγορίθμου σε συνεργατικά περιβάλλοντα πολλαπλών πρακτόρων.

Σαν συμπέρασμα, αυτή η διατριβή συνεισφέρει στην προαγωγή του πεδίου του MARL με την πρόταση του πλαισίου CAM, την ανάλυση των οφελών του μέσω αυστηρών πειραμάτων και τον ενημερωτικό φωτισμό σχετικά με την αποτελεσματικότητα του αλγορίθμου "actor-critic" σε συνεργατικά περιβάλλοντα. Μέσω αυτής της έρευνας, τονίζεται η σημασία της μάθησης νοήματος αναπαράστασης για την ενίσχυση του συντονισμού και της επίδοσης των πρακτόρων σε περιβάλλοντα πολλαπλών πρακτόρων.

Μελλοντικά, σκοπεύουμε να συγκρίνουμε το CAM με αλγορίθμους MARL που βασίζονται στην επικοινωνία, καθώς και με άλλους παρόμοιους αλγορίθμους MARL. Επιπλέον, σχεδιάζουμε να διεξάγουμε πειράματα σε άλλα περιβάλλοντα δοκιμών, όπως το SMAC και το Google Research Football. Τέλος, θα αξιολογήσουμε τα εξατομικευμένα βάρη που παρουσιάσαμε σε αυτήν τη διατριβή στο πλαίσιο των μεθόδων επικοινωνίας με σκοπό να καταστήσουμε την επικοινωνία των πρακτόρων πιο αποδοτική όσον αφορά την αύξηση του αριθμού των πρακτόρων.

Chapter 2

Introduction

2.1 General Discussion

In the contemporary landscape of technological advancements, the relationships between Artificial Intelligence (AI), Multi-Agent Systems (MAS), and Machine Learning (ML) have emerged as pivotal constituents in shaping the development and application of intelligent systems. This introductory chapter sets the stage for the exploration of these intricate interplays and investigates their collaborative potential in addressing complex challenges across various domains.

Artificial Intelligence represents the overarching concept that encompasses a spectrum of methodologies aimed at creating machines with human-like intelligence. Through the integration of techniques such as knowledge representation, natural language processing, computer vision, and robotics, AI seeks to enable machines to perform tasks that traditionally demand human cognitive abilities. Concurrently, Machine Learning has emerged as a subset of AI, focusing on the development of algorithms that allow systems to autonomously learn patterns and insights from data, subsequently enabling informed decision-making and predictions.

Multi-Agent Systems involve multiple autonomous agents collaborating or competing to achieve specific objectives. Agents can be software entities, robots, or even humans, and their interactions can encompass cooperative, competitive, or mixed scenarios. The complexity of real-world scenarios often necessitates the use of MAS to model and solve intricate problems that extend beyond the scope of single-agent approaches.

The integration of Machine Learning techniques within AI systems has been instrumental in creating adaptable and data-driven intelligent systems. ML enables AI models to autonomously learn from datasets, adapt to dynamic environments, and enhance their performance iteratively. Notably, AI applications such as self-driving cars [6, 12], traffic controllers [41, 39], medical [48, 5] and web [62, 38] systems, have leveraged ML to achieve unprecedented accuracy and reliability. The infusion of AI into Multi-Agent Systems introduces novel capabilities to individual agents, enabling them to learn, adapt, and optimize strategies. This amalgamation empowers agents within a MAS to dynamically respond

to changing environments, ultimately enhancing the collective performance of the system. For instance, AI techniques can empower agents in a supply chain optimization MAS to make informed decisions based on historical data and real-time information.

The interplay among AI, ML, and MAS is exemplified in applications where these domains converge to address intricate real-world challenges. Notable instances include AI-driven chatbots, such as in [1, 20], employing ML to comprehend and respond to user queries. These chatbots, functioning as agents in a multi-agent environment, can seamlessly collaborate with other agents within the organization to resolve complex issues, exemplifying the collaborative potential of these intertwined domains.

In the dynamic landscape of modern technology, the integration of Reinforcement Learning (RL) within Multi-Agent Systems (MAS) has garnered significant attention as a means to enhance the capabilities of autonomous entities. This introductory chapter sets the stage for delving into the realm of reinforcement learning within the context of MAS, highlighting its potential to revolutionize how agents collaborate and make decisions in complex, interconnected environments.

Reinforcement Learning, a subfield of Machine Learning, offers a framework for enabling agents to learn optimal actions through interaction with an environment. Unlike classical machine learning paradigms, RL systems learn from trial-and-error, where agents receive feedback in the form of rewards or penalties based on their actions. This learning process empowers agents to iteratively improve their decision-making strategies, adapt to uncertainties, and make informed choices in dynamic environments.

The integration of Reinforcement Learning techniques within Multi-Agent Systems introduces a new dimension to the collaborative dynamics of autonomous agents through an highly emergent research field called Multi-Agent Reinforcement Learning (MARL). By enabling agents to learn and adapt their behaviors based on both individual experiences and interactions with other agents, RL-equipped MAS can lead to emergent behaviors and sophisticated strategies that go beyond what traditional approaches can achieve. This adaptive capability holds promise for addressing challenges that involve intricate coordination, competition, and resource allocation among multiple agents.

2.2 Thesis Scope and Our Contributions

The aim of this thesis is to explore different MARL approaches and algorithms for solving cooperative benchmarking gaming environments. We are interested in studying the state-of-the-art MARL algorithms of the literature in solving cooperative team games, emphasizing more on the following two aspects of the learning process of such algorithms: (a) the representation learning power of MARL algorithms for reaching meaningful vector interpretations of the underlying environment which would encourage the agents to coordinate their actions in an effective manner, and (b) the agents' ability to perceive and design meaningful agent goals that would help the whole team strive for a better overall performance in the underlying task.

The main contributions of this thesis are the following:

- We propose **C**ount-based **A**gent **M**odelling (**CAM**), a novel MARL framework which can be built upon any MARL algorithm, under the Centralized-Training-Decentralized-Execution (CTDE) schema, and combines techniques from agent modelling with intrinsic exploration. CAM aims to (a) use meaningful learnable representations for guiding the agents' cooperation improving the agents' policies, and also (b) to utilize these representations for novel intrinsic exploration in solving difficult sparse rewards tasks.
- We carry out extensive experimental analysis of the proposed framework in two widely used MARL benchmarking testbeds, namely the MPE and LBF multi-agent gaming environments, and show that CAM outperforms state-of-the-art MARL algorithms in terms of total reward in these tasks.
- This thesis provides an additional proof-of-concept of the good performance of the actor-critic algorithm under the CTDE schema in solving MARL tasks.

2.3 General Introduction to Multi-Agent Systems

Imagine a scenario where a group of independent agents must collaborate in a shared environment to accomplish specific objectives [3]. These agents might share a common goal, like a fleet of mobile robots working together to efficiently move goods within a large warehouse, or they might have conflicting goals, such as agents engaged in a virtual market, each striving to maximize their individual gains. In this setting, rather than instructing the agents on how to interact, they are allowed to figure it out autonomously. They begin by taking actions in the environment and gathering experiences about how the environment changes as a consequence of their actions, as well as observing the behavior of other agents. Over time, they acquire skills needed for their tasks and learn how to coordinate their actions with other agents, even potentially developing a shared communication system. Eventually, the agents become highly proficient, achieving expertise in optimal interaction to achieve their goals.

The above description encapsulates the essence of multi-agent reinforcement learning (MARL). MARL builds upon reinforcement learning (RL), a concept where agents learn optimal decision-making policies by experimenting with different actions and receiving rewards. The ultimate objective is for agents to select actions that maximize their cumulative rewards over time. While single-agent RL focuses on determining the optimal policy for an individual agent, MARL is centered around discovering optimal policies for multiple agents and addressing the unique challenges that arise in this multi-agent learning problems.

The general framework of a multi-agent system contains multiple decision-making agents and an environment in which they interact to achieve a specified goal. A graphical representation of a multi-agent system can be seen in Fig. 2.1 while the basic components are analyzed bellow.

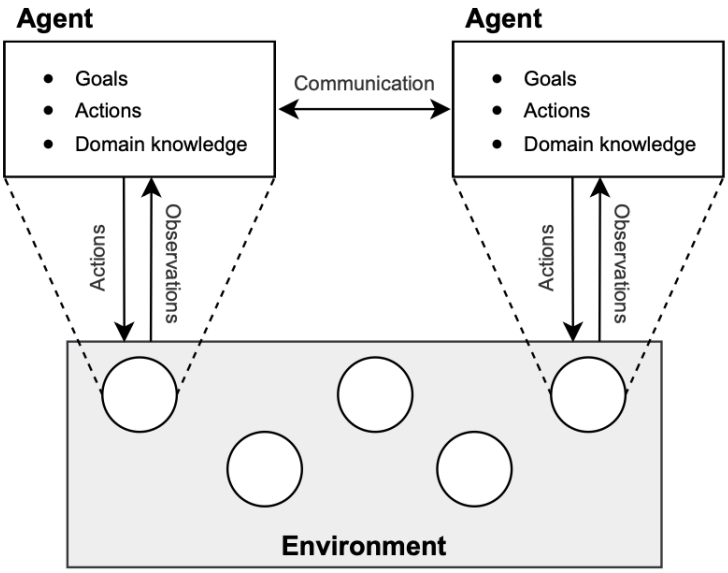

Figure 2.1: General framework of a multi-agent system [3]

Environment An environment(which might be physical or virtual), evolves over time and is influenced by the decisions made by the agents within it. The environment outlines the available actions agents can choose from and the observations each agent obtains regarding the environment's state. The environment's state can be described using discrete or continuous values, or a combination of both. For instance, in a 2D maze, the state might encompass agents' integer positions and their continuous orientations in radians. Actions in this environment can also be discrete or continuous, like navigating directions (up/down/left/right) in the maze and rotating by a continuous angle. Multi-agent environments typically stand out due to agents having a restricted and imperfect perspective of the surroundings. This implies that agents may only perceive partial information about the environment state, and the observations they receive could vary among agents.

Agents An agent is a entity that gains information about the state of the environment and can opt for various actions to impact that state. Agents may possess diverse prior knowledge about the environment, its potential states and how actions taken by agents influence those states. It's noteworthy that agents are driven by certain goals, meaning they have specific objectives and make action choices towards them. These goals could involve achieving a particular state within the environment or optimizing factors like financial gains. In the context of multi-agent reinforcement learning (MARL), these objectives are defined through reward functions, which outline numeric reward signals agents receive upon executing specific actions within specific states. The term "policy" refers to a function employed by the agent to pick actions (or assign probabilities to different actions) based on the present state of the environment. When an agent can only partially observe the

environment, the policy might be conditioned on the agent's current and past observations.

A unique characteristic of multi-agent systems is that agents should coordinate their actions by cooperate or compete with each other to achieve their objectives. In the cooperative setting agents' objectives are aligned and they collaborate to achieve a common goal. On the other hand in the competitive setting agents have adverse goals and thus they should compete with each other. A third setting might be the one between the two extremes, where agents might have aligned goals in some parts and different in other, requiring both cooperation and competition which can lead to highly complex multi-agent interactions among them.

Research in multi-agent systems can span across different directions as [3] how to design algorithms that enables agents to act optimally towards their goals; how to design environments that motivate agents to adopt specified long-term behaviors; how to communicate and propagate information among agents; and how norms, conventions and roles may emerge in a collective of agents. However, in this thesis we will focus solely in utilize RL algorithms to optimize agents' policies to maximize their accumulated rewards over time.

2.4 Intro to Multi-Agent Reinforcement Learning

Reinforcement Learning (RL) is a Machine Learning (ML) paradigm in which an artificial agent, or a group of artificial agents, aim to maximize the notion of a cumulative reward overtime by interacting with an environment, where they operate in. RL agents are not given explicit instructions on what actions to take; instead, they must discover the most effective behaviors through a process of trial-and-error learning [73]. Since rewards may be delayed, agents phase the dilemma of either exploiting the states that currently offer the highest known rewards or explore other states that have the potential to provide even higher rewards in the future. This trade-off between exploitation and exploration is crucial for the agent to learn and make optimal decisions in the long run [8]. Although RL algorithms can learn to make decisions without supervision or having complete models of the environment their performance deteriorate due to the curse of dimensionality (e.g. as the dimensions of the state-action space increase). Recently, this problem has been alleviated by combining RL algorithms with Deep Learning techniques, as neural networks can find find low-dimensional representations of high-dimensional data, enabling agents to master a wide variety of complex decision making task such as the board game Go [60], the card Game Poker [14], indoor robot navigation [79], trade execution [52] and cyber security [32].

Given the recent breakthroughs in the field of single agent RL researchers started investigating if those algorithms can also be applied in multi-agent systems. According to the single agent case the policies are learned via trial-and-error with the aim to maximize the agents' cumulative rewards. The MARL training loop is depicted in Fig.2.2

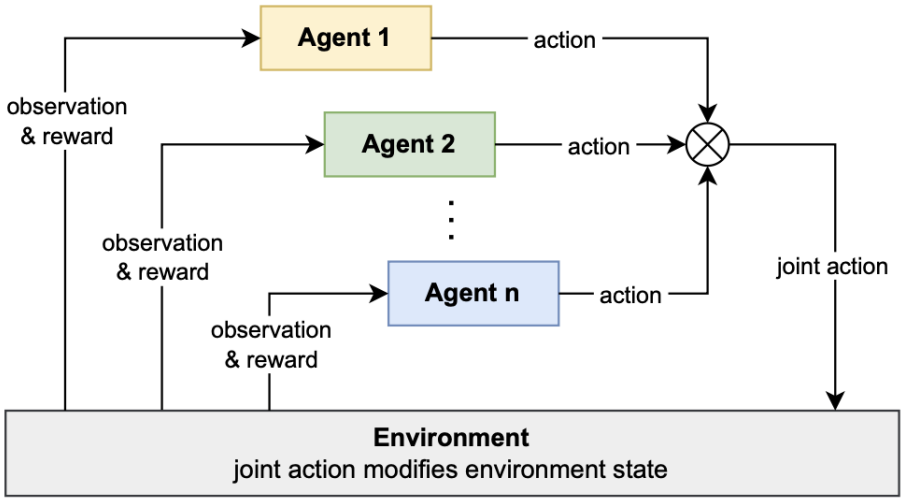

Figure 2.2: Training loop a multi-agent system [3]

More specifically, n agents chose their individual actions, which combination constitute the joint action. The joint action alters the state of the environment according to its dynamics, and the agents are receive their individual rewards based on this change as well as their individual observations about the new state [3]. This loop is repeated until a terminal condition is meet or indefinitely. A run which starts from the initial state and ends at the terminal state is called episode. The trajectories collected from multiple such episodes (e.g. observations, actions, rewards) are utilized to train the policies of the agents.

However although this process seams resonable, handling multiple agents is inherently more intricate due to (a) the fact that future rewards are contingent on the collective actions of several players and (b) the computational complexity rises as a consequence. The primary distinction between multi-agent and single systems lies in the fact that in the former, the dynamics of the environment are influenced not only by the uncertainty already present but also by the combined actions of all agents within that environment. Thus, when the environment becomes nonstationary, each agent encounters the challenge of dealing with a moving-target problem. This means that the optimal policy for an agent changes as the policies of other agents also change [15, 55]. Except for the stationarity assumption (the dynamics and reward do not change overtime) that does not hold in the multi-agent case, the curse of dimensionarity is even worse in those settings as the state-action space increases exponentially with the number of agents. However, multi-agent systems also provide the opportunity to agents to share knowledge, imitate or directly learn from other agents [17] making the learning procedure more efficient.

As multi-agent settings come with a number of challenges compared to the single agent ones, we analyze each of those in order to get a better sense of the problem formulation before continuing with the discussion of the main approaches. Those challenges can be categorized into computational complexity, nonstationarity, partial observability, and credit assignment. However, these challenges are not isolated but often co-occur, presenting a

combination of issues to tackle simultaneously. For instance, all multi-agent problems share the characteristic of high computational demands, which increase with the number of agents involved, making these problems computationally intensive. Also, the problem of nonstationarity arises when agents continuously adapt to the actions of the other agents, leading to infinity loops. This issue becomes more pronounced when agents have partial observability of the environment. With less information available, it becomes even more challenging for agents to distinguish the effects of their own actions from those of the other agents, complicating more the process of credit assignment. As a consequence, agents struggle to determine their individual contributions to the team's overall rewards.

2.4.1 Computational complexity

At present a major limitation of RL algorithms is their low sample efficiency, meaning that agents should interact an enormous amount of time with the environment in order to learn a useful policy [76]. For instance although humans can master the game of Pong in dozen of trials, an RL algorithm may require at least thousands of samples [18]. By studying computational complexity in RL we desire to examine how much computation is required (in terms of time and memory requirements) to collect sufficient data samples to output an approximation to the target [36]. The problem in RL is that the sample complexity worsens when multiple interacting agents are learning simultaneously, thus many studies focus on finding sample efficiency and scalability of algorithms to deal with the computational complexity in RL.

2.4.2 Nonstationarity

In multi-agent environments the Markov assumption is no longer valid because the state of the environment does not give sufficient information for optimal decision making, which is problematic given that most RL algorithms assume stationary environments to guarantee convergence. This nonstationarity can be naturally explained, since the new state of the environment is dependent on the joint action of all agents instead of the individual behaviors of the agents. Thus the agents may prefer to adapt to other agents' changing policies.

The nonstationarity problem can be addressed in different ways by focusing on the setting (e.g. cooperative [61], competitive [9]), the availability of opponents information [13] or whether the execution is centralized [22] or decentralized [65]. Other more advanced techniques may include learning as much as possible about the environment dynamics and mechanisms; for example, using centralised training with decentralised execution, through opponent modelling, and sharing information among the agents.

2.4.3 Partial observability

Under partial observability of the environment, the agents receive only local and incomplete observations and cannot access the full state of the environment, and as a result

the training process becomes a challenging procedure. In certain cases, the rewards and actions of other agents are not observable and the agents cannot attribute a change in the environment to their own actions. The most well-studied setting under partial observabilty is the Dec-POMDP setting, where a group of agents maximises a team reward using a joint policy. To deal with partially observability, approaches either adopt a centralised training and decentralised execution paradigm, such as in [46, 45], or use communication protocols to share information among the agents about the environment, such as in [21, 49].

2.4.4 Credit assignment

The most important credit assignment problem in MARL is that agents are not able to identify their contribution to the joint reward, because all agents act concurrently [50]. Thus, the problem of learning optimal policies becomes difficult, since agents cannot distinguish if changes in the global reward are due to their actions or to the other agents' actions. A solution can be to provide local rewards to each agent instead of a global reward. However, this approach, although it might increase the agents' rewards, it can encourage selfish behavior which may render challenging environments that require complex agent collaboration. A more appropriate method is proposed in [53] where the agents instead of receiving only the global reward, they also get an additional local reward to solve the credit assignment problem.

The credit assignment problem can also take the form of designing a reward function to promote effective collaborative behavior. However, such direction is difficult to follow when there exist mixed incentives within the same environment. Lastly, in MARL the "lazy" agent problem is also of great importance, because in the multi-agent setting with simultaneous interactions, when one agent learns a good policy, the others may avoid learning appropriate behaviors to not affect the performance of the former. [63].

2.5 Related Work

To address the above mentioned challenges different MARL algorithms have been developed and can be categized into the following groups: (1) centralised training and decentralised execution, (2) opponent modelling, (3) communication, (4) efficient coordination and (5) reward shaping. Fig. shows how those challenges relate to the different algorithms 2.3.

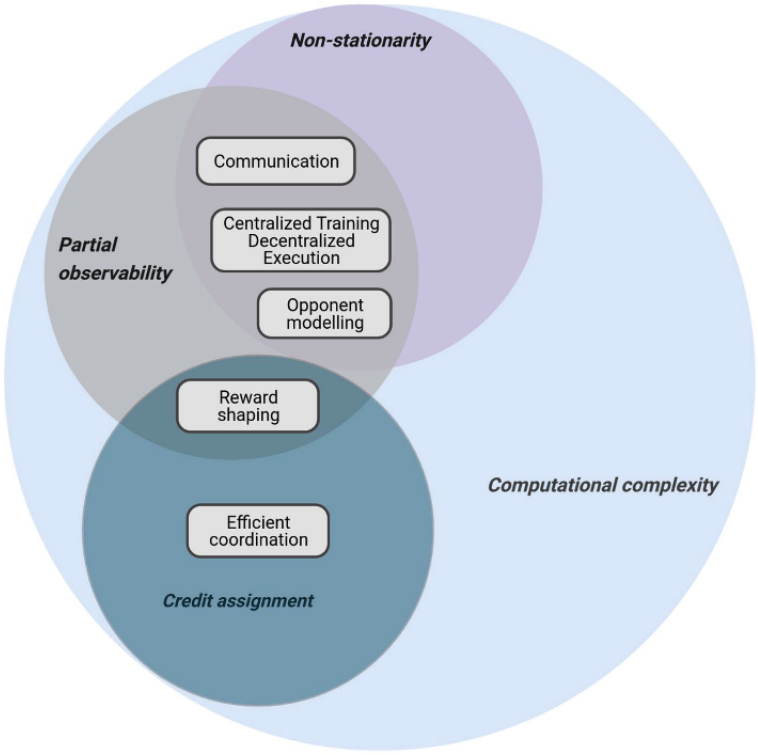

Figure 2.3: MARL challenges and solutions

As the our research focuses on the centralized training and decentralized execution as well as on the opponent modelling framework we will extensively analyze those sub fields.

2.5.1 Centralized Training and Decentralized Execution

Multi-agent systems can be trained under different schemes. The most common ones are depicted in the Fig. 2.4.

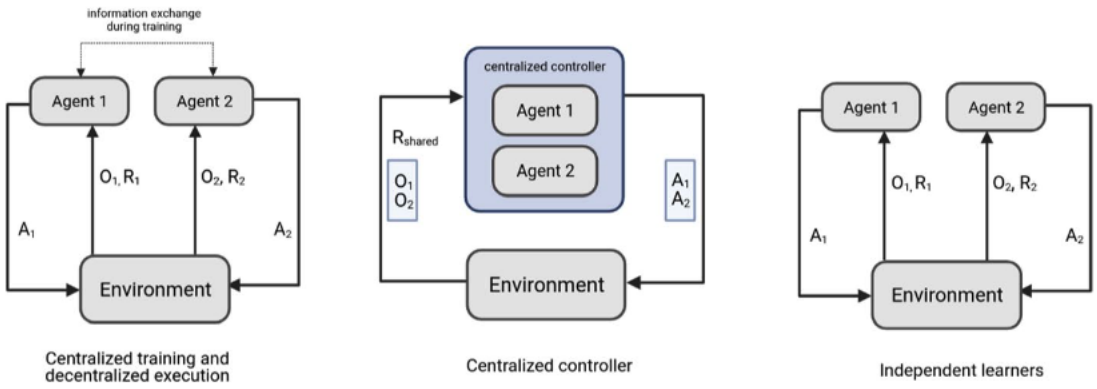

Figure 2.4: MARL training schemes

Starting from the simpler approach under the centralized scheme we can train multiple

agents utilizing a centralized controller reducing the problem to a single-agent one. In that case, all agents send their observations and policies to the centralized controller and the controller decides the action for each individual one. Nevertheless, this approach does not scale in large environments where the complexity increases exponentially. On the other hand, one can train each agent independently; however, this is not an efficient approach as each agent tries to find selfishly optimize its objective without aiming to collaborate in order for the system to solve non-trivial multi-agent environments.

One good approach is through the centralized training decentralized execution (CTDE) scheme [40]. In CTDE, during training the agents can access extra information, e.g. other agents' observations, rewards, gradients and parameters. Nevertheless, during execution each agent should execute its policy based only on its local observations. Adopting that framework mitigates nonstationarity and partial observability, as access to additional information during training stabilizes agents' learning [73]. The CTDE scheme can also be devided according to the adopted RL algorithm. If a value based method is used cooperating agents should optimize a joined value function, and studies investigate the best way to decompose and optimize this value function. Regarding the policy-gradient methods, as in the multi-agent setting, nonstationarity makes learning more challenging given that agents update their policies simultaneously, commonly the actor-critic architecture is utilized, in which a centralized critic is used to exchange extra information during training.

Value-based methods under the CTDE framework try to decompose centrally learned value functions utilizing them for decentralized execution. Value Decomposition Networks (VDN) [63] aim to decompose the centralized value function into a sum of linear individual value functions. Then the optimal policy is achieved by action greedily with respect to the Q-value, as Q-values estimate the expected reward from a particular state and action. An improved version of VDN is QMIX [58] which decomposes the joint value function into a nonlinear combination of individual value functions with an additional monotonic constraint to guarantee tractable optimization and consistency between the centralized and decentralized policies. However the monotonic constrain can be problematic in settings that require significant coordination [58]. Further extensions of QMIX constitute Weighted QMIX [57] which extends QMIX algorithm to nonmonotonic environments, adding a weighted scheme on joint actions with higher rewards and Attention QMIX [33] which enhances QMIX algorithm with an attention mechanism (for each agent a multi-head attention (MHA) layer is applied to summarize the information of the other agents) to deal with an indefinite number of agents. QTRAN [61] proposes another factorization method which escapes monotonicity and additivity constraints however it requires specific regularizations which can add further computational requirements to the already complex multi-agent system. The exploration problem that arise in QMIX is addressed in MAVEN [46], which utilizes a combined value and policy gradient method by conditioning value-based agents on the shared latent variable controlled by a hierarchical policy. The proposed solution is committed exploration e.g. by coordinated exploratory actions over extended time steps in dealing with environments that require long-term coordination.

Policy gradient methods are based on the actor-critic algorithm by utilizing a centralized critic to train the decentralized actors. Counterfactual multi-agent (COMA) [22] uses a centralized critic which has access to both the joint action taken by all agents and all the relevant information about the current state of the environment. On the other hand, each individual agent's policy only relies on the sequence of actions and observations it has experienced in the past. In addition it utilizes a counterfactual baseline to address the credit assignment problem. Accordingly, multi-agent Deep Deterministic Policy Gradient (MADDPG) [45] is an extension of the Actor-Critic algorithm, which enhances the training process by providing the critic with additional information and restricting the actor's access to only local information. Contrary to COMA, MADDPG utilizes a centralised critic for each agent to have different reward functions in competitive environments while it uses continuous instead of discrete policies. Other extensions of MADDPG are R-MADDPG [69] which is suited for partially observable environments utilizing recurrent actor and critics to keep a history of previous observations, as well as M3DDPG [43] which performs a minimax optimisation to learn robust policies against agents with changing strategies. As above mentioned methods concatenate all observations in the critic side which might be inefficient when dealing with large state spaces, Mean-Field Actor-Critic [75] aims to provide a solution by factorizing the Q-function on mean-field theory.

2.5.2 Opponent modelling

Opponent modelling aim to construct models of the beliefs, behaviours, and goals of other agents in the environment [4], which can used by the agents to augment their decision-making capabilities. Opponent modelling methods try to predict the action probabilities of the modelled opponent given a sequence of interaction as an input. Then agents can learn their policy utilizing the model of their opponent, enabling them to discover their intentions. Those algorithms are considered more data-efficient while they mitigate the nonstationarity and partially observability problem especially in adversarial settings, as agents collect historical observations to learn about the environment developing better policies. The first Opponent modelling approaches assumed that opponents have a fixed play and tried to find a Nash equilibrium in imperfect information games as Poker [29] by keeping a record opponents' historical behaviors in order to choose a best-response to their average strategies. Other approaches to the fixed play assumption are Alphazero [60] which is based on self-play and Monte Carlo Tree Search and has an excellent performance in the games of Go, chess, and Shogi as well as MuZero [59] which achieves the same performance by modelling only the value, policy and reward instead of the hole environment.

Later methods started modelling nonstationary environments and agents with changing policies. Switching Agent Model (SAM) [19] learns opponent models from observed state-action trajectories and utilizes a Bayes' rule variant to assign probabilities to opponent's actions starting from a prior belief that is updated during training. Deep Reinforcement Opponent Network (DRON) [28] uses two networks one to learn the Q-values and a second

one to learn the opponents policies in parallel with an expert network that models different opponent strategies. Deep Recurrent Policy Inference Q-Network (DRPIQN) [30] includes policy features as a hidden vector into the deep Q-network to adapt to unknown opponents, while its recurrent network enables learning in partially observable environments. In learning with Opponent-Learning Awareness (LOLA) [23] each agent shapes the anticipated learning of the other agents, including a term to measure the impact of an agent's policy on the learning behavior of opponents.

2.5.3 Representation Learning

The approach of using representative embeddings aiming to train agents more effectively derives from the single agent setting. [27] uses variational inference to learn different robot skills embeddings. During testing, the policy remain fixed and a new embedder is learned that interpolates between already learned skills. [16] learns an embedding utilizing a VAE by encoding state trajectories and decoding states and actions. By framing their problem as a hierarchical RL task the latent space aims to model low level skills which can be controlled by a higher level policy. [77] learn a latent representation using learned dynamics and reward modules and by conditioning the policy on the embedding show that transfering the encoder to unseen environments helps learning.

In the multi-agent setting and closer to our approach [24] introduces an encoder-decoder framework for modelling the agent's policy. In particular the encoder learn a representation of different agent trajectories, while the decoder reconstructs the modelled agent's policy. Accordingly [80] proposes a VAE for modelling agents in fully-observable settings. Local Information Agent Modelling (LIAM) [54] uses an encoder-decoder architecture to learn representations about the modeled agents conditioned only on the local observations of the controlled agent. This representation is then used as an extra feature during the policy learning. Our approach can be seen as a generalization of LIAM, which only has one controlled agent and the other agents are considered modeled, to the more challenging task of learning efficient representations when all agents participate in the training process and try to model the others. [81] learn to adapt to different tasks, by training a VAE conditioned on the observation, action, reward triplet of the controlled agent aiming to learn a latent representations of the observation and reward functions.

2.5.4 Exploration and Intrinsic Motivation

In reinforcement learning, intrinsic motivation refer to the internal or self-generated rewards that an agent receives during the learning process. Their goal is to encourage the agent to explore the environment and learn more efficiently without relying solely on extrinsic rewards aiming to mitigate the exploration-exploitation dilemma. Intrinsic rewards differ from extrinsic rewards as they are not originated from the environment but are designed based on curiosity-driven exploration, intrinsic motivation, and self-supervised learning techniques. The most intuitive strategy for computing the intrinsic rewards is

by counting observations, or observation-action pairs. In that case, and if we reward the agents inverse proportionally to the count of encountered observations frequent states will have a low intrinsic rewards (as the agent visits already seen states). Although this approach is efficient for small and discrete state-spaces, it becomes problematic when dealing with large and continuous state-spaces. Bellemare et al. [7] propose the use of pseudo-counts for observation-state pairs. They are produced based on a density model which predicts the recorded probabilities of the states. The authors in [66] compute hash-values of observations to achieve generalization. The density model in that case is hash tables of counts, while the agents are again rewarded inverse proportional to the counts. The hash algorithm is SimHash which measures the angular distance based on the sign of a randomized mapping making it computationally efficient and allowing similar inputs to be mapped to similar hash values.

Lately, there is a surge of approaches studying exploration in MARL for solving sparse reward tasks. MAVEN [47] employs a hierarchical control approach, wherein agent policies are contingent upon the latent variable generated by a shared hierarchical policy. Bohmer et al. [11] propose Independent centrally-assisted Q-learning (ICQL), an centralized agent that stabilized the impact of the exploration. This agent is utilized exclusively during the training phase and shares the replay buffer with decentralized IQL agents. The decentralized IQL agents are trained solely using extrinsic rewards. This framework allows the avoidance of unreliable influences from potentially misleading intrinsic rewards while still benefiting from the exploration incentives they provide. In the work of Wang et al. [71], a pair of exploration techniques, namely EITI and EDTI, are introduced to stimulate cooperative exploration by capturing the impact of one agent's actions on the behaviors of other agents. EITI evaluates the effects on state transition dynamics, while EDTI assesses both the influences on transition dynamics and rewards. Nevertheless, their scalability is limited due to the requirement of employing an approximation approach for gauging the impact of numerous agents on a single agent. This reliance on approximation can lead to potential failures in the methodology. SMMAE [78] pursues effective teamwork by dynamically balancing individual exploration and collaborative cooperation. In SMMAE, distinct exploration policies are developed for each agent, optimizing their personal state exploration while adapting their exploration likelihood according to the overall team policy's stability.

ROMA [70] aims at naturally materializing agent roles, prompting agents with analogous roles to collaborate in learning and to specialize in distinct sub-tasks. This is facilitated through the creation of a stochastic role embedding space, achieved via the introduction of innovative regularizers, and the alignment of individual policies with designated roles. Jiang et al. [35] propose Emergence of Individuality (EOI), which employs a probabilistic classifier to anticipate agent identities based on observations. EOI establishes intrinsic rewards for agents when their identities are accurately predicted by the classifier, motivating agents to explore their distinctive observations and thereby enhancing their recognizability through reinforcement learning. Li et al. [42] propose an innovative

information-theoretical regularization strategy aimed at amplifying the mutual information between agents' identities and their trajectories. This encourages diverse individualized behaviors and extensive exploration. Additionally, they enhance representation learning by integrating agent-specific modules into a shared neural network architecture, effectively fostering shared learning among agents through L1-norm regularization while maintaining essential diversity. MASER [34] undertakes the task of automatically generating suitable subgoals for multiple agents using information from the experience replay buffer. This process integrates individual Q-values and total Q-values and is complemented by the design of individual intrinsic rewards tailored to each agent. These rewards are grounded in actionable representations pertinent to Q-learning, fostering agents' attainment of subgoals while optimizing the collective action value.

2.6 Thesis Structure

The thesis is structured as follows: The introduction section (Chapter 1) provides an initial discourse, outlining the scope and contributions of the research (Section 1.2), and offering a broader perspective on Multi-Agent Systems (Section 1.3) and Multi-Agent Reinforcement Learning (Section 1.4). The latter subsection further delves into aspects like computational complexity, nonstationarity, partial observability, and credit assignment. A survey of related work (Section 1.5) encompasses topics such as centralized training, opponent modeling, representation learning, and exploration. The section culminates with an overview of the thesis structure (Section 1.6). The theoretical framework (Chapter 2) commences by elucidating the fundamentals of Markov Decision Processes (Section 2.1) and then traverses the realm of Reinforcement Learning, discussing policy gradient methods and actor-critic algorithms (Section 2.2). Multi-Agent Reinforcement Learning (Section 2.3) is dissected, touching on problem representations, Dec-POMDPs, and various MARL algorithms and frameworks, including Independent Policy Gradient, Multi-Agent Policy Gradient, MAA2C, COMA, MADDPG, VDN, QMIX, ATM and MASER. The chapter culminates with an exploration of Variational Autoencoders, spanning problem formulation, variational lower bounds, and the reparameterization trick (Section 2.4). The proposed framework, namely Count-based Agent Modelling (CAM), is described in Chapter 3 and unfurls with a presentation of the problem statement and motivation (Section 3.1), followed by the full presentation of the proposed methodology in Sections 3.2 and 3.3. The evaluation chapter (Chapter 4) embarks with an exposition of the experimental setup (Section 4.1), involving the Multi-Agent Particle Environment (MPE) and Level-based Foraging (LBF) testbeds. The subsequent sections showcase results, both on MPE and LBF, and delve into an ablation study, including the visualization of embeddings (Section 4.2 and 4.3). Finally, the discussion and future work chapter (Chapter 5) concludes the thesis by examining the implications of the findings and avenues for future research (Section 5).

Chapter 3

Theoretical Framework

3.1 Markov Decision Process

To begin discussing any RL algorithm we should first define the Markov Decision Process (MDP) [31]; an appropriate framework which models sequential decision-making problems. An MDP is a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, r, \gamma, \mu \rangle$ [2] where:

- $\mathcal{S}$ is a finite set of states
- $\mathcal{A}$ is a finite set of actions
- $\mathcal{P} : S \times A \to \Delta(S)$ is a transition function, where $\Delta(S)$ is the space of probability distributions over S and describes the dynamics of the environment. In that way $P(s' \mid s, a)$ denotes the probability of reaching the state s' given that the previous state and action was (s', a) accordingly.
- $r : S \times A \to [0, 1]$ is the reward function when an agent takes an action a at a state s.
- $\gamma \in [0, 1)$ is the discount factor and defines a horizon for the problem in continuous tasks without a terminal state, balancing short- and long-term rewards
- $\mu \in \Delta(S)$ is the initial state distribution, which determines the initial state s_0. The interaction of an agent with the environment in a MDP is illustrated in Figure 3.1.

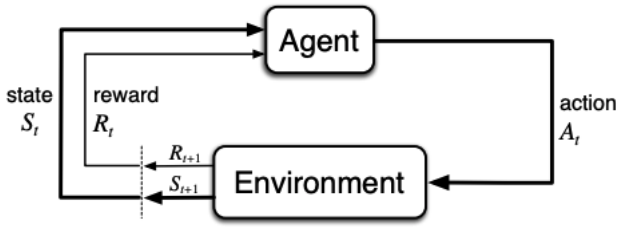

Figure 3.1: The agent–environment interaction in a Markov decision process[64]

In particular, the agent starts at an initial state $s_0 \sim \mu$ and at each time step t, selects an action $a_t \in \mathcal{A}$. After one time step, the agent receives the reward r_{t+1} for the selected action and observes a new state $s_{t+1} \sim P(\cdot \mid s_t, a_t)$ based on the dynamics of environment P. This consecutive interaction defines a trajectory $\tau = s_0, a_0, r_1, s_1, \ldots, s_t, a_t, r_t$. The agent in order to determine his decision-making strategy learns a stationary policy $\pi : \mathcal{S} \to \Delta(\mathcal{A})$, where $\Delta(\mathcal{A})$ denotes the space of probability distributions over $\mathcal{A}$, which allows him to choose actions based on the current state e.g. $a_t \sim \pi(\cdot \mid s_t)$. If the policy is deterministic and stationary it can be defined as $\pi : \mathcal{S} \to \mathcal{A}$. The agent learns his policy in order to maximize the expected discounted sum of future rewards G_t:

$$G_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \tag{3.1}$$

To solve an MDP we need to define the expected return from a particular state, which for a fixed policy and a starting state $s_0 = s$ can be defined as the state-value function $V_\pi : \mathcal{S} \to \mathbb{R}$:

$$\begin{aligned} V_\pi(s) &= \mathbb{E}_\pi[G_t \mid S_t = s] \\ &= \mathbb{E}_\pi[r_{t+1} + \gamma G_{t+1} \mid S_t = s] \\ &= \sum_a \pi(a \mid s) \sum_{s'} \sum_r p(s', r \mid s, a) \left[r + \gamma \mathbb{E}_\pi[G_{t+1} \mid S_{t+1} = s'] \right] \\ &= \sum_a \pi(a \mid s) \sum_{s'} \sum_r p(s', r \mid s, a) \left[r + \gamma V_\pi(s') \right] \end{aligned} \tag{3.2}$$

where the expectation is with respect to the randomness of the trajectory, and specifically the randomness of the state transition and the stochastisity of π. Also, by bounding the $r \in [0, 1]$, it holds that $0 \leq V_\pi(s) \leq \frac{1}{1-\gamma}$. Having defined the state value function, given a state s, the agent tries to find a policy π to maximize the value function; i.e $\max_\pi V(s)$.

We also define the expected return for a particular state-action pair as the action-value function $Q_\pi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$:

$$\begin{aligned} Q_\pi(s, a) &= \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a] \\ &= \sum_{s'} P(s, a, s')[r + \gamma V_\pi(s')] \\ &= \sum_{s'} P(s, a, s')[r + \gamma \sum_{a'} \pi(a'|s') Q_\pi(s', a')] \end{aligned} \tag{3.3}$$

where the expectation is again with respect to the randomness of the trajectory and bounded by $\frac{1}{1-\gamma}$.

3.2 Reinforcement Learning

3.2.1 Policy Gradient Methods

The most natural approach to find the optimal policy is to directly optimize the policy using a gradient-based algorithm. This family of approaches is often labelled as *policy*

gradient (PG) algorithms in the literature [64]. The distribution of a trajectory τ, given a class of parametric policies $\{\pi_\theta \mid \theta \in \Theta \subset \mathbb{R}^d\}$ and a starting distribution μ can be defined as:

$$Pr_\mu^\pi(\tau) = \mu(s_0)\pi(a_0 \mid s_0)P(s_1 \mid s_0, a_0)\pi(a_1 \mid s_1)\dots$$

The total discounted reward for the trajectory τ can be defined as $r(\tau) = \sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)$, where the (s_t, a_t) denote the trajectory's state-action pairs. In that sense, we denote the value function of the starting distribution under the policy π_θ by $J(\theta) = V^{\pi_\theta} = \mathbb{E}_{\tau \sim Pr_\mu^{\pi_\theta}}[r(\tau)]$. Recalling now that the agent's goal is to find a policy π to maximize the value function, we have to calculate the gradient of the previous defined value function, which is known as the policy gradient theorem. Before proceeding to the derivation of the equation, we define the state value gradient in terms of an expectation under the trajectory distribution as follows:

$$\begin{aligned}
\nabla V^{\pi_\theta}(\mu) &= \nabla \sum_\tau r(\tau) Pr_\mu^{\pi_\theta}(\tau) \\
&= \sum_\tau r(\tau) Pr_\mu^{\pi_\theta}(\tau) \nabla log Pr_\mu^{\pi_\theta}(\tau) \\
&= \sum_\tau r(\tau) Pr_\mu^{\pi_\theta}(\tau) \nabla log(\mu(s_0)\pi(a_0 \mid s_0)P(s_1 \mid s_0, a_0)\pi(a_1 \mid s_1)\dots) \\
&= \sum_\tau r(\tau) Pr_\mu^{\pi_\theta}(\tau) \left(\sum_{t=0}^{\infty} \nabla log \pi_\theta(a_t \mid s_t) \right) \\
&= \mathbb{E}_{\tau \sim Pr_\mu^{\pi_\theta}} [\nabla log \pi_\theta(\tau) r(\tau)]
\end{aligned} \tag{3.4}$$

Given the above derivation we introduce the REINFORCE algorithm. The pseudocode of REINFORCE is presented in Algorithm 3.

Algorithm 3 REINFORCE algorithm

While True (for each episode)

Generate an episode $s_0, a_0, r_1, \dots, s_{T-1}, a_{T-1}, r_T$ following $\pi(\cdot \mid \cdot; \theta)$

Loop for each step of the episode $t = 0, 1, \dots, T-1$:

$G \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} r_k$

$\theta \leftarrow \theta + \alpha \gamma^t G \nabla log \pi(a_t \mid s_t; \theta)$

REINFORCE constitutes the most primitive version of a PG algorithm; however it suffers from a number of issues that deteriorate its performance. The most crucial issue is that PG algorithms suffer from high variance, since policy gradients estimate the expected gradient of the policy's performance using samples collected from interactions with the environment. The Monte Carlo estimation of the gradient involves sampling multiple trajectories and computing their gradients. However, these trajectories are sampled based on the current policy, which means that they are not independent and also can come up with very different reward outcomes. This could lead gradient estimates to high variance. Another contributing factor to the high variance problem is the high-dimensional action spaces, in cases where the number of actions is large. In those cases, it becomes challenging

for the agent to explore the action space effectively, leading to high variance in the gradient estimates, as the agent may not have enough samples to accurately estimate the gradient for each action.

To mitigate this problem a number of solutions have been proposed in the literature. The simplest one is called Reward-to-Go and can be derived just by simple numerical operations.

$$\begin{aligned}\nabla V^{\pi_\theta}(\mu) &= \mathbb{E}_{\tau \sim Pr_\mu^{\pi_\theta}}\left[\left(\sum_t \nabla \log \pi_\theta(a_t \mid s_t)\right)\left(\sum_t r(s_t, a_t)\right)\right] \\ &= \frac{1}{N}\sum_{i=1}^{N}\left[\left(\sum_{t=1}^{T} \nabla \log \pi_\theta(a_{i,t} \mid s_{i,t})\right)\left(\sum_{t=1}^{T} r(s_{i,t}, a_{i,t})\right)\right]\end{aligned} \tag{3.5}$$

Writing the RL objective in the form of the equation 3.5, we can observe that by distributing the $\sum_{t=1}^{T} r(s_t, a_t)$ term in the $\sum_{t=1}^{T} \nabla \log \pi_\theta(a_{i,t} \mid s_{i,t})$ sum, we calculate the gradient of $\log \pi_\theta$ at a given time step weighted by the sum of rewards of all time steps. However, this should not be the case as it creates a causality problem given that policies at time t' cannot affect rewards at time t when $t < t'$. Therefore we can replace the sum $\sum_{t=1}^{T} r(s_{i,t}, a_{i,t})$ with the partial sum $\sum_{t'=t}^{T} r(s_{i,t'}, a_{i,t'})$ and denote $\hat{Q}_{i,t}$ by reward-to-go.

A more advanced technique to reduce the policy gradient variance is through the use of the *baselines*. To understand the intuition behind this idea, we can think of the policy gradient calculation as a weighted maximum likelihood procedure. In that sense, when we optimize our algorithm with different trajectory samples, we aim to increase the probability density of the trajectories that have a positive reward and reduce to probability density of trajectories with negative reward. However, it might be the case that also bad trajectories might have a positive but small reward. Therefore a more appropriate way to form our optimization goal is to increase the probability density of trajectories that are better than average and decrease the probability density of trajectories that are worse than average. We also note that the optimal parameters θ do not change if we add or subtract a constant value from the rewards. Given that the equation is very sensitive on the choice of the constant it will be beneficial to calculate its optimal value, so as to achieve minimum variance and also not introduce a bias term in the gradient estimation. Firstly, we show that the estimator is unbiased even despite the introduction of the baseline. In other words we need to show that:

$$\mathbb{E}_{\tau \sim Pr_\mu^{\pi_\theta}}[\nabla \log \pi_\theta [r(\tau) - b]] = \mathbb{E}_{\tau \sim Pr_\mu^{\pi_\theta}}[\nabla \log \pi_\theta r(\tau)]$$

Using the linearity of expectations in order to split the terms we have that:

$$\begin{aligned}\mathbb{E}_{\tau \sim Pr_\mu^{\pi_\theta}}[\nabla \log \pi_\theta b] &= \int \pi_\theta \nabla \log \pi_\theta b d\tau \\ &= \int \nabla \pi_\theta b d\tau \\ &= b\nabla \int \pi_\theta \tau \\ &= b\nabla 1 = 0\end{aligned}$$

Therefore we observe that subtracting a baseline is unbiased in expectation. We can now focus on finding the optimal value for the baseline. The most natural choice would be the average reward $b = \frac{1}{N}\sum_{i=1}^{N} r(\tau)$ over all the sampled trajectories as in that case only the probability density of better than average trajectories get increased. To derive the lowest variance baseline we can minimize the policy gradient variance using the variance definition $Var[x] = \mathbb{E}[x^2] - \mathbb{E}[x]^2$. Using the variance definition we have :

$$Var[\nabla V^{\pi_\theta}(\mu)] = \mathbb{E}_{\tau\sim Pr_\mu^{\pi_\theta}}[(\nabla \log \pi_\theta[r(\tau) - b])^2] - \mathbb{E}_{\tau\sim Pr_\mu^{\pi_\theta}}[\nabla \log \pi_\theta[r(\tau) - b]]^2$$

where the $\mathbb{E}_{\tau\sim Pr_\mu^{\pi_\theta}}[\nabla \log \pi_\theta[r(\tau) - b]]^2$ reduces to $\mathbb{E}_{\tau\sim Pr_\mu^{\pi_\theta}}[\nabla \log \pi_\theta r(\tau)]^2$ as we proved before, thus we can focus on only on the first term of the right hand side of the equation.

Calculating its derivative (and defining $g(\tau) = \nabla log \pi_\theta(\tau)$) we have that:

$$\begin{aligned}
\frac{dVar}{db} &= \frac{d}{db}\mathbb{E}_{\tau\sim Pr_\mu^{\pi_\theta}}[g(\tau)^2(r(\tau) - b)^2] \\
&= \frac{d}{db}\left(\mathbb{E}_{\tau\sim Pr_\mu^{\pi_\theta}}[g(\tau)^2 r(\tau)^2] - 2\mathbb{E}_{\tau\sim Pr_\mu^{\pi_\theta}}[g(\tau)^2 r(\tau) b] + b^2\mathbb{E}_{\tau\sim Pr_\mu^{\pi_\theta}}[g(\tau)^2]\right) \\
&= 0 - 2\mathbb{E}_{\tau\sim Pr_\mu^{\pi_\theta}}[g(\tau)^2 r(\tau)] + 2b\mathbb{E}_{\tau\sim Pr_\mu^{\pi_\theta}}[g(\tau)^2]
\end{aligned}$$

and by setting the derivative equal to zero and solving for b we get:

$$b = \frac{\mathbb{E}_{\tau\sim Pr_\mu^{\pi_\theta}}[g(\tau)^2 r(\tau)]}{\mathbb{E}_{\tau\sim Pr_\mu^{\pi_\theta}}[g(\tau)^2]}$$

So the optimal value for b is the expected reward re-weighted by the expected gradient magnitudes.

3.2.2 Actor-Critic Algorithms

Actor critic algorithms are based on the policy gradient framework but they are also augmented with learned value and action-value functions. To derive the actor-critic algorithm we will begin by examining the policy gradient equation utilizing the reward to go.

$$\begin{aligned}
\nabla V^{\pi_\theta}(\mu) &= \frac{1}{N}\sum_{i=1}^{N}\left[\left(\sum_{t=1}^{T}\nabla \log \pi_\theta(a_{i,t} \mid s_{i,t})\right)\left(\sum_{t'=t}^{T} r(s_{i,t'}, a_{i,t'})\right)\right] \\
&= \frac{1}{N}\sum_{i=1}^{N}\left[\left(\sum_{t=1}^{T}\nabla \log \pi_\theta(a_{i,t} \mid s_{i,t})\right)\hat{Q}_{i,t}\right]
\end{aligned}$$

This $\hat{Q}_{i,t}$ denotes an estimate of the expected reward if we take action $a_{i,t}$ in the state $s_{i,t}$ and subsequently follow the policy until the end of the trajectory. However, if we visit the same state action pair again we will end up with a different estimate as the policy and the MDP contain randomness. Therefore to get a better estimate of the reward to go we can calculate its full expectation e.g. the action-value function $Q(s_t, a_t) = \sum_{t'=t}^{T}\mathbb{E}_{\pi_\theta}[r(s_{t'}, a_{t'}) \mid s_t, a_t]$. The resulted modification leads to the calculation of the reward to go using many

samples and as a result has much lower variance than using just one as in the case of the reward to go approach. Following the logic of the policy gradients we can lower the variance even more by subtracting a baseline. A common choice for the baseline is the value function as in that case, the difference of the action value function and the value function represents an estimate of how much better the action $a_{i,t}$ is on average, than the average action that one can take at the state $s_{i,t}$. This particular choice of baseline is known as *advantage*, denoted by $A(a_{i,t}, s_{i,t})$.

$$\nabla V^{\pi_\theta}(\mu) = \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T}\nabla \log \pi_\theta(a_{i,t} \mid s_{i,t})A(s_{i,t}, a_{i,t}) \tag{3.6}$$

where $A(s_{i,t}, a_{i,t}) = Q(s_{i,t}, a_{i,t}) - V(s_{i,t})$. In equation 3.6 in order to calculate the advantage function we have to use a function approximator (such as a neural network) which introduces some bias to the algorithm. However, the equation 3.6 achieves a huge decrease in the variance of the estimation compared to the policy gradient with a baseline which uses one sample estimates of the reward to go thus has very high variance.

In order to estimate the advantage function we have to approximate both the value and the action value function. As this is an inefficient approach, we can perform some assumptions. More specifically we can rewrite the action value function as:

$$\begin{aligned}
Q(s_t, a_t) &= \sum_{t'=t}^{T}\mathbb{E}_{\pi_\theta}[r(s_{t'}, a_{t'}) \mid s_t, a_t] \\
&= r(s_t, a_t) + \sum_{t'=t+1}^{T}\mathbb{E}_{\pi_\theta}[r(s_{t'}, a_{t'}) \mid s_t, a_t] \\
&= r(s_t, a_t) + \mathbb{E}_{s_{t+1}\sim p(s_{t+1}|s_t,a_t)}[V(s_{t+1})] \\
&\approx r(s_t, a_t) + V(s_{t+1})
\end{aligned}$$

We can observe that by making the approximation $\mathbb{E}_{s_{t+1}\sim p(s_{t+1}|s_t,a_t)}[V(s_{t+1})] \approx V(s_{t+1})$ we can obtain the action value function just by adding the value of the next state to the current reward. To approximate the value function we make use of a neural network that takes as an input the current state and outputs the value of that state. The value function can be expressed as (and by utilizing a neural network with parameters ϕ for the value function):

$$\begin{aligned}
V(s_{i,t}) &= \sum_{t'=t}^{T}\mathbb{E}_{\pi_\theta}[r(s_{t'}, a_{t'}) \mid s_{i,t}] \\
&= r(s_{i,t}, a_{i,t}) + \sum_{t'=t+1}^{T}\mathbb{E}_{\pi_\theta}[r(s_{t'}, a_{t'}) \mid s_{i,t+1}] \\
&\approx r(s_{i,t}, a_{i,t}) + V(s_{i,t+1}) \\
&\approx r(s_{i,t}, a_{i,t}) + \hat{V}_\phi(s_{i,t+1})
\end{aligned}$$

Therefore, by using a neural network with parameters ϕ for the value function we can approximate the target value by directly using the previous fitted value function. We can

also train the neural network by minimizing the MSE loss between the predicted values and the target values as:

$$\mathcal{L}(\phi) = \frac{1}{2}\sum_i ||\hat{V}_\phi(s_i) - y_i||^2$$

where $y_i = r(s_{i,t}, a_{i,t}) + \hat{V}_\phi(s_{i,t+1})$. Given the above description we formalize the actor-critic algorithm in Algorithm 4.

Algorithm 4 Actor Critic algorithm

While True (for each episode)

sample $\{s_i, a_i\}$ from $\pi_\theta(a \mid s)$

fit $V_\phi(s)$ to sampled reward sums

evaluate $\hat{A}(s_i, a_i) = r(s_i, a_i) + \hat{V}_\phi(s_i') - \hat{V}_\phi(s_i)$

$\nabla J(\theta) \approx \sum_i \nabla \log \pi_\theta(a_i \mid s_i)\hat{A}(s_i, a_i) \qquad \theta \leftarrow \theta + \alpha \nabla J(\theta)$

3.3 Multi-Agent Reinforcement Learning

3.3.1 Multi-Agent problem representations

To model multi-agent systems we need to advance our MDP framework given the violation of the stationarity assumption as already discussed. A major difference in multi-agent settings is that the definition of the problem is related to the nature of the interaction between agents, namely cooperative, competitive or mixed, or even on whether agents take their actions sequentially or simultaneously. A visualization of the different approaches can be found in Figure 3.2.

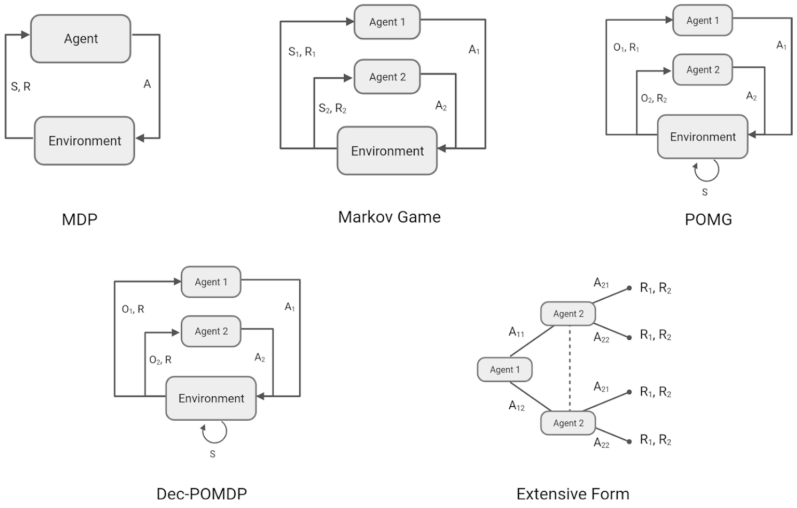

Figure 3.2: Problem representation MARL

In more detail if agents have full observability of the whole state the problem can be modelled as a Markov Game. If they also collaborate aiming to maximize a common

reward then the problem can be defined as a team Markov Game. Otherwise, if the agents have partial observability of the state as a partially observable Markov Game. On the other hand, if the agents have partial observability and cooperate but execute actions in a decentralized manner then the problem can be formulated as a decentralized POMDP (Dec-POMDP). Lastly, if agents take actions sequentially instead of simultaneously, the problem is represented as an extensive-form game.

3.3.2 Dec-POMDPs

We consider the fully cooperative multi-agent task as a **Dec-POMDP**, which is formally denoted by a tuple $G = \langle S, A, P, r, Z, O, N, \gamma \rangle$. S is the state space. At each timestep t, every agent $i \in \mathcal{A} \equiv \{1, 2, ..., N\}$ selects an action $a^i \in A$ which is a part of the joint action $\mathbf{a} \in \mathbf{A} \equiv A^N$. $P(s' \mid s, \mathbf{A}) : S \times \mathbf{A} \to [0, 1]$ is the state transition function. $r(s, \mathbf{a}) : S \times \mathbf{A} \to R$ is the reward function which is shared by all agents and $\gamma \in [0, 1)$ is the discount factor. We consider partial observability in our settings; each agent does not have access to the state yet samples observations $o \in O$ according to the observation function $Z(s, i) : S \times \mathcal{A} \to O$. We also consider that at each timestep the full state is the the union of the agents' observations (in accordance to the dec-MDP definition in [10]). The action-observation history for agent i is denoted by $\tau^i \in T \equiv (O \times A)^*$, on which the agent can condition its individual policy $\pi^i(a^i \mid \tau^i) : T \times A \to [0, 1]$. For brevity, we write a^{-1} to be the joint action of all the agents other than i and also use a similar convention for the policies π^{-1}. The joint policy π characterizes the action-value function: $Q^\pi(s_t, \mathbf{a}_t) = \mathbb{E}_{s_{t+1:\infty}, \mathbf{a}_{t+1:\infty}} \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k} \mid s_t, \mathbf{a}_t \right]$. The goal is to find the optimal action-value function Q^*.

Dec-POMDPs constitute a challenging framework in terms of computational complexity as the are not solvable utilizing polynomial-time algorithms while optimal solution search in the policy space is intractable [10].

3.3.3 Multi-Agent RL Algorithms

In order to solve multi-agent problems we should extend the single agent Reinforcement Learning algorithms to the multi-agent case. Our analysis will be based on [3]. For consistency we will use ϕ for the parameters of the learned value functions and θ for the policy functions parameters. As we will operate under the CTDE framework during training agents commonly utilize the joint observations but during execution they condition only on their local observations. We will present the MARL algorithms using the notation for partially observable environments, denoting with h the histories of observations. However, as in some cases the algorithms utilize the full state of the environment we will highlight where the local observation history of agent $h_i^t = (o_i^0, o_i^1, \ldots, o_i^t)$ of agent i, the joint observation history $h^t = (o^0, o^1, \ldots, o^t)$ or the state s^t at timestep t should be used. Also, in partially observable environments where the full state is not accessible, it can be approximated by the joint observation history $s^t \approx h^t$. On the other hand, in fully

observable environments agents can utilize the full state of the environment instead of the individual or joint observation histories. To condition deep value functions and policies on the history of observations we will utilize recurrent neural networks (RNN) as described in [26]. As RNNs are able to represent the full history of observation as a hidden state just by receiving one observation at a timestep we can also condition the value and policy functions only on the most recent observation.

3.3.4 Independent Policy Gradient Method

The easiest way to approach the multi-agent problem is to allow multiple agents to act and learn simultaneously in a shared environment. In that case agents perceive other agents as part of the environment but the are trained using single-agent Reinforcement Learning algorithms, thus the are considered independent. Beginning with the REINFORCE algorithm in multi-agent settings each agent has its own policy which is based on its own observations, actions and rewards. Thus, agents follow their policy gradient by computing the gradient of the expected return with respect to their own policy parameters.

$$\begin{aligned} \nabla J(\theta_i) &= \mathbb{E}_\pi \left[r_i^t \frac{\nabla \pi(a_i^t \mid h_i^t; \theta_i)}{\pi(a_i^t \mid h_i^t; \theta_i)} \right] \\ &= \mathbb{E}_\pi \left[r_i^t \nabla \log \pi(a_i^t \mid h_i^t; \theta_i) \right] \end{aligned} \tag{3.7}$$

We again update the parameters of the policy in the direction in which the probability of selecting an action increases ($\nabla \pi(a_i^t \mid h_i^t; \theta_i)$) proportional to the returns (R_i^t), while normalizing by the inverse of the probability of selecting an action under the current policy $\pi(a_i^t \mid h_i^t; \theta_i)$. The independent REINFORCE algorithm is provided in algorithm 5

Algorithm 5 Independent REINFORCE algorithm

Initialise n actor networks with random parameters $\theta_1, \ldots, \theta_n$
Collect environment observations $o_1^0, \ldots, o_n^0$
for time step $t = 0, 1, 2, \ldots$ **do**
  for agent $i = 1, \ldots, n$ **do**
    Sample actions a_i^t from $\pi(a_i^t \mid o_i^t; \theta_i)$
Execute actions and collect observations o_i^{t+1} and rewards r_i^t.
for agent $i = 1, \ldots, n$ **do**
  Update parameters θ_i by minimizing the loss of the equation 3.7

The described independent REINFORCE algorithm is an on-policy algorithm which in multi-agent systems has an advantage over off policy algorithm as agents learn from the most recent policies of the other agents. By on-policy we mean that gradients are computed based on the most recent experiences of the agents, generated by their most recent policies. As in multi-agent systems agents' policies constantly evolve learning from the latest policies of all other agents leads to more stable learning.

Following the same approach we can also generalize the A2C algorithm to the multi-agent setting and apply it to each individual agent. As A2C runs in parallel environments,

the environment trajectories and the multiple agents form high dimensional batches. To illustrate that the observations of K environments for a given time step t are represented as:

$$\begin{bmatrix} o_1^{t,1} & \dots & o_n^{t,1} \\ \vdots & \ddots & \vdots \\ o_1^{t,K} & \dots & o_n^{t,K} \end{bmatrix}$$

One should note that actions and rewards form also respective matrices. Thus, the calculation of the A2C loss requires the iteration and summation over all the individual losses. Focusing on the loss of a single agent from a specific environment k:

$$\mathcal{L}(\theta_i \mid k) = -\log \pi(a_i^{t,k} \mid h_i^{t,k}; \theta_i) \left(r_i^{t,k} + \gamma V(h_i^{t+1,k}; \phi_i) - V(h_i^{t,k}; \phi_i) \right) \tag{3.8}$$

while the final loss of the policy is:

$$\mathcal{L}(\theta) = \frac{1}{K} \sum_i \sum_k \mathcal{L}(\theta_i \mid k) \tag{3.9}$$

Accordingly to calculate the value loss we have:

$$\mathcal{L}(\phi_i \mid k) = \left(V(h_i^{t,k}; \theta_i) - y_i \right)^2 \tag{3.10}$$

where $y_i = r_i^{t,k} + \gamma V(h_i^{t+1,k}; \bar{\theta})$ The Independent A2C algorithm is shown in Algorithm 6

Algorithm 6 Independent A2C algorithm

Initialize n actor networks with random parameters $\theta_1, \dots, \theta_n$

Initialize n critic networks with random parameters $\phi_1, \dots, \phi_n$

Initialize K parallel environments

Build a batch of initial observations for each agent and environment: $\begin{bmatrix} o_1^{0,1} & \dots & o_n^{0,1} \\ \vdots & \ddots & \vdots \\ o_1^{0,K} & \dots & o_n^{0,K} \end{bmatrix}$

for time step $t = 0, 1, 2, \dots$ **do**

Sample actions $\begin{bmatrix} a_1^{t,1} & \dots & a_n^{t,1} \\ \vdots & \ddots & \vdots \\ a_1^{t,K} & \dots & a_n^{t,K} \end{bmatrix}$ from policy $\pi(a_i^t \mid o_i^t; \theta_i)$

Execute actions and observe $\begin{bmatrix} o_1^{t+1,1} & \dots & o_n^{t+1,1} \\ \vdots & \ddots & \vdots \\ o_1^{t+1,K} & \dots & o_n^{t+1,K} \end{bmatrix}$ and rewards $\begin{bmatrix} r_1^{t,1} & \dots & r_n^{t,1} \\ \vdots & \ddots & \vdots \\ r_1^{t,K} & \dots & r_n^{t,K} \end{bmatrix}$.

for agent $i = 1, \dots, n$ **do**

Update parameters θ_i by minimizing the loss of the equation 3.8 over the batch.

Update parameters ϕ_i by minimizing the loss of the equation 3.10 over the batch.

ia2cl2

3.3.5 Multi-Agent Policy Gradient Method

Although independent execution of the RL algorithms can solve some multi-agent problems it suffers from the non-stationarity problem as agents perceive other agents (which constantly change their behaviour) as part of the environment. Adapting the CTDE paradigm can mitigate this problem, allowing agents to share information during training in order to stabilize training, while executing their policies in a decentralized manner.

The basis of all Policy Gradient Method is the policy gradient theorem which states that the gradients of a policy can be calculated as:

$$\begin{aligned} \nabla_\theta J(\theta) &= \sum_{s \in S} \Pr(s \mid \pi) \sum_{a \in A} Q^\pi(s,a) \nabla_\theta \pi(a \mid s;\theta) \\ &= \mathbb{E}_\pi \left[Q^\pi(s,a) \nabla_\theta \log \pi(a \mid s;\theta) \right] \end{aligned} \tag{3.11}$$

The equation 3.11 can be extended to multi-agent settings by noting that the expected returns of any agent's policy in multi-agent RL are dependent on the polices of all agents. Thus, the multi-agent version of the policy gradient theorem for agent i with an expectation over the policies of all agents can be definied as:

$$\nabla_\theta J(\theta_i) = \mathbb{E}_{a_i \sim \pi_i, a_{-i} \sim \pi_{-i}} \left[Q_i^\pi(s, \langle a_i, a_{-i} \rangle) \nabla_{\theta_i} \log \pi_i(a_i \mid s;\theta_i) \right] \tag{3.12}$$

As in single-agent case we can apply multi-agent policy gradient theorem to derive different policy gradient update rules. For instance in independent learning policy gradient algorithms, the expected return estimate of agent i, $Q_i(s, \langle a_i, a_{-i} \rangle)$, is calculated utilizing a value function conditioned only of the history of observations and actions of agent i, e.g., $Q_i(h_i, a_i) \approx Q_i(s, \langle a_i, a_{-i} \rangle)$. In the next section we will adopt the CTDE paradigm and estimate the expected return conditioned on centralized information.

3.3.6 Multi-Agent Actor-Critic (MAA2C) Algorithm

To redefine the actor-critic algorithm under the CTDE paradigm, we should analyze both the actor and the critic networks. Starting with the actor in independent case was defined as $\pi(h_i^t; \theta_i)$. Therefore, the agent requires only its local observation history to choose its actions. This property will ensure that the algorithm can be executed in a decentralized manner and thus should not be altered or enriched with centralized information.

However, considering the critic network one can observe that it is utilized only during training and it is discarded during execution. In that case, a centralized version of the critic makes more sense as it provides further information to the training without affecting the execution. The critic under the CTDE paradigm can be defined as $V(s^t; \phi_i)$ conditioning of the full state of the environment in order to approximate the agents' i state value. The redefined algorithm has access to the full state during training which is of significant importance especially in partially observable environments, as otherwise the critic would lack valuable information for the estimation of state values. The centralized critic has

access to other agents observations and thus can adopt to their changing behavior. We should mention that the new value loss for the critic can be calculated as:

$$\mathcal{L}(\phi_i) = \big(V(s^t;\phi_i) - y_i\big)^2 \tag{3.13}$$

where $y_i = r_i^t + \gamma V(s^{t+1};\phi_i)$. The Centralized A2C algorithm is provided in Algorithm 7, while the architecture of the centrilized critic can be found in Figure 3.3.

Algorithm 7 MAA2C algorithm

Initialize n actor networks with random parameters $\theta_1, \dots, \theta_n$
Initialize n critic networks with random parameters $\phi_1, \dots, \phi_n$
Collect environment observations $o_1^0, \dots, o_n^0$
for time step $t = 0, 1, 2, \dots$ **do**
    for agent $i = 1, \dots, n$ **do**
        Sample actions a_i^t from $\pi(a_i^t \mid h_i^t; \theta_i)$
Execute actions and collect states s^t, observations o_i^{t+1} and rewards r_i^t.
for agent $i = 1, \dots, n$ **do**
    Update parameters θ_i by minimizing the loss of the equation 3.8
    Update parameters ϕ_i by minimizing the loss of the equation 3.13

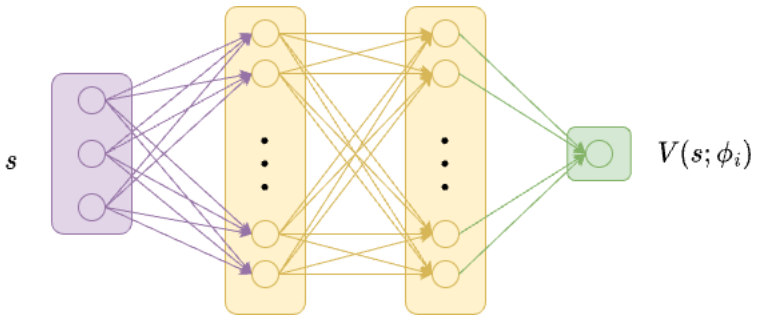

Figure 3.3: centralised state-value critic for agent i

In practice, the policy network is approximated by a Long short-term memory (LSTM) network followed by an MLP, while the critic network by a simple MLP.

3.3.7 COMA

To describe the COMA algorithm we should first recall the MAA2C algorithm which utilized a centralized state value function. Instead of the value function we can choose to learn the action-value function as a critic. In that way the centralized training of the critic Q_i of agent i is conditioned not only on the full state but also on the actions of all agents. The redefined loss is:

$$\mathcal{L}(\phi_i) = \big(Q(s^t, a^t;\phi_i) - y_i)\big)$$

where $y_i = r_i^t + \gamma Q(s^{t+1}, a^{t+1};\phi_i)$. We should note that in order to compute y_i for agent i we used the next state and the actions of all agents based on the current policy. Also, the

policy loss for agent i is:

$$\mathcal{L}(\phi_i) = -\log \pi(a_i^t \mid h_i^t, \theta_i) Q(s^t, a^t; \phi_i)$$

and the multi-agent policy gradient can be defined as:

$$\nabla_{\theta_i} J(\theta_i) = \mathbb{E}_\pi[\nabla_{\theta_i} \log \pi(a_i^t \mid h_i^t, \theta_i) Q(s^t, a^t; \phi_i)] \tag{3.14}$$

In general in the single agent setting the value based methods that optimize the action-value function, such as DQN [51], are off-policy i.e. they do not use the current policy in order to find the target y_i and instead they use a max operator over the next actions from batches or experiences sampled from a replay buffer. However this technique cannot be utilized in the actor critic algorithm, because the expected returns in the policy gradient theorem are calculated under the current policy of all agents. Thus, a replay buffer with experiences created under different policies cannot be utilized for this purpose and the method remains on-policy.

In the current version of our actor-critic we utilize the action value function to directly quantify the impact of the selected action on the expected returns. As in the single agent case, replacing the action value function with the advantage estimator can stabilize the training process since we need to calculate how much better a certain action is than the average action. However in the multi-agent case action-value functions can be utilized in order to solve the credit assignment problem based on the concept of difference rewards [72]. A suggested schema is through the difference rewards which aim to quantify the difference between the received reward and the reward agent i would have received if they had chosen a different action $\tilde{a}_i$ [3]:

$$d_i = r(s, \langle a_i, a_{-i} \rangle) - r(s, \langle \tilde{a}_i, a_{-i} \rangle) \tag{3.15}$$

Action $\tilde{a}_i$ is known as the default action. A difference reward tries to calculate the reward that agent i would have received if she instead had selected the default action. This question might be helpful in settings with common rewards as they quantify the contribution of each agent to the received reward. Although difference rewards seam beneficial to the training process, calculating them in practice is difficult as the selection of the default action for each player is unclear and computing $r(s, \langle \tilde{a}_i, a_{-i} \rangle)$ requires access to the full reward function.

Counterfactual multi-agent policy gradient (COMA) [22] tries to solve this issue by approximating the difference rewards utilizing a centralized action-value critic. The authors introduce a counterfactual baseline which marginalizes out the action of agent i to estimate the advantage of agent i for selecting the action a_i over following its current policy π_i [3]:

$$Adv_i(s, h_i, a) = Q(s, a) - \sum_{a_i' \in A_i} \pi_i(a_i' \mid h_i) Q(s, \langle a_i', a_{-i} \rangle) \tag{3.16}$$

A practical detail that it is worth noting regards the implementation of the centralized action value critic. A naive approach would be to use a neural network that has the state as input and outputs one action-value for each combination of action. However, in this

approach the number of outputs grows exponentially with the number of agents. A more efficient way to perform the same computation of the action-value function of agent i is to use both the state and the actions of the other agents a_{-i} as input the state and to output an action-value for each the agent's i actions. This modeling approach is illustrated in Figure 3.4.

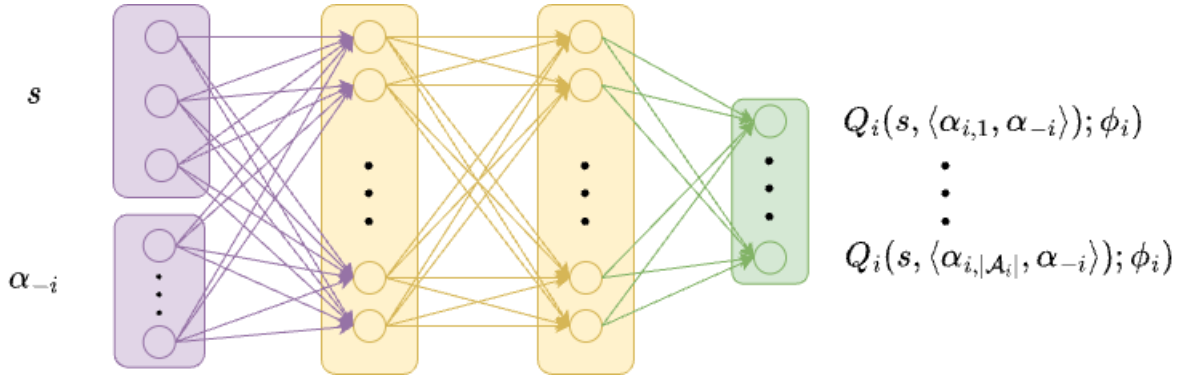

Figure 3.4: centralised action-value critic for agent i

3.3.8 MADDPG

Multi-agent deep deterministic policy gradient (MADDPG) [45] constitutes the multi-agent extension of the deep deterministic policy gradient (DDPG) [44]. Although we have not analyzed the DDPG algorithm, its main contribution is the introduction of a deterministic policy that also handles continuous actions. The interesting reader can refer to the corresponding paper for its details. MADDPG's main architecture is depicted in Figure 3.5.

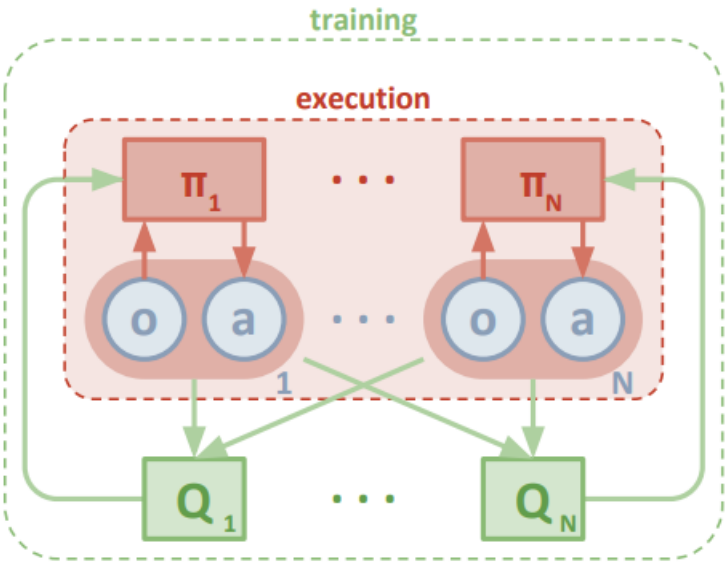

Figure 3.5: MADDPG architecture [45]

MADDPG can be also adapted to the CTDE schema, in order to allow the critic to use extra information and to be conditioned on both the state and actions of all agents during training.

The gradient of the expected return ($J(\theta_i) = \mathbb{E}[R_i]$) for agent i, with policy π_i and parameters θ_i is as follows:

$$\nabla J(\theta_i) = \mathbb{E}_{s\sim p^{\mu}, a_i\sim\pi_i}[\nabla \log \pi_i(a_i \mid h_i; \theta_i) Q_i^{\pi}(s, a_1, \ldots, a_N)] \tag{3.17}$$

where we utilize a centralized action-value function that receives as input the whole state $s = (o_1, \ldots, o_N)$, as well as the actions of all agents $(a_1, \ldots, a_N)$ while it outputs the Q-value for agent i. However, we note that the agent's i policy receives as input only the observations of agent i, such that agent t to select action a_i.

Extending this idea by introducing continuous deterministic policies μ_{θ_i} or μ_i in short, with parameters θ_i for agent i, the gradient takes the following form:

$$\nabla_{\theta_i} J(\mu_i) = \mathbb{E}_{s,a\sim D}[\nabla_{\theta_i} \mu_i(a_i \mid h_i; \theta_i) Q_i^{\mu}(s, a_1, \ldots, a_N) \mid_{a_i=\mu_i(h_i)}] \tag{3.18}$$

However here, since the policies are deterministic, we introduce a replay buffer D which contains tuples in the form $(s, s', a_1, \ldots, a_N, r_1, \ldots, r_N)$ with the experiences of all agents. The loss function of the centralized critic is defined as:

$$\mathcal{L}(\theta_i) = \mathbb{E}_{s,a,r,s'\sim D}[(Q_i^{\mu}(s, a_1, \ldots, a_N) - y)^2] \tag{3.19}$$

where $y = r_i + \gamma Q_i^{\mu'}(s', a'_1, \ldots, a'_N) \mid_{a'_j=\mu'_j(h_j)}$, with μ' denoting the set of target policies with delayed parameters θ'_i.

An advantage of the MADDPG algorithm is that by knowing the actions of the other agents the environment becomes stationary even as the policy changes, because $P(s' \mid s, a_1, \ldots, a_N, \pi_1, \ldots, \pi_N) = P(s' \mid s, a_1, \ldots, a_N) = P(s' \mid s, a_1, \ldots, a_N, \pi'_1, \ldots, \pi'_N)$ for any $\pi_i \neq \pi'_i$, which does not hold if we do not condition on the actions of other agents, like in most traditional RL algorithms.

3.3.9 VDN and QMIX

Although until now we have utilized centralized action-value functions to train multi-agent policy gradient methods, their direct learning (with Q-learning based algorithms) is difficult due to the exponential growth of the joint action space with the number of agents. Thus, one direction in MARL focuses on how to factorize action-value functions in order to be effectively learned. Before discussing approaches that can effectively factorize the action-value function we should remind that its centralized version can be written as:

$$Q(s^t, a^t; \phi) = \mathbb{E}\left[\sum_{\tau=t}^{\infty} \gamma^{\tau-t} r_s^{\tau} \mid s^t, a^t\right] \tag{3.20}$$

where r_s^{τ} represents the common rewards for all agents at time step τ.

The easiest way to solve the decomposition problem is to assume that the common reward can be linearly decomposed into individual utilities for each agent:

$$r_s^t = \bar{r}_1^t + \cdots + \bar{r}_n^t \tag{3.21}$$

where we define each individual utility of agent i at time step t as $\bar{r}_i^t$, while the bar denotes that the utility is obtained by the decomposition and not by the environment itself. Then the action-value function can be decomposed as:

$$\begin{aligned} Q(s^t, a^t; \phi) &= \mathbb{E}_\pi \left[\sum_{\tau=t}^{\infty} \gamma^{\tau-t} r_s^\tau \mid s^t, a^t \right] \\ &= \mathbb{E}_\pi \left[\sum_{\tau=t}^{\infty} \gamma^{\tau-t} \left(\sum_{i \in I} r_i^\tau \right) \mid s^t, a^t \right] \\ &= \sum_{i \in I} \mathbb{E}_\pi \left[\sum_{\tau=t}^{\infty} \gamma^{\tau-t} r_i^\tau \mid s^t, a^t \right] \\ &= \sum_{i \in I} Q(h_i^t, a_i^t; \phi_i) \end{aligned} \tag{3.22}$$

The method was introduced as Value Decomposition Networks (VDN) [63] and provides a computationally tractable method to optimize the individual utilities while allows for learning individual policies. The algorithm runs off-policy learning, utilizing a replay buffer D which contains the experiences of all agents and aims to optimize the loss of the equation 3.23, which approximates the centralized value function of all agents.

$$\mathcal{L}(\phi) = \frac{1}{B} \sum_{(h^t, a^t, r_s^t, h^{t+1}) \in \mathcal{B}} \left(r_s^t + \gamma \max_a Q(h^{t+1}, a; \bar{\phi}) - Q(h^t, a^t; \phi) \right)^2 \tag{3.23}$$

where

$$Q(h^t, a^t; \phi) = \sum_{i \in I} Q(h_i^t, a_i^t; \phi_i)$$

and

$$\max_a Q(h^{t+1}, a; \bar{\phi}) = \sum_{i \in I} \max_{a_i} Q(h_i^{t+1}, a_i; \bar{\phi}_i)$$

The architecture of VDN is illustrated in Figure 3.6

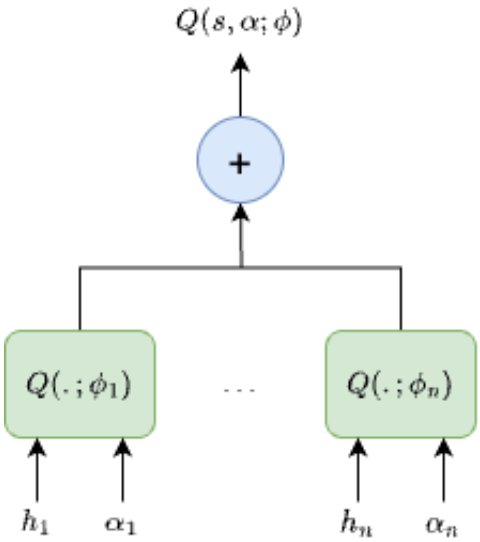

Figure 3.6: VDN architecture

Although VDN provides a solution to the decomposition problem, often the linearity assumption does not hold and thus a non-linear approach might be more suitable. The most well-known method to decompose the action-value functions non-linearly is QMIX [58]. To introduce QMIX we consider the problem of decomposition as learning individual action-value functions $Q(h_i, a_i; \phi_i)$ for each agent i conditioning only on the observation history and action of each agent. However those functions are optimized to approximate the centralized action-value function and not the individual expected return of each agent, while the must also satisfy the so called individual-global-max (IGM) property with respect to the centralized action-value function :

$$\underset{a=(a_1,\ldots,a_n)}{\arg\max}\ Q(s^t, a; \phi) = \begin{pmatrix} \arg\max_{a_1} Q(h_1^t, 1; \phi_1) \\ \vdots \\ \arg\max_{a_n} Q(h_n^t, n; \phi_n) \end{pmatrix} \tag{3.24}$$

The introduction of IGM property ensures that by greedily following each agent its policy with respect to its action-value function, all agents select greedily the joint action with respect to the decomposed centralized action-value function. This also means that the individual action value functions factorize the centralized action-value function [61]. The QMIX algorithm satisfy the IGM property by requiring the monotonicity of the centralized action-value function with respect to the individual ones, e.g., the derivative of the centralized action-value function with respect to agent action-value functions should be non-negative:

$$\forall i \in I, \forall a \in A : \frac{\partial Q(s, a; \phi)}{\partial Q(h_i, a_i; \phi_i)} \geq 0 \tag{3.25}$$

which means that if an agent i increases its value for a particular observation h_i and action a_i the centralized value for any joint actions $a' = \langle a'_{-i}, a_i \rangle$ in which agent i applies action a_i should never be decreased.

Similar to VDN, QMIX uses an individual action-value network (deep Q-network) for each agent. For the monotonic decomposition, QMIX also defines a mixing feedforward neural network f_{mix}, which combines the individual functions to approximate the centralized action-value function:

$$Q(s^t, a^t, \phi) = f_{mix}\left(Q(h_1^t, a_1^t, \phi_1), \ldots, Q(h_n^t, a_n^t, \phi_n); \phi_{mix}\right) \tag{3.26}$$

The mixing network ensures that the constraint of equation 3.25 will be satisfied if the weight matrices ϕ_{mix} take only positive values, a property that it is not required for the bias vectors in ϕ_{mix}. To guarantee that the weights used in QMIX are always positive, QMIX employs a distinct network referred to as the hypernetwork f_{hyper}. This hypernetwork is defined by a set of parameters ϕ_{hyper} and takes the entire state s as input. Its primary purpose is to generate the parameters ϕ_{mix} necessary for the mixing network. Additionally, to maintain a monotonic relationship, the hypernetwork applies an absolute value function to the outputs that pertain to the weight matrix of the mixing network f_{mix}. This ensures that the weights remain non-decreasing. When the optimization requires the computation of $Q(s, a; \phi)$, the separate utility values $Q(h_1, a_1; \phi_1), \ldots, Q(h_n, a_n; \phi_n)$ are calculated. These individual utilities are determined, and the mixing network parameters ϕ_{mix} are derived by inputting the state into the hypernetwork. These utilities are then combined into $Q(s, a; \phi)$ using the mixing network, which employs the parameters obtained from the hypernetwork. During the optimization process, all the parameters ϕ of the decentralized action-value function are collectively optimized. This includes the parameters of the individual utility networks $\phi_1, \ldots, \phi_n$, as well as the parameters of the hypernetwork ϕ_{hyper}, achieved by minimizing the value loss:

$$\mathcal{L}(\phi) = \frac{1}{B} \sum_{(s^t, a^t, r_s^t, s^{t+1}) \in \mathcal{B}} \left(r_s^t + \gamma \max_a Q(s^{t+1}, a; \bar{\phi}) - Q(s^t, a^t; \phi)\right)^2 \tag{3.27}$$

where batch $\mathcal{B}$ is sampled from the replay buffer, while the centrilized action-value function and its target value are given by equation 3.26 with parameters ϕ and $\bar{\phi}$ respectively. The optimization process for the parameters of the mixing network doesn't involve gradient-based techniques. Instead, these parameters are always generated as outputs from the optimized hypernetwork. QMIX's training procedure follows the same structure outlined as the VDN algorithm. However, QMIX adds the initialization and optimization of both the mixing network and the hypernetwork, while the value loss to be minimized is specified in equation 3.27.

It is also important to mention that in QMIX, the replay buffer stores the complete state s^t, which includes the individual observation histories $(h_1^t, \ldots, h_n^t)$ of each agent, extending the approach used in VDN. This complete state is required in QMIX for calculating the monotonic mixing behavior, as the hypernetwork function f_{hyper} takes the full environment state as a conditioning factor. The QMIX architecture is depicted in Figure 3.7.

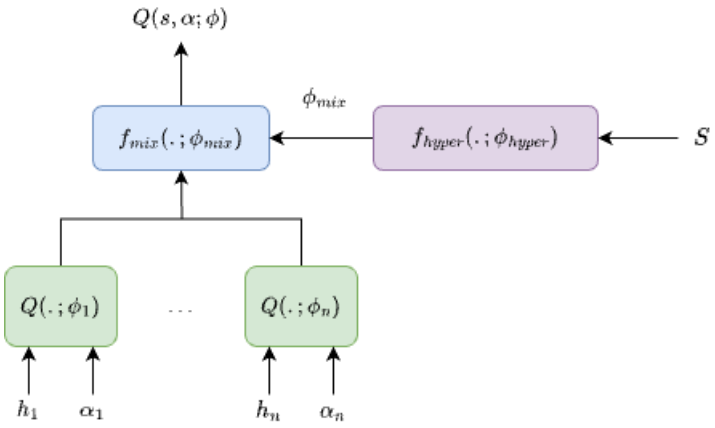

Figure 3.7: QMIX architecture

3.3.10 ATM

Agent **T**ransformer **M**emory (**ATM**) [74] network with a transformer-based memory aims to solve challenging multi-agent partial observable settings by considering that the multiagent observation consists of a number of object entities or the action space shows clear entity interactions. ATM network serves two main purposes:

1. Unified Processing: The transformer architecture is utilized to handle both the individual entities present in the environment and the memory component. This approach allows for a cohesive treatment of these different elements within the model.

2. Inspired by the human cognitive process of working memory—where a limited amount of information is temporarily held to guide decision-making—the ATM network maintains a memory of fixed capacity. This memory is dynamically updated using a schema akin to how humans update their working memory.

Furthermore, the ATM network takes into account the specific interactions between entities in the environment for each agent's actions. As each action performed by an agent involves particular entities, the ATM network analyzes the action space to introduce a semantic inductive bias. This is achieved by linking each action with its corresponding involved entity, which aids in predicting the state-action value or logit. In other words, the ATM network employs a mechanism to associate the actions an agent takes with the entities they interact with, enhancing the model's understanding of the context. The ATM framework is illustrated in Figure 3.8.

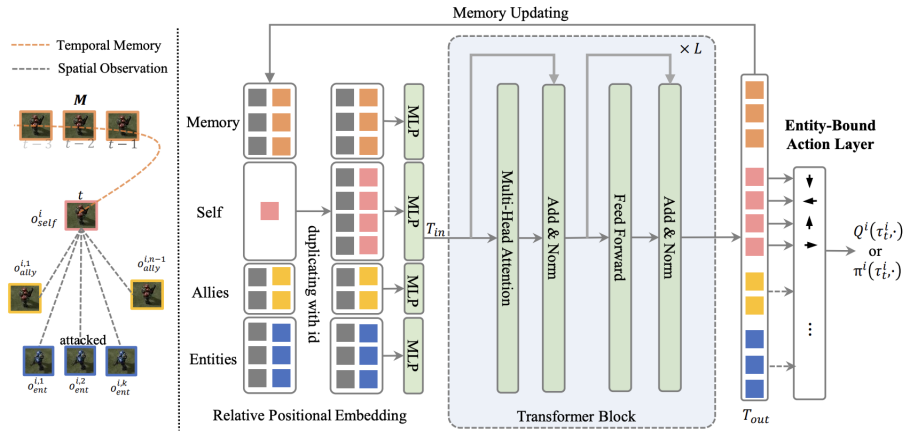

Figure 3.8: ATM architecture [74]

To handle partial observability in ATM, each agent has a slot-based memory where it stores past key information in τ^i, which is updated by the working memory mechanism. It also uses a fixed capacity memory buffer M in order not to apply the transformer on the whole trajectories. Furthermore, the Agent Transformer Memory (ATM) network extends its functionality by incorporating relative positional embeddings for both spatial entities and memory slots, denoted as T_{in}. These embeddings are then fed into the transformer block for processing. Through this process, ATM generates spatio-temporal embeddings labeled as T_{out}. These embeddings encapsulate information about the spatial layout and temporal dynamics of the environment.

With the spatio-temporal embeddings T_{out} as input, the ATM network proceeds to compute individual Q-values or policy logits. This computation involves the use of an Entity-Bound Action Layer, where the unique entity embeddings contained in T_{out} are semantically linked to specific actions. In essence, this mechanism associates distinct entity embeddings with corresponding actions, allowing the network to predict Q-values or policy logits for each action in a contextually relevant manner. For more details about ATM algorithm, Working Memory Updating Schema and the Entity-Bound Action Layer the interested reader can refer to the original paper [74].

3.3.11 MASER

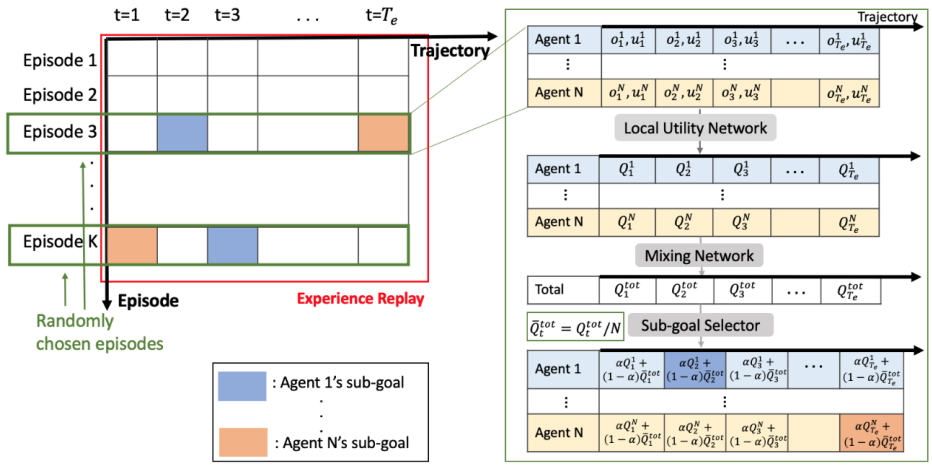

Figure 3.9: Diagram of subgoal determination from replay buffer. (Left) MASER chooses M episodes randomly from the replay buffer and each episode of length equal to T_e (Horizon) has sequential sample information of action and observation of all agents and extrinsic reward from $t = 1$ to $t = T_e$. (Right) The right side shows how to select a subgoal for each agent based on the current Q-function estimate.

Multi-**A**gent reinforcement learning with **S**ubgoals generated from **E**xperience **R**eplay buffer (**MASER**) is a framework proposed by Jeon et al. [34] for cooperative multi-agent reinforcement learning (MARL) in sparse reward settings. Given the prevalent hypothesis of training centralization and decentralized execution, alongside a uniform Q-value decomposition in Multi-Agent Reinforcement Learning (MARL), MASER employs an automated process to derive suitable subgoals for multiple agents using the experience replay buffer. This is achieved by taking into account both individual and total Q-values. Furthermore, MASER formulates distinctive intrinsic rewards for each agent, utilizing actionable representations pertinent to Q-learning. This approach aims to guide agents towards their respective subgoals while concurrently optimizing the overall action value across the group.

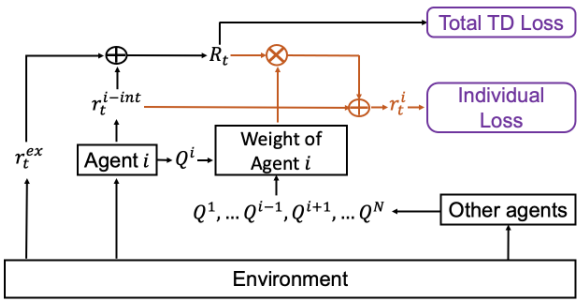

Figure 3.10: The MASER intrinsic motivation framework.

In particular, MASER assumes a finite joint action space and makes use of off-policy learning. Based on the previously mentioned Q-decomposition, MASER employs a block-wise approach with a block length equivalent to the episode horizon. It incorporates three core elements to effectively learn within environments characterized by sparse rewards:

- Subgoal generation and assignment: MASER identifies subgoals for agents using the experience replay buffer, eliminating the need for pre-defined subgoals based on domain expertise.
- Individual reward provision: MASER constructs unique rewards for local agents to guide them toward their respective subgoals while concurrently maximizing the collective return.
- Actionable distance informed by Q-learning: The intrinsic reward is determined by the Euclidean distance, enabling a measure of intrinsic reward that aligns with the principles of Q-learning (and thus it is based on QMIX).

The subgoal determination routine of MASER is depicted in Figure 3.9 and the high level MASER framework is illustrated in Figure 3.10. For a more detailed analysis of MASER, we encourage the interested reader to see the original work of Jeon et al. [34].

3.4 Variational Autoencoders

3.4.1 Problem Formulation

Let $X = \{x^{(i)}\}_{i=1}^{N}$ be a dataset containing N i.i.d samples of a continuous or discrete variable x, which is generated by a random variable z. To obtain a sample x^i we first have to generate a value z^i from a prior distribution $p_\theta(z)$ and then generate the value $x^(i)$ for the conditional distribution $p_\theta(x \mid z)$. We also make the assumption that the prior $p_\theta(z)$ and the likelihood $p_\theta(x \mid z)$ depend on the parameters θ and that their PDFs are differentiable almost everywhere w.r.t. both θ and z. However both the true parameters θ and the latent variable z^i are unknown. Our goal is to be able to find the marginal likelihood $p(x)$ in order to generate new samples form X, we have to encounter intractability issues. One example is that the computation of the integral $p_\theta(x) = \int p_\theta(z)p_\theta(x \mid z)dz$ is not feasible over all values of z or that the posterior $p(z \mid x)$ is unknown. We want to define an objective in order to find a maximum likelihood estimate of the parameters θ and approximate the posterior distribution in order to represent the data efficiently.

3.4.2 Variational lower bound

As we are interested in calculated the $p_\theta(x)$ we can start by applying the base rule:

$$p_\theta(x) = \frac{p_\theta(x \mid z)p_\theta(z)}{p_\theta(z \mid x)}$$

We can assume a Gaussian prior for z e.g. $p_\theta(z) \sim \mathcal{N}(\mu, \sigma^2)$, however we still cannot calculate $p_\theta(x)$ as the posterior is intractable. To mitigate this issue [37] propose to approximate the posterior distribution with an encoder network q, with parameters ϕ such that:

$$q_\phi(z \mid x) \approx p_\theta(z \mid x)$$

Taking the logarithm of the bayes rule and by performing simple calculations we have:

$$\begin{aligned}
\log p_\theta(x) &= \log \frac{p_\theta(x \mid z)p(z)}{p_\theta(z \mid x)} \\
&= \log \frac{p_\theta(x \mid z)p(z)q_\phi(z \mid x)}{p_\theta(z \mid x)q_\phi(z \mid x)} \\
&= \log p_\theta(x \mid z) + \log \frac{p(z)}{q_\phi(z \mid x)} + \log \frac{q_\phi(z \mid x)}{p_\theta(z \mid x)} \\
&= \log p_\theta(x \mid z) - \log \frac{q_\phi(z \mid x)}{p(z)} + \log \frac{q_\phi(z \mid x)}{p_\theta(z \mid x)}
\end{aligned} \tag{3.28}$$

By taking the expectation on both sided of the equation 3.28 we have:

$$\begin{aligned}
\log p_\theta(x) &= \mathbb{E}_{z\sim q_\phi(z|x)}[\log p_\theta(x)] \\
&= \mathbb{E}_{z\sim q_\phi(z|x)}[\log p_\theta(x \mid z)] - \mathbb{E}_{z\sim q_\phi(z|x)}\left[\log \frac{q_\phi(z \mid x)}{p(z)}\right] + \mathbb{E}_{z\sim q_\phi(z|x)}\left[\log \frac{q_\phi(z \mid x)}{p_\theta(z \mid x)}\right] \\
&= \mathbb{E}_{z\sim q_\phi(z|x)}[logp_\theta(x \mid z)] - D_{KL}(q_\phi(z \mid x), p(z)) + D_{KL}(q_\phi(z \mid x), p_\theta(z \mid x))
\end{aligned} \tag{3.29}$$

However the last KL-Divergence term is intractable, because the posterior is still unknown. Since that the KL-Divergence is always non-negative, we optimize a lower-bound of the previous equation:

$$\log p_\theta(x) \leq \mathbb{E}_{z\sim q_\phi(z|x)}[\log p_\theta(x \mid z)] - D_{KL}(q_\phi(z \mid x), p(z)) \tag{3.30}$$

The above lower bound is known in the literature as *Variarional Lower Bound*, or *Evidence Lower Bound* (ELBO). Therefore, we write the VAE loss, denoted by L_{ELBO}, that we aim to minimize with respect to parameters θ and ϕ:

$$L_{ELBO}(\theta, \phi; x) = \mathbb{E}_{z\sim q_\phi(z|x)}[\log p_\theta(x \mid z)] - D_{KL}(q_\phi(z \mid x), p(z)) \tag{3.31}$$

Analyzing our optimization objective we observe that the first term constitutes a reconstruction term, measuring how efficient is the reconstruction performed by the decoder while the second term functions as a regularizer pushing the approximate posterior closer to the prior. As it is often the case that prior is the standard normal distribution. In practice, both θ and ϕ are approximated by deep neural networks, such as multilayer perceptrons (MLPs).

3.4.3 Reparameterization trick

To implement and optimize the variational lower bound we need to generate values for the latent variable z in a way that is differentiable and allows us to perform stochastic gradient descent. The proposed solution in [37] is as follows: first sample a noise variable ϵ from a simple distribution $p(\epsilon)$ (e.g. the standard Normal distribution $\mathcal{N}(0,1)$) $\epsilon \sim p(\epsilon)$ and then apply a deterministic transformation function $g_\phi(\epsilon, x)$ to map the random noise into a more complex distribution, $z = g_\phi(\epsilon, x)$. In the case of Gaussian variables the reparametrization trick can be a $z = g_{\mu,\sigma}(\epsilon) = \mu + \sigma\epsilon$ where $\epsilon \sim \mathcal{N}(0,1)$ instead of $z \sim q_{\mu,\sigma}(z) = \mathcal{N}(\mu, \sigma)$.

Chapter 4

Proposed Framework

4.1 Problem Statement and Motivation

In this thesis, we aim to deal with complex partially-observable multi-agent environments that require effective cooperation from the agents in order to be solved. The cooperation may be found in terms of coordination of the agents' knowledge about the environment dynamics and their team goals. From the structure of the game environments that we tackle in our experiments, all agents receive the same reward signal at each timestep of the simulating environment, which describes the shared team goal(s) along a single game; e.g. whether the players are close to win the game, or arrive to their favourite landmarks. As we discussed in the previous chapter, the theoretical framework that models the above problem setting is the DecPOMDP. Moreover, we follow the Centralized-Training-Decentralized-Execution (CTDE) schema which has been widely used for benchmarking MARL algorithms in the literature. Based on the structure of the DecPOMDP and the concept of the CTDE schema, we identify the following issues that strongly motivated us for this thesis.

The shared reward signals that the agents receive in a DecPOMDP could mislead the training process of the agents, rendering many MARL tasks to be very challenging to solve. Since both the immediate rewards and the overall team goals are shared, it is difficult for each agent to identify the contribution of their own policy to the received rewards. This difficulty could make some agents be "lazy" to explore effectively their local observation space even if the received team reward is relatively high due to existence of good policies of some other agents. Therefore, the lazy agents would opt for actions that are suboptimal, because their estimated value would be large on average. In other words, this issue could lead agents to suboptimal approximate Nash equilibria, a problem that has been labelled as *Relative Overgeneralization* in the MARL literature. The above issue is deteriorated under the existence of sparse reward settings, in which the agents receive good rewards only when they have reached some good states; e.g. when they solve the task entirely. In such cases, the agents should be urged to adopt more exploratory policies, with each one of them aiming at exploring effectively their local observation space in a both individual

and cooperative manner.

Under the CTDE schema, in testing the agents should solve the task given only their own observations (since execution is decentralized), although they can use any information of the other agents for coordination when learning their individual policies during the centralized training. Nevertheless, it is not straightforward how to use the information of the other agents during the centralized training. Recently, there is a massive interest in communication MARL methods, which aim to use efficient centralized communication among agents via messages and protocols in order to solve complex cooperative tasks. Communication methods address the problem of agent coordination by incorporating into each agent's policy some information sharing (such as shared observations and/or actions) by learning a communication network of the agents that optimizes the performance of the whole team. In this way, each agent learns an enhanced policy that also takes explicitly into account what observations/actions their fellow connected players observe/take. However, such methods are not built upon the CTDE schema, because they do not use decentralized execution; the agents always use some information from some other agents as part of their policies in testing.

One major question that we are interested in addressing in this thesis is the following: *Can we combine the CTDE schema with the benefits of the communication methods to train agents able to perform better in difficult tasks?* Moreover, considering the remarkable performance of the communication methods in difficult benchmark environments, we are also interested in addressing the following question: *Can we train agents under the CTDE schema with enhanced policies, i.e. exploiting some kind of information sharing, and use this information sharing for coordinated exploration in difficult sparse reward tasks?*

To address the aforementioned questions, in this thesis we propose **C**ount-based **A**gent **M**odelling (**CAM**), a novel agent modelling approach which utilizes (a) variational inference and self-supervised learning for generating information sharing among the agents through agent (opponent) modelling, and (b) a count-based intrinsic reward based on the generated information sharing in order to allow agents to better explore difficult multi-agent environments with/without sparse rewards. The proposed framework is built upon MAA2C due to its remarkable performance in various benchmarks [56]. The overall framework can be separated into two distinct parts. The first one involves learning the variational embeddings that will be used in addition with the observations of each agent as an input to their policy in order to provide an enriched representation of the game's state and enable the agents to discover better policies. The second one involves the introduction of an intrinsic reward based on the aforementioned embedding utilizing the Count-Based exploration algorithm described in [66] which will act as an exploration bonus in sparse reward settings.

4.2 CAM: Learning Variational Embeddings for Opponent Modelling

In the proposed method, each agent adopts systematic opponent modelling by aiming to predict the observations of the other agents at a timestep given only the agent's own observations. As in [54], each agent i models the observations of the other agents, denoted by o_t^{-i}, via an internal belief, $p(o_t^{-i} \mid o_t^i; \theta_i)$, parameterized by the random variable Θ_i with values $\theta_i \in \mathbf{\Theta_i}$. This belief describes how the agent perceives her unseen parts of the full state based on her own partial observability. We assume (a) a prior $p(\theta_i)$ over θ_i, (b) the existence of some latent variables z^i in the space $\mathcal{Z}$ and (c) at each timestep t the latent variables z^i contain information about the opponent modelling of agent i.

To learn a good representation (embedding) for the opponent modelling latent variable z^i, we utilize a variational encoder (p_{θ_i})-decoder (q_{ϕ_i}) for each agent which aims to capture the observations of the other agents (o_t^{-i}) at timestep t given only the agent's own observations (o_t^{-i}). In parallel, at each timestep t, we enhance the agent's policy π_t^i by concatenating the original policy input (i.e. o_t^i) with z_t^i. One could assume that we ideally hope that the encoder-decoder could predict perfectly the observations of the other agents, so that z_t^i could be as much informative about what the other players observe, as it could be. However, as it was noted by recent work [25], using the full information of the state (even when utilizing a compressed embedding) to enhance the agent's policy does not always increase the performance due to abundant state information unnecessary to the agent. We note that similar results can be also found in our experiments in the next chapter of this thesis.

Aiming to learn meaningful latent embeddings for representing z_t^i, we mitigate the use of unnecessary state information by first filtering out the less informative features of o_t^{-i}, which the agent's encoder-decoder tries to predict. Similar to [25], we make use of additional learnable weights, $w^i = \sigma(\phi_w(o^i))$, one weight for each feature of o^{-i}; we use an MLP as ϕ_w and σ as the sigmoid activation function in order to keep the value of each entry between $[0, 1]$. Intuitively, w^i represents the importance of each feature of o^{-i} to agent i. For each agent i, we train a variational encoder-decoder by minimizing the following *self-supervised* reconstruction loss:

$$L_{rec}(\theta_i, \phi_i, \phi_w^i) = (\tilde{w}^i \cdot o^{-i} - w^i \cdot \hat{o}^{-i})^2 \tag{4.1}$$

where $\hat{o}^{-i}$ are the predicted observations of the encoder-decoder (using the reparameterization trick), $\hat{o}^{-i}$ are the true target observations of the other agents, w^i are the predicted weight filters and $\cdot$ means element-wise multiplication. To stabilize the training of w^i, similarly to standard deep RL updates, such as in [51, 68], we use target weights $\tilde{w}^i$ to filter out the target observations $\hat{o}^{-i}$ with their values being assigned to previous values of w^i and remaining fixed during the update of w^i.

The reconstruction loss in 4.1 generalizes the standard reconstruction loss of a VAE

(see 3.31). In particular, if both w^i and $\tilde{w}^i$ are equal to the identity matrix, then we receive exactly the reconstruction loss term of 3.31. In the general case, the reconstruction loss of 4.1 accounts for both (a) reconstructing effectively o^{-i} and (b) learning w^i in a self-supervised manner.

To ensure that the values of w^i do not vanish to zero throughout all dimensions (note that in such case the reconstruction loss would be zero), we also add the following L2 normalization loss term:

$$L_{norm}(\phi_w^i) = -||w^i||^2 \tag{4.2}$$

To further ensure that the values of w^i are meaningful for solving the given task, and thus for maximizing the return, we add the following regularization term for agent i, derived by the critic loss of the standard MAA2C:

$$L_{critic}(\phi_w^i, k_i) = [r_t + V_{k_i'}(\hat{s}_t) - V_{k_i}(\hat{s}_t)]^2 \tag{4.3}$$

where r_t is the received reward at timestep t, s_t is the state of the environment at timestep t, V_{k_i} is the state value parameterized by a neural network k_i, $V_{k_i'}(s)$ is the state value parameterized by a target network k_i' and $\hat{s}_t = o_t \oplus (w^i \cdot o_t^{-1})$ is the predicted state generated by agent i's belief, where $\oplus$ means vector concatenation. We note that $V_{k_i}(\hat{s})$ (and thus $V_{k_i'}(\hat{s})$) is *not* trained as an auxiliary task; in the sense that $V_{k_i}(\hat{s})$ is also utilized as the centralized critic of MAA2C and is not conditioned on true state s.

As in VAE [37], we also add a regularization term which aims to minimize the KL divergence between the posterior and the prior (which is typically assumed to be the normal distribution), as it follows:

$$L_{KL}(\phi_i) = D_{KL}(q_{\phi_i}(z^i|o^i)||p_{\theta_i}(z^i) = \mathcal{N}(0,1)) \tag{4.4}$$

Overall, the total loss of CAM for learning variational embeddings for agent i, denoted by L_{TOTAL}, is the following:

$$L_{TOTAL}(\theta_i, \phi_i, \phi_w^i, k_i) = L_{rec}(\theta_i, \phi_i, \phi_w^i) + L_{norm}(\phi_w^i) + L_{critic}(\phi_w^i, k_i) + L_{KL}(\phi_i) \tag{4.5}$$

Each agent's variational encoder-decoder receives the observations of the agent i, o_t^i, at each timestep t and aims to learn a meaningful latent embedding having as target the observations of the other agents, o_t^{-i}, multiplied (in an element-wise manner) with the weight vector w^i, in order to allow each agent to focus only on the information that might be relevant. Here, we should highlight the importance of the weight vector, as if it did not exist then the target of the encoder-decoder grows linearly with the number of agents and the learning procedure is not expected to result in a meaningful embedding. Finally, at each timestep t, each agent i uses an enhanced policy $\pi^i(\cdot \mid o_t^i, z_t^i)$ which is conditioned on both o_t^i and the predicted variational embedding z_t^i.

To train the agents' encoder-decoders with i.i.d data, in practice we sample off-policy transitions from the received trajectories of the agents, in order to break the high correlation of the transitions within the same trajectories. We note that our base MAA2C model is trained using on-policy data, as we discussed in the previous chapter of this thesis.

4.3 CAM: Count-based Intrinsic Motivation

In this section, we introduce the second part of CAM which aims at designing effective intrinsic rewards for better exploration in complex sparse reward tasks. However, as we discussed above, it is important for the agents to coordinate their exploration in an efficient way, since the random initial policies (which are due to the entropy term in the PG loss) could be unsuccessful to reach valuable states. In other words, the high value states could require a sequence of joint actions that is very difficult to be found through random independent policies.

To further empower the agents' joint policy to reach high value states in sparse reward settings, we exploit the expressive power of the predicted variational embeddings z^i, which capture the belief of agent i about the partial observable full state, and design intrinsic rewards which naturally account for both *individual* and *cooperative* benefit. Since z^i is informative about what the other players observe, we propose to encourage each agent i to reach observations that lead to novel z^i and thus to explicitly explore different parts of their own local state (observation) space, associated with different z^i, for their individual benefit. However, by doing so, agent i also results in different targets of agent modelling for the other agents, $-i$; i.e. the other agents, $-i$, now should learn from unseen agent modelling targets due to novel observations received from agent i. Consequently, agent i implicitly forces the other agents, $-i$, to better explore their agent modelling target space helping them improve their beliefs, i.e. z^{-i}, about the full state space. On the other hand, by making the other agents, $-i$, as well, explore regions of their local state space that could lead to more informative z^i, the proposed intrinsic motivation approach implicitly encourages *coordinated exploration* for cooperative benefit.

To ensure such goals, we adopt the Count-Based hashing intrinsic reward schema which uses the SimHash algorithm proposed in [66]. The description of the SimHash algorithm can be found in Algorithm 8. Overall, we present the overview of the proposed framework in Figure 4.1. The SimHash algorithm discretizes the embeddings (or the state space in general) utilizing a hash function $H : S \to \mathbb{Z}$. Then an exploration bonus $\hat{r} : S \to \mathbb{R}$ is calculated as follows:

$$\hat{r} = \frac{\beta}{\sqrt{n(H(s_m))}} \tag{4.6}$$

and added to the reward function. The $\beta \in \mathbb{R}_{\geq 0}$ term describes the bonus coefficient, while the initial counts $n(\cdot)$ are set to zero for the whole range of H. During training, for every visited embedding o_t^i at timestep t, $n(H(o_t^i))$ is increased by one and agent i is trained

Algorithm 8 Count-based exploration through static hashing, using SimHash

Define state preprocessor $g : S \to \mathbb{R}^D$

Initialize $A \in \mathbb{R}^{k \times D}$ with entries drawn i.i.d. from the standard Gaussian $\mathcal{N}(0, 1)$

Initialize a hash table with values $n(\cdot) \equiv 0$

For each iteration j **do**

Collect a set of state-action samples $\{(s_m, a_m)\}_{m=0}^{M}$ with policy π

Compute hash codes using SimHash, $\phi(s_m) = \text{sgn}(Ag(s_m))$

Update the hash table counts $\forall m : 0 \leq m \leq M$ as $n(H(s_m)) \leftarrow n(H(s_m)) + 1$

Update the policy π using rewards $\left\{ r(s_m, a_m) + \frac{\beta}{\sqrt{n(H(s_m))}} \right\}_{m=0}^{M}$ with any RL algorithm

End for

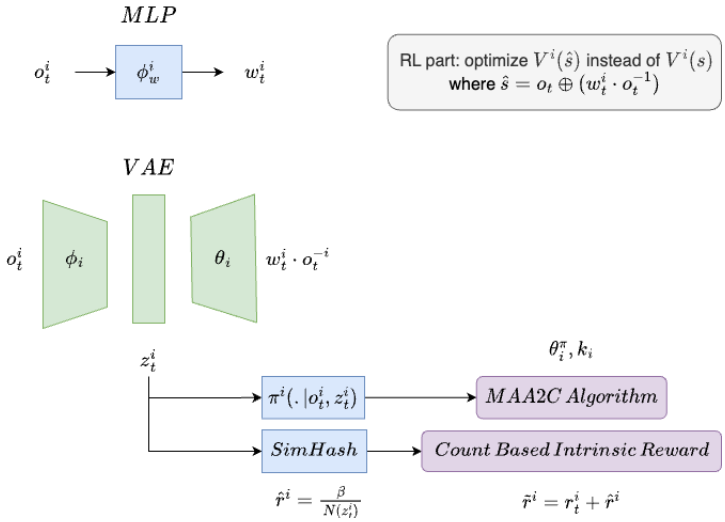

Figure 4.1: The Count-Based Agent Modelling (CAM) framework

with rewards $\tilde{r}_t^i = (r_t^i + \hat{r}_t^i)$, where r_t^i is the original reward of agent i (which is the same for all agents in cooperative games) and $\hat{r}_t^i$ is the intrinsic reward of agent i.

The performance of the algorithm is highly dependent on the choice of hash function H, which should ideally count differently distant embeddings and merge similar ones. The decision is made by the SimHash algorithm which measures similarity by angular distance using the following function:

$$H(s_m) = \text{sgn}(Ag(s_m)) \in \{-1, 1\}^k, \tag{4.7}$$

where $g : S \to \mathbb{R}^D$ constitutes a preprocessing function and A is a $k \times D$ matrix with i.i.d entries from the standard Gaussian distribution $\mathcal{N}(0, 1)$. We chose the SimHash algorithm which in practice performs a simple mathematical transformation of the input instead of a more elaborate approach, due to its simplicity and its ability to allow nearby inputs to be transformed into nearby hash values.

To make the intrinsic rewards be more stable and guide exploration effectively throughout the training process of each agent, we should make z^i more stable; i.e. the parameters of the encoder-decoders to not change dramatically between successive training episodes. To this aim, we opt for performing periodical hard updates of the encoder-decoders using a fixed large number of training episodes as an update period, which is a hyperparameter for a given task. We present the overview of the CAM algorithm in Algorithm 9.

Algorithm 9 CAM algorithm

Initialize n actor networks with random parameters $\theta_1^\pi, \ldots, \theta_n^\pi$
Initialize n critic network with random parameters $k_1, \ldots, k_n$
Initialize n VAE networks with random parameters $\theta_1, \ldots, \theta_n$ and $\phi_1, \ldots, \phi_n$
Initialize n weight networks with random parameters $\phi_w^1, \ldots, \phi_w^n$
Collect environment observations $o_1^1, \ldots, o_1^n$
for time step $t = 1, \ldots,$ EpisodeHorizon **do**
    for agent $i = 1, \ldots, n$ **do**
        Sample embedding z_i^t from variational encoder q_{ϕ_i}
        Sample actions a_i^t from $\pi(a_t^i \mid h_t^i, z_t^i; \theta_i^\pi)$
Execute actions and collect states s_{t+1}, observations o_{t+1}^i and rewards r_t^i.
for time step $t = 1, \ldots,$ EpisodeHorizon **do**
    for agent $i = 1, \ldots, n$ **do**
        Calculate the count-based intrinsic reward $\hat{r}_t^i$ based on algorithm 8
        and use $\tilde{r}_i = r_t^i + \hat{r}_t^i$ as the reward of each agent
for agent $i = 1, \ldots, n$ **do**
    Update parameters θ_i^π by minimizing the loss of the equation 3.8
    Update parameters k_i by minimizing the loss of the equation 4.3
for agent $i = 1, \ldots, n$ **do**
    Update parameters $\phi_w^i, \theta_i, \phi_i$ by minimizing the loss of the equation 4.5

Chapter 5

Evaluation

In this chapter we discuss the evaluation of the proposed framework (CAM) using (a) the multi-agent particle environment (MPE) proposed by [45], (b) the Level-based Foraging (LBF) proposed by [56]. Both environments include cooperative and competitive tasks, while the latter one also offers tasks with sparse rewards. We aim to benchmark CAM on tasks from the aforementioned testbeds and compare its performance to baseline algorithms that we described in Chapter 2.

5.1 Experimental Setup

First, we start with the discussion of the experimental setup that we follow in our experiments. We make use of two benchmarking testbeds that have been widely used for evaluating the performance of MARL algorithms and frameworks. In all tasks of both testbeds, instead of visual information, the agents receive high-level features as observations about the current state of the task.

5.1.1 Multi-Agent Particle Environment (MPE)

The Multi-Agent Particle Environment (MPE) is one of the first environments designed for benchmarking MARL algorithms. All different task environments, that are included within MPE, entail landmarks in which the agents should reach for performing a certain task. All MPE tasks require the agents to operate in a continuous two dimensional area. Agents also receive rewards in every timestep that provide useful information for their actions. Finally, the action space in all environments are discrete.

5.1.1.1 Spread: A Cooperative Navigation task

The cooperative navigation task involves an equal number (N) of moving agents and static landmarks. The shared goal of the agents is to navigate to a landmark that is not occupied by another agent while avoiding collisions with each other. The environment is depicted in Figure 5.1.

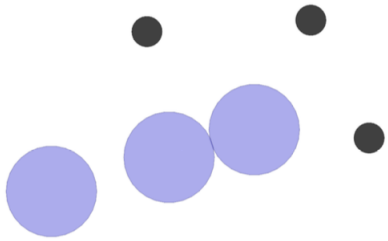

Figure 5.1: Spread: A Cooperative Navigation task

- **Observation space**: All agents receive their current velocity, position, and the distance between landmarks and other agent positions as their input.
- **Action space**: The action space is discrete and involves 5 actions: standing still, moving right, left, up and down.
- **Reward**: All agents receive the same reward, which includes the summed negative minimum distance to any other agent. Additionally, the collisions between any two agents are with a negative reward of -1.

In our experiments, we make use of $N = \{3, 4, 6\}$, while we note that the default value, as it is in the official github repository of MPE, for N equals 3.

5.1.1.2 Speaker Listener: A Cooperative Communication task

In this environment two agents (speaker and listener) should collaborate to identify a goal among three possible landmarks and move towards it. The speaker agent does not have the ability to move and only receives information about the goal, while the listening agent should reach the specified communicated goal by listening to the speaker agent. The environment is depicted in Figure 5.2.

- **Observation Space**: The speaker agent observes only the colour of the goal landmark which is represented as three numeric values. The listener agent receives as observations its velocity, relative landmark positions as well as the communication of the speaking agent.
- **Action Space**: Similar to other MAPE environments the listener's action is space is discrete and includes five actions: (standing still, moving right, left, up and down). The speaker's action space is also discrete however it includes three actions to communicate the goal to the listener.

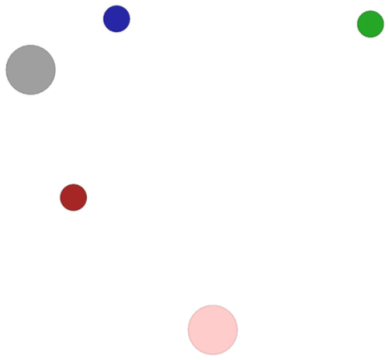

Figure 5.2: SpeakerListener: A Cooperative Communication task

- **Reward**: The reward is common for both agents and it is calculated as the negative square Euclidean distance between listener's position and the goal landmark's position.

5.1.2 Level-based Foraging (LBF)

In this environment, agents navigate a grid world aiming at collecting food by cooperating or competing with other agents. In more detail, each agent has an assigned level, while food is randomly positioned in the grid world, each one having a level on its own. Agents navigate the grid world trying to collect food, which can be successfully accomplished only if the sum of the levels of the agents involved in loading is equal to or higher than the level of the food. In that case, agents are rewarded with the level of the food they collected, divided by their level. The main challenge of this environment is that rewards can be sparse since the agents are rewarded only if they participate in the food collection. The environment is depicted in Figure 5.3.

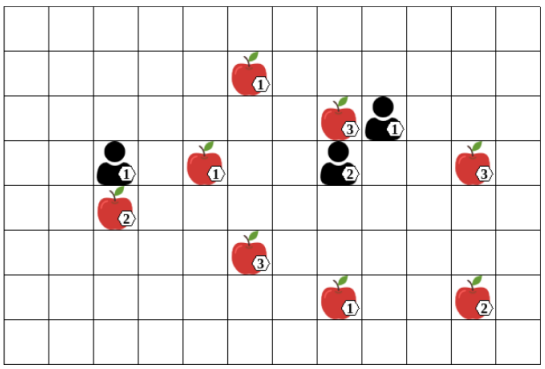

Figure 5.3: Level-based Foraging

- **Observation space**: All agents receive triplets of the form $(x, y, level)$ as observa-

tions. The number of triples is equal to the sum of the number of foods and the number of agents in the environment. The observations begin with the food triples and are followed by the agents triples. Each triplet contains the x and y coordinates and level of each food item or agent.

- **Action space**: The action space is discrete and involves six actions: standing still, move North, move South, move West, move East, pickup.

- **Reward**: In Level-Based Foraging environment agents are rewarded only when they pick up food. This reward depends on booth the level of the collected food as well as the level of each contributing agent and can be defined for the agent i as

$$r^i = \frac{\text{FoodLevel} * \text{AgentLevel}}{\sum \text{FoodLevels} \sum \text{LoadingAgentsLevel}}$$

  Also the rewards are normalized in order for the sum of all agent's returns on a solved episode to equal to one.

5.2 Results

In this section we discuss the results of the proposed framework on the MPE and LBF MARL testbeds, in which we manage to achieve good performance over the selected baseline methods. In the following experimental analysis, we illustrate the performance of the evaluated MARL algorithms in the figures (plots) below. In all figures, the horizontal axis shows the time progress (in timesteps) of the training process of the algorithms and the vertical axis measures the total episodic reward in testing obtained during a fixed training timestep. In particular, to measure the total episodic reward, every 50000 training timesteps, we evaluate the MARL algorithm on 10 parallel random environment instances (i.e. 10 environments generated from 10 random seeds) and take the mean total episodic reward obtained by running the policy in these environments. In all experiments, we set the time training horizon (i.e. the horizontal axis of each plot) equal to 10 million timesteps. We also show the mean total episodic reward without any normalization, so that the results to be easily replicated.

5.2.1 Results on MPE

Starting with the MPE testbed, we compare CAM to the following baselines: COMA, MAA2C and ATM (all of which are based on policy gradient updates, presented in Chapter 2). We make use of four environmental benchmark settings, namely Spread (with N taking values $\{3, 4, 6\}$) and SpeakerListener. Since all environments do not have sparse reward settings, we do not use any intrinsic motivation in CAM (i.e. we utilize $\beta = 0$). This enables a fair comparison with baselines which do not use either exploration-based schemas but rely on the policy objectives and their overall neural architecture. We set the time horizon for the MPE tasks equal to 25 timesteps, as it is defined in the official implementation.

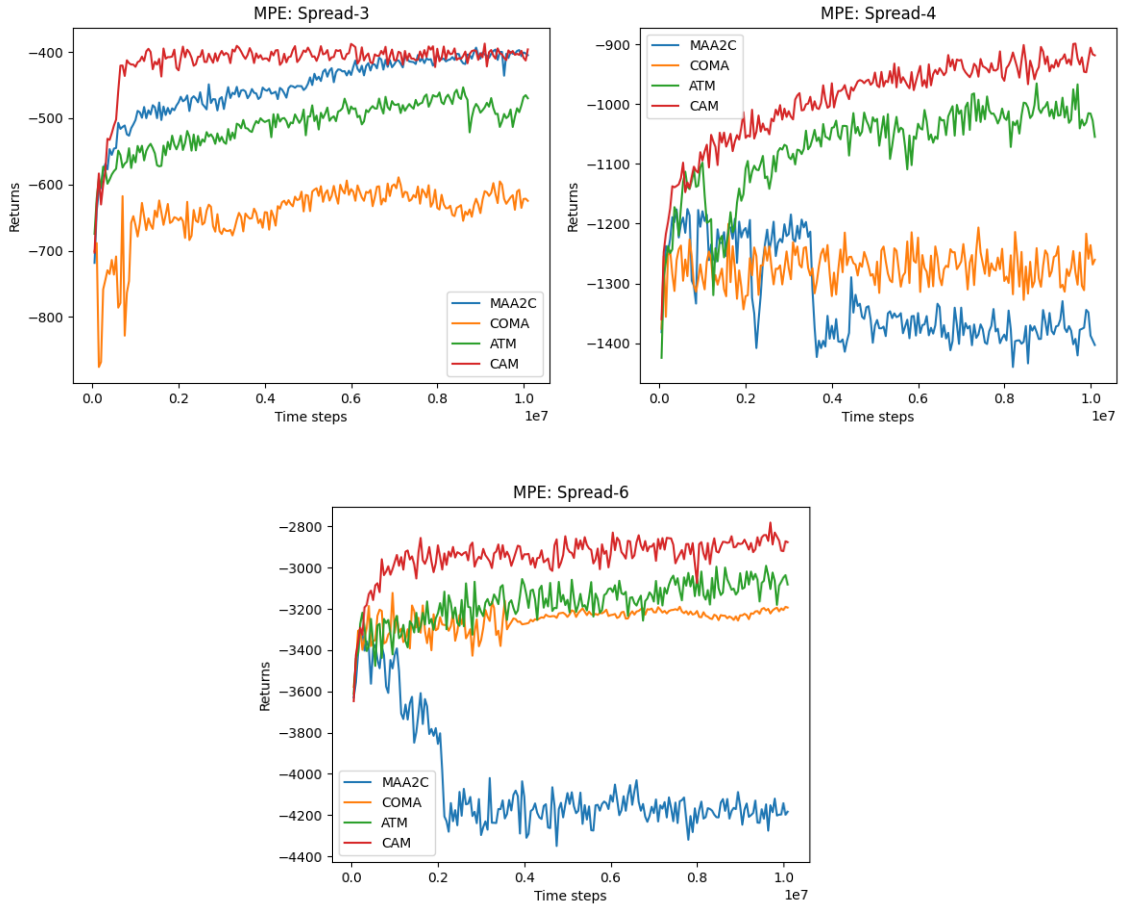

Figure 5.4: Results on MPE: Spread with number of agents being equal to 3 (top left), 4 (top right) and 6 (bottom)

In Figure 5.4, we illustrate the comparison of the proposed framework (CAM) with the baselines in the Spread settings. We observe that CAM (in red) outperforms all baselines in all Spread environments; CAM is able to consistently reach higher total episodic reward in significantly less time steps. In particular, except for Spread-3 in which MAA2C also is able to reach score similar to CAM, yet it needed more than 5 million training steps to do so, in the other Spread settings, CAM significantly outperforms the baselines. It is worth noting that ATM shows better performance than MAA2C, which is its backbone (as it is for CAM as well) in Spread-4 and Spread-5, in which MAA2C totally fails to solve the task. As for COMA, it also totally fails to solve the task in all Spread environments. Similar results we obtain in SpeakerListener (see Figure 5.5) as well; CAM needs around 2 million timesteps to solve the task, whereas ATM and MAA2C need more than 4 million timesteps. Again COMA fails to solve the MPE task, yet it shows some improvement with respect to its starting point.

Therefore, since CAM is able to outperform all other evaluated policy gradient based MARL methods in these environments, it may also be able to learn effective representa-

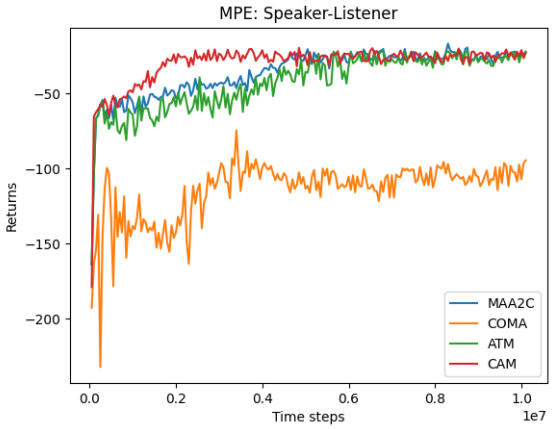

Figure 5.5: Results on MPE: SpeakerListener

tions, due to agent modelling, that help to improve dramatically the performance of its backbone (MAA2C). Although, CAM never fully reconstructs the observations of the other agents (due to the partial observability of the underlying environment and the use of the self-supervised weights), each agent models the others in a meaningful manner that boosts the overall team performance. As we highlight in our ablation study next, this improvement is also mainly attributed to the use of the self-supervised learnable weights, which aim to proportionally select which features of the other agents are more important for facilitating the training process of each agent and also for boosting the team performance.

5.2.2 Results on LBF

We now proceed to the LBF testbed, where we compare CAM to the previous baselines and also to MASER, an intrinsic motivation based framework that has achieved state-of-the-art performance in various sparse reward settings. We benchmark these algorithms in four LBF environments, namely: (a) $7 \times 7, 3p, 2f$, (b) $8 \times 8, 3p, 2f$, (c) $11 \times 11, 4p, 3f$ and (d) $12 \times 12, 2p, 2f$, all of which are sparse-reward tasks. We note that solving an LBF task corresponds to achieving total episodic reward equal to 1. We note that we now use the full version of CAM, that is including the use of the intrinsic rewards. We set the time horizon for the MPE tasks equal to 50 timesteps, as it is defined in the official implementation.

In Figure 5.6, we illustrate the performance of the evaluated algorithms in the aforementioned LBF tasks. We observe that CAM is able to consistently solve all tasks always achieving optimal total episodic rewards. On the other hand, except for the $8 \times 8, 3p, 2f$ task, both ATM and MAA2C show some improvement over COMA, which fails completely to solve the tasks. MAA2C solves the $7 \times 7, 3p, 2f$ and $12 \times 12, 2p, 2f$ (similar to ATM as well) tasks; however it needed much more timesteps than CAM to reach that performance (CAM needed less than half the timesteps to converge to a good policy). Remarkably, CAM also achieved to solve $8 \times 8, 3p, 2f$, which is the most challenging environment that we evaluated, in the sense that all other algorithms completely failed to increase their total

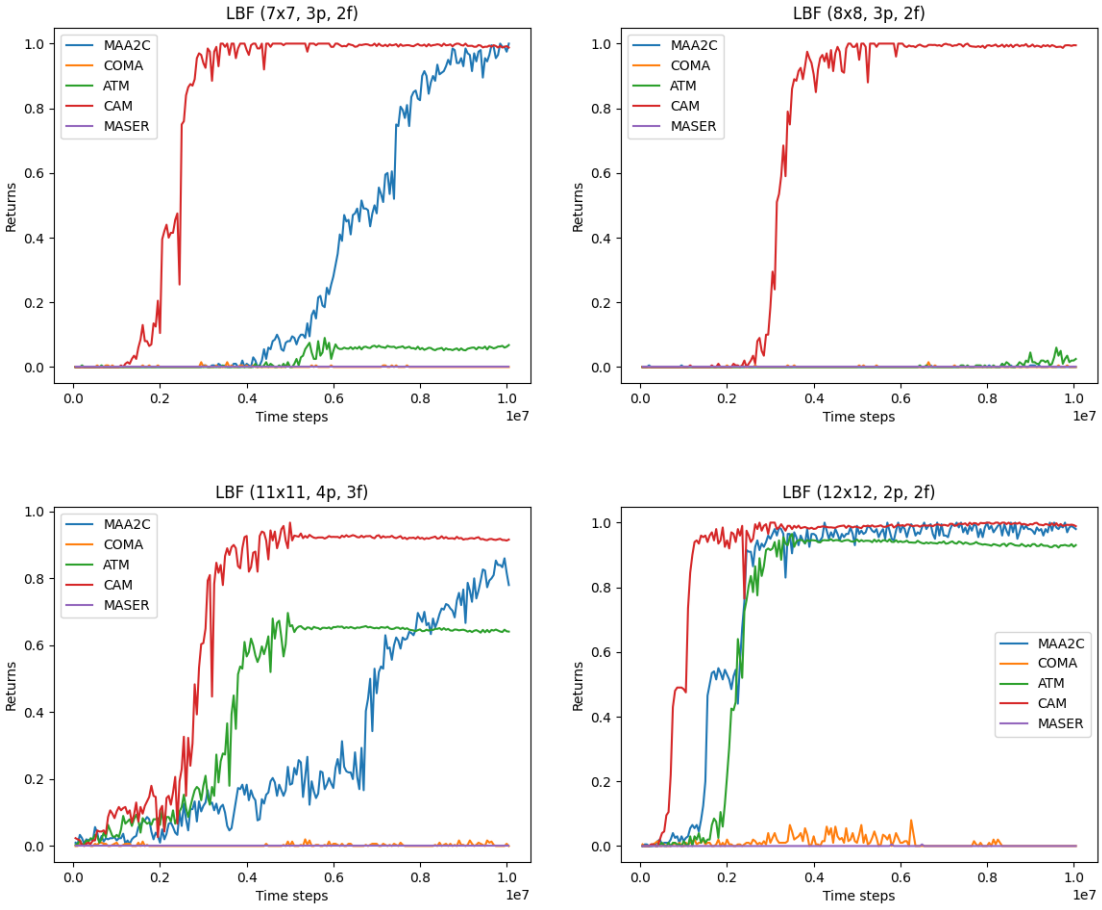

Figure 5.6: Results on LBF tasks

episodic reward. As for MASER, similar to COMA, it completely failed to increase the total episodic reward. We attribute this to the fact that MASER is an off-policy algorithm based on QMIX, and thus requires more training to find good policies, and was also probably designed primarily for solving sparse reward tasks from the SMAC testbed.

5.3 Ablation Study

5.3.1 Justifying the design choices in CAM

In this section we perform an ablation study to justify the design choices in CAM's architecture and design. The experiments are based on the LBF environment as it requires our full approach to be solved and specifically on its $8 \times 8, 3p, 2f$ version, which is one of the most challenging tasks. The ablated versions are (a) CAM without the use of the weights w, and (b) CAM without neither the use of the weights w nor the intrinsic reward. The results of the study are depicted in Figure 5.7. We observe that only the full approach is able to solve the environment, while the ablated versions are unable to find even a sub-optimal solution for the problem, retaining a near zero reward throughout the whole

training.

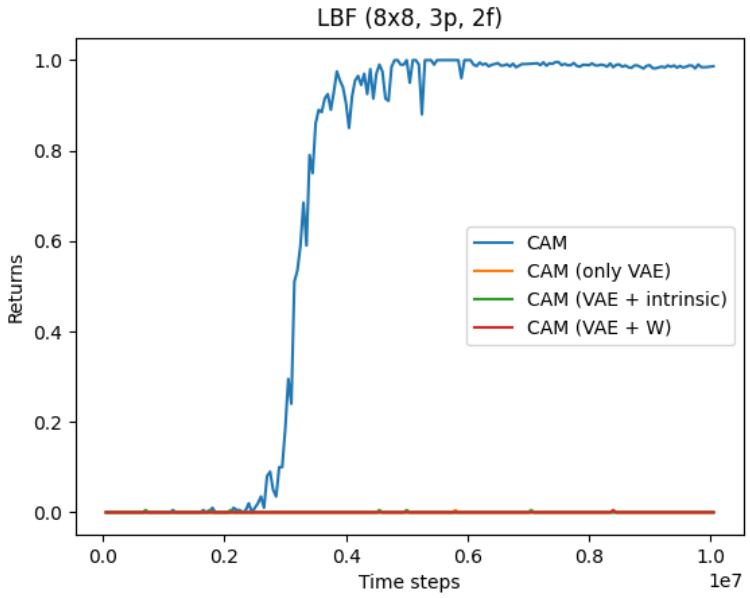

Figure 5.7: Ablation Study on LBF environment

5.3.2 CAM vs Full Communication

To further evaluate our approach, we perform another ablation study on the Spread (4 agents) of the MPE testbed. Having ensured that all parts of our algorithm are essential for its functionality, we now compare its performance against a fully centralized version of MAA2C which uses the full state information in both the actor and the critic networks. This approach can be considered as a full communication approach because the agents have exact knowledge of the full state during both training and execution. The results of this experiment are illustrated in Figure 5.8. The results are crucial as they show that allowing the agents to have full knowledge of the multi-agent state does not perform better than our approach; an observation also observed in [25], though in different experimental environments. This is because giving the full state to each individual agent policy is not useful to the agent due to the abundancy of meaningless information, which is not useful for their training, but also increases the input dimensionality of the policy networks linearly with the number of agents. On the other hand, our approach provides embeddings of fixed dimensionality, much lower than that of the full state, which allows the policy networks to infer good representations that lead to good policies. Moreover, the embeddings generated by the proposed framework are able to deliver a good performance without requiring any sort of communication among the agents during execution, since the whole learning process follows the CTDE schema and the communication has been replaced by agent modelling.

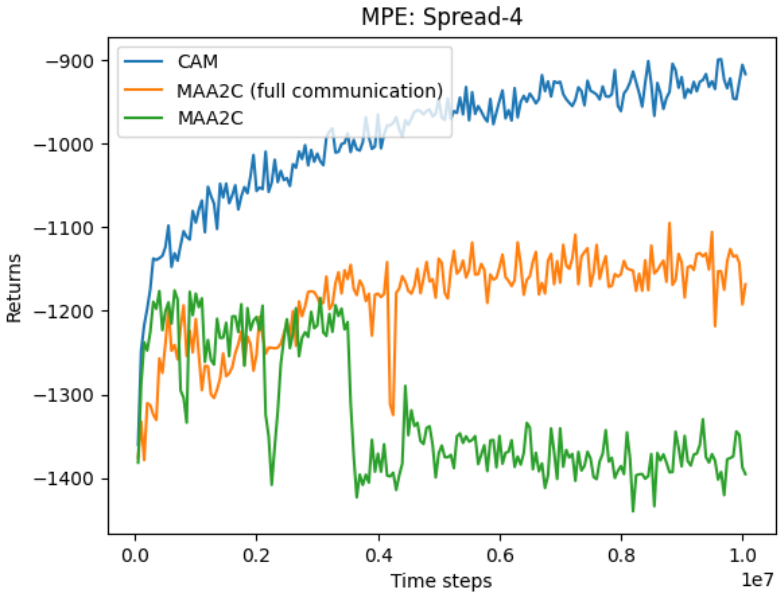

Figure 5.8: Ablation Study on MPE environment

5.3.3 Embedding Visualization

At this point, our aim is to visually represent the embeddings generated by the Variational Encoder-Decoder for each individual agent. The intention behind this visualization is to substantiate the meaningfulness and relevance of these embeddings. To achive that we will utilize the t-SNE technique [67] which aims to visualize high-dimensional data by giving each datapoint a location in a two or three-dimensional map. The 2D visualization produced by t-SNE can reveal certain patterns and relationships in your high-dimensional data that might be difficult to discern directly from the original data. While t-SNE is primarily a tool for visualization and exploratory data analysis, it can provide insights into the structure, clusters, and relative distances of your embeddings. Here's what you can gather from the 2D t-SNE visualization:

1. Clustering: Points that are close to each other in the t-SNE plot represent embeddings that are similar in the original high-dimensional space. If embeddings belonging to the same agent tend to cluster together, it indicates that the embeddings from each agent have distinctive characteristics that allow them to be separated.

2. Agent Differentiation: The color-coded points help you distinguish embeddings from different agents. If you see well-separated clusters of points with the same color, it suggests that the embeddings from each agent are significantly different from each other.

3. Embedding Variability: The spread or concentration of points in the t-SNE plot can give you an idea of the variability within each agent's embeddings. A tightly clustered

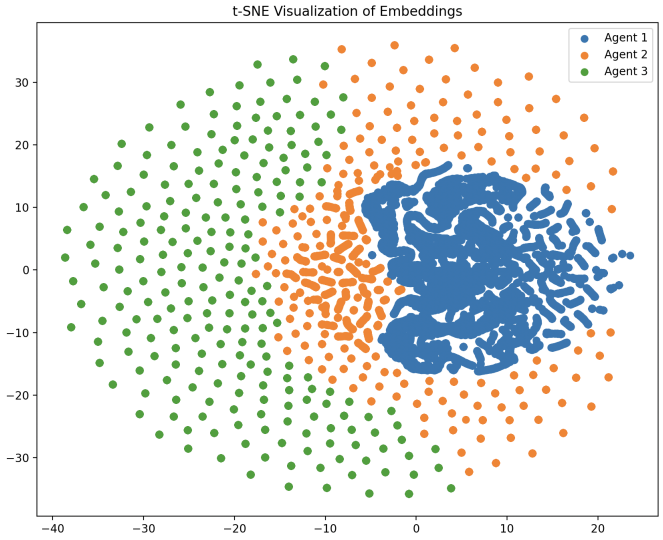

Figure 5.9: t-SNE visualization of the embedding representations for z (LBF: $8 \times 8, 3p, 2f$)

group of points might suggest that the embeddings for that agent are relatively consistent, while a more scattered group could indicate greater variability.

4. Local vs. Global Relationships: t-SNE tends to preserve local relationships better than global ones. This means that embeddings that are close to each other in the t-SNE plot are likely to be similar, but the overall structure of the plot might not accurately reflect the true distances between all points in the high-dimensional space.

5. Outliers: Outlying points in the t-SNE plot may represent unusual or distinct embeddings that stand out from the rest of the data. These outliers could potentially hold valuable information about unique situations or scenarios.

6. Patterns and Trends: By observing the overall arrangement of points and their colors, you might identify trends, patterns, or correlations that were not immediately apparent in the original data.

7. Model Evaluation: The t-SNE visualization can also be used as an evaluation tool. For example, if your VAE model is successful in producing embeddings that accurately capture differences between agents, you should see clear separation between the agent clusters.

It is important to note that while t-SNE is a powerful visualization tool, it is not a definitive solution. Different t-SNE runs or parameter settings can lead to slightly different visualizations.

The t-SNE visualization of the embedding representations for z of all agents in the LBF $8 \times 8, 3p, 2f$ task, at the 20th timestep of an episode, is illustrated in Figure 5.9. We observe that each agent learns its own unique embeddings in order to model the behavior

of the other and the embeddings seem to not overlap but to belong to identifiable clusters. Moreover, they occupy a unique area in the 2D dimensional greed which further validates that the learnt embedding representations are meaningful, in the sense that they allow each agent to infer useful information about the observations of the others and thus they could enhance their policies appropriately with such information.

Chapter 6

Conclusion and Future Work

This thesis embarked on a comprehensive exploration of various Multi-Agent Reinforcement Learning (MARL) approaches and algorithms for tackling cooperative benchmarking gaming environments. The investigation focused on examining state-of-the-art MARL algorithms within the context of solving cooperative team games, with a specific emphasis on two key aspects of the learning process: (a) the capacity of MARL algorithms to facilitate representation learning, enabling the creation of meaningful vector interpretations of the underlying environment. These interpretations promote effective coordination among agents in their actions, and (b) the agents' aptitude for perceiving and formulating significant agent goals, contributing to the collective endeavor of enhancing overall performance in the given task.

Remarkably, we introduced the innovative framework of Count-based Agent Modelling (CAM), which is applicable to any MARL algorithm operating under the Centralized-Training-Decentralized-Execution (CTDE) paradigm. CAM amalgamates principles from agent modelling with intrinsic exploration, pursuing two fundamental objectives. Firstly, CAM utilizes meaningful, learnable representations to guide agent cooperation and enhance their policies. Secondly, these representations are harnessed for novel intrinsic exploration, especially valuable in addressing challenges posed by tasks with sparse rewards. Extensive experimental analyses were conducted using two prevalent MARL benchmarking testbeds, namely the Multi-Agent Particle Environment (MPE) and the Level-based Foraging (LBF) multi-agent gaming environments. The outcomes demonstrate that CAM surpasses state-of-the-art MARL algorithms in terms of total reward achieved across these tasks, establishing its effectiveness in enhancing cooperative behavior and performance. Last but not least, this thesis offers an additional confirmation of the actor-critic algorithm's competence within the CTDE framework when addressing MARL tasks. This reaffirms the algorithm's suitability and efficiency in navigating cooperative multi-agent scenarios.

In conclusion, this thesis contributes to the advancement of the MARL field by proposing the CAM framework, elucidating its benefits through rigorous experimentation, and shedding light on the efficacy of the actor-critic algorithm in cooperative environments.

Through this exploration, it underscores the significance of meaningful representation learning and goal formulation in enhancing the cooperation and performance of agents in multi-agent settings.

In future work, we aim to compare CAM with communication MARL algorithms, as well as further MARL algorithms on the evaluated testbeds. Moreover, we plan to run experiments on other testbeds, such as SMAC and Google Research Football. Last but not least, we would evaluate the self-supervised learnable weights, that we introduced in this thesis, on communication methods aiming to make the agent communication more efficient as the number of agents increases.

Appendix A

Implementation Dependencies

The project was implemented in Python 3.9. For scientific computation the NumPy version 1.23.4 library was utilized, while the deep neural networks were implemented in PyTorch 1.12.1. Lastly Matplotlib was used for the generation of all plots in this thesis. A full list of all such dependencies is shown in Table 6.1. The implementation of our proposed methodology is publicly available on GitHub. Finally the training of the algorithms was performed using an 11th Gen Intel(R) Core(TM) i9-11900 and an NVIDIA RTX 3080 Ti GPU with 12 GB memory.

Name	Description	Version	Source
Python	programming language	3.9	https://www.python.org/
NumPy	scientific computation library	1.23.4	https://numpy.org/
PyTorch	deep learning platform	1.12.1	https://pytorch.org/
Matplotlib	2d plotting library	3.6.0	https://matplotlib.org/
OpenAI Gym	RL environment toolkit	0.21.0	OpenAI Gym
EPyMARL	MARL environment	/	EPyMARL

Table 6.1: Code Dependencies for Implementation

Appendix B

Evaluation Parameterisation

In Table 6.2 we provide a full list of hyperparameters used for training and evaluating our proposed algorithm as well as the respective baselines.

Name	Description	Value
$N_{timesteps}$	total number of training timesteps	10,000,000
$N_{max-episode-length}$	maximum length of an episode	25
N_{tup}	number of environment steps before optimisation	100
N_D	replay buffer capacity (for VAE training)	50,000
N_{bs}	replay buffer batch size	32
N_{ED}	update Encoder-Decoder parameters every up_vae time steps	2000
N_w	update filter parameters every up_filter time steps	2000
N_{wtup}	update target filter parameters every tar_up_filter time steps	50,000
lr_π	learning rate for RL algorithm	0.0005
lr_w	learning rate for weight calculation	0.00005
lr_{ED}	learning rate for VAE	0.0005
hidden_dim	hidden dimensionality of Actor and Critic NN	128
latent_dim	latent dimensionality of Encoder-Decoder	32
γ	discount factor	0.99
λ_{KL}	lambda coefficient for L_{KL}	0.1
λ_{norm}	lambda coefficient for L_{norm}	0.0001
λ_{intr}	intrinsic reward coefficient	0.1
N_{eval_freq}	number of time steps before evaluation	2500
N_{eval_eps}	number of episodes for each evaluation	5

Table 6.2: Implementation Details of CAM

Bibliography

[1] E. Adamopoulou and L. Moussiades. Chatbots: History, technology, and applications. *Machine Learning with Applications*, 2:100006, 2020.

[2] A. Agarwal, N. Jiang, S. M. Kakade, and W. Sun. Reinforcement learning: Theory and algorithms. *CS Dept., UW Seattle, Seattle, WA, USA, Tech. Rep*, 32, 2019.

[3] S. V. Albrecht, F. Christianos, and L. Schäfer. *Multi-Agent Reinforcement Learning: Foundations and Modern Approaches*. MIT Press, 2023.

[4] S. V. Albrecht and P. Stone. Autonomous agents modelling other agents: A comprehensive survey and open problems. *Artificial Intelligence*, 258:66–95, 2018.

[5] E. Axiotis, A. Kontogiannis, E. Kalpoutzakis, and G. Giannakopoulos. A personalized machine-learning-enabled method for efficient research in ethnopharmacology. the case of the southern balkans and the coastal zone of asia minor. *Applied Sciences*, 11(13):5826, 2021.

[6] C. Badue, R. Guidolini, R. V. Carneiro, P. Azevedo, V. B. Cardoso, A. Forechi, L. Jesus, R. Berriel, T. M. Paixao, F. Mutz, et al. Self-driving cars: A survey. *Expert Systems with Applications*, 165:113816, 2021.

[7] M. Bellemare, S. Srinivasan, G. Ostrovski, T. Schaul, D. Saxton, and R. Munos. Unifying count-based exploration and intrinsic motivation. *Advances in neural information processing systems*, 29, 2016.

[8] R. Bellman. A markovian decision process. *Journal of mathematics and mechanics*, pages 679–684, 1957.

[9] C. Berner, G. Brockman, B. Chan, V. Cheung, P. Dębiak, C. Dennison, D. Farhi, Q. Fischer, S. Hashme, C. Hesse, et al. Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*, 2019.

[10] D. S. Bernstein, R. Givan, N. Immerman, and S. Zilberstein. The complexity of decentralized control of markov decision processes. *Mathematics of operations research*, 27(4):819–840, 2002.

[11] W. Böhmer, T. Rashid, and S. Whiteson. Exploration with unreliable intrinsic reward in multi-agent reinforcement learning. *arXiv preprint arXiv:1906.02138*, 2019.

[12] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, et al. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016.

[13] M. Bowling, N. Burch, M. Johanson, and O. Tammelin. Heads-up limit hold'em poker is solved. *Science*, 347(6218):145–149, 2015.

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

[15] L. Busoniu, R. Babuska, and B. De Schutter. A comprehensive survey of multiagent reinforcement learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 38(2):156–172, 2008.

[16] J. Co-Reyes, Y. Liu, A. Gupta, B. Eysenbach, P. Abbeel, and S. Levine. Self-consistent trajectory autoencoder: Hierarchical reinforcement learning with trajectory embeddings. In *International conference on machine learning*, pages 1009–1018. PMLR, 2018.

[17] F. L. Da Silva and A. H. R. Costa. A survey on transfer learning for multiagent reinforcement learning systems. *Journal of Artificial Intelligence Research*, 64:645–703, 2019.

[18] Z. Ding and H. Dong. Challenges of reinforcement learning. *Deep Reinforcement Learning: Fundamentals, Research and Applications*, pages 249–272, 2020.

[19] R. Everett and S. Roberts. Learning against non-stationary agents with opponent modelling and deep reinforcement learning. In *2018 AAAI spring symposium series*, 2018.

[20] Y. Farag, C. O. Brand, J. Amidei, P. Piwek, T. Stafford, S. Stoyanchev, and A. Vlachos. Opening up minds with argumentative dialogues. *arXiv preprint arXiv:2301.06400*, 2023.

[21] J. Foerster, I. A. Assael, N. De Freitas, and S. Whiteson. Learning to communicate with deep multi-agent reinforcement learning. *Advances in neural information processing systems*, 29, 2016.

[22] J. Foerster, G. Farquhar, T. Afouras, N. Nardelli, and S. Whiteson. Counterfactual multi-agent policy gradients. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

[23] J. N. Foerster, R. Y. Chen, M. Al-Shedivat, S. Whiteson, P. Abbeel, and I. Mordatch. Learning with opponent-learning awareness. *arXiv preprint arXiv:1709.04326*, 2017.

[24] A. Grover, M. Al-Shedivat, J. K. Gupta, Y. Burda, and H. Edwards. Evaluating generalization in multiagent systems using agent-interaction graphs. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, pages 1944–1946, 2018.

[25] C. Guan, F. Chen, L. Yuan, C. Wang, H. Yin, Z. Zhang, and Y. Yu. Efficient multi-agent communication via self-supervised information aggregation. *Advances in Neural Information Processing Systems*, 35:1020–1033, 2022.

[26] M. Hausknecht and P. Stone. Deep recurrent q-learning for partially observable mdps. In *2015 aaai fall symposium series*, 2015.

[27] K. Hausman, J. T. Springenberg, Z. Wang, N. Heess, and M. Riedmiller. Learning an embedding space for transferable robot skills. In *International Conference on Learning Representations*, 2018.

[28] H. He, J. Boyd-Graber, K. Kwok, and H. Daumé III. Opponent modeling in deep reinforcement learning. In *International conference on machine learning*, pages 1804–1813. PMLR, 2016.

[29] J. Heinrich and D. Silver. Deep reinforcement learning from self-play in imperfect-information games. *arXiv preprint arXiv:1603.01121*, 2016.

[30] Z.-W. Hong, S.-Y. Su, T.-Y. Shann, Y.-H. Chang, and C.-Y. Lee. A deep policy inference q-network for multi-agent systems. *arXiv preprint arXiv:1712.07893*, 2017.

[31] R. A. Howard. Dynamic programming and markov processes. 1960.

[32] Y. Huang, L. Huang, and Q. Zhu. Reinforcement learning for feedback-enabled cyber resilience. *Annual reviews in control*, 53:273–295, 2022.

[33] S. Iqbal, C. A. S. de Witt, B. Peng, W. Böhmer, S. Whiteson, and F. Sha. Ai-qmix: Attention and imagination for dynamic multi-agent reinforcement learning. *arXiv preprint arXiv:2006.04222*, 2020.

[34] J. Jeon, W. Kim, W. Jung, and Y. Sung. Maser: Multi-agent reinforcement learning with subgoals generated from experience replay buffer. In *International Conference on Machine Learning*, pages 10041–10052. PMLR, 2022.

[35] J. Jiang and Z. Lu. The emergence of individuality. In *International Conference on Machine Learning*, pages 4992–5001. PMLR, 2021.

[36] S. M. Kakade. *On the sample complexity of reinforcement learning.* University of London, University College London (United Kingdom), 2003.

[37] D. P. Kingma and M. Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[38] A. Kontogiannis, D. Kelesis, V. Pollatos, G. Paliouras, and G. Giannakopoulos. Tree-based focused web crawling with reinforcement learning. *arXiv preprint arXiv:2112.07620*, 2021.

[39] A. Kontogiannis and G. Vouros. Xdqn: Inherently interpretable dqn through mimicking. *arXiv preprint arXiv:2301.03043*, 2023.

[40] L. Kraemer and B. Banerjee. Multi-agent reinforcement learning as a rehearsal for decentralized planning. *Neurocomputing*, 190:82–94, 2016.

[41] T. Kravaris, K. Lentzos, G. Santipantakis, G. A. Vouros, G. Andrienko, N. Andrienko, I. Crook, J. M. C. Garcia, and E. I. Martinez. Explaining deep reinforcement learning decisions in complex multiagent settings: towards enabling automation in air traffic flow management. *Applied Intelligence*, 53(4):4063–4098, 2023.

[42] C. Li, T. Wang, C. Wu, Q. Zhao, J. Yang, and C. Zhang. Celebrating diversity in shared multi-agent reinforcement learning. *Advances in Neural Information Processing Systems*, 34:3991–4002, 2021.

[43] S. Li, Y. Wu, X. Cui, H. Dong, F. Fang, and S. Russell. Robust multi-agent reinforcement learning via minimax deep deterministic policy gradient. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pages 4213–4220, 2019.

[44] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[45] R. Lowe, Y. I. Wu, A. Tamar, J. Harb, O. Pieter Abbeel, and I. Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. *Advances in neural information processing systems*, 30, 2017.

[46] A. Mahajan, T. Rashid, M. Samvelyan, and S. Whiteson. Maven: Multi-agent variational exploration. *Advances in neural information processing systems*, 32, 2019.

[47] A. Mahajan, T. Rashid, M. Samvelyan, and S. Whiteson. Maven: Multi-agent variational exploration. *Advances in neural information processing systems*, 32, 2019.

[48] A. Mandalios, A. Chortaras, G. Stamou, and M. Vazirgiannis. Enriching graph representations of text: Application to medical text classification. In *International Conference on Complex Networks and Their Applications*, pages 92–103. Springer, 2020.

[49] H. Mao, Z. Gong, Y. Ni, and Z. Xiao. Accnet: Actor-coordinator-critic net for" learning-to-communicate" with deep multi-agent reinforcement learning. *arXiv preprint arXiv:1706.03235*, 2017.

[50] M. Minsky. Steps toward artificial intelligence. *Proceedings of the IRE*, 49(1):8–30, 1961.

[51] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[52] Y. Nevmyvaka, Y. Feng, and M. Kearns. Reinforcement learning for optimized trade execution. In *Proceedings of the 23rd international conference on Machine learning*, pages 673–680, 2006.

[53] A. Y. Ng, D. Harada, and S. Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *Icml*, volume 99, pages 278–287. Citeseer, 1999.

[54] G. Papoudakis, F. Christianos, and S. Albrecht. Agent modelling under partial observability for deep reinforcement learning. *Advances in Neural Information Processing Systems*, 34:19210–19222, 2021.

[55] G. Papoudakis, F. Christianos, A. Rahman, and S. V. Albrecht. Dealing with non-stationarity in multi-agent deep reinforcement learning. *arXiv preprint arXiv:1906.04737*, 2019.

[56] G. Papoudakis, F. Christianos, L. Schäfer, and S. V. Albrecht. Benchmarking multi-agent deep reinforcement learning algorithms in cooperative tasks. *arXiv preprint arXiv:2006.07869*, 2020.

[57] T. Rashid, G. Farquhar, B. Peng, and S. Whiteson. Weighted qmix: Expanding monotonic value function factorisation for deep multi-agent reinforcement learning. *Advances in neural information processing systems*, 33:10199–10210, 2020.

[58] T. Rashid, M. Samvelyan, C. S. De Witt, G. Farquhar, J. Foerster, and S. Whiteson. Monotonic value function factorisation for deep multi-agent reinforcement learning. *The Journal of Machine Learning Research*, 21(1):7234–7284, 2020.

[59] J. Schrittwieser. Antonoglou i hubert t simonyan k sifre l schmitt s guez a lockhart e hassabis d graepel t et al. *Mastering atari, go, chess and shogi by planning with a learned model Nature*, 588(7839):604, 2020.

[60] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

[61] K. Son, D. Kim, W. J. Kang, D. E. Hostallero, and Y. Yi. Qtran: Learning to factorize with transformation for cooperative multi-agent reinforcement learning. In *International conference on machine learning*, pages 5887–5896. PMLR, 2019.

[62] G. Stoilos, G. Stamou, J. Z. Pan, V. Tzouvaras, and I. Horrocks. Reasoning with very expressive fuzzy description logics. *Journal of Artificial Intelligence Research*, 30:273–320, 2007.

[63] P. Sunehag, G. Lever, A. Gruslys, W. M. Czarnecki, V. Zambaldi, M. Jaderberg, M. Lanctot, N. Sonnerat, J. Z. Leibo, K. Tuyls, et al. Value-decomposition networks for cooperative multi-agent learning. *arXiv preprint arXiv:1706.05296*, 2017.

[64] R. S. Sutton. Sutton & barto book: Reinforcement learning: An introduction. In *A Bradford Book*. MIT Press Cambridge, MA, 1998.

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

[66] H. Tang, R. Houthooft, D. Foote, A. Stooke, O. Xi Chen, Y. Duan, J. Schulman, F. DeTurck, and P. Abbeel. # exploration: A study of count-based exploration for deep reinforcement learning. *Advances in neural information processing systems*, 30, 2017.

[67] L. Van der Maaten and G. Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(11), 2008.

[68] H. Van Hasselt, A. Guez, and D. Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30, 2016.

[69] R. E. Wang, M. Everett, and J. P. How. R-maddpg for partially observable environments and limited communication. *arXiv preprint arXiv:2002.06684*, 2020.

[70] T. Wang, H. Dong, V. Lesser, and C. Zhang. Roma: Multi-agent reinforcement learning with emergent roles. In *International Conference on Machine Learning*, pages 9876–9886. PMLR, 2020.

[71] T. Wang, J. Wang, Y. Wu, and C. Zhang. Influence-based multi-agent exploration. In *International Conference on Learning Representations*, 2019.

[72] D. H. Wolpert and K. Tumer. Optimal payoff functions for members of collectives. *Advances in Complex Systems*, 4(02n03):265–279, 2001.

[73] A. Wong, T. Bäck, A. V. Kononova, and A. Plaat. Deep multiagent reinforcement learning: Challenges and directions. *Artificial Intelligence Review*, 56(6):5023–5056, 2023.

[74] Y. Yang, G. Chen, W. Wang, X. Hao, J. Hao, and P.-A. Heng. Transformer-based working memory for multiagent reinforcement learning with action parsing. *Advances in Neural Information Processing Systems*, 35:34874–34886, 2022.

[75] Y. Yang, R. Luo, M. Li, M. Zhou, W. Zhang, and J. Wang. Mean field multi-agent reinforcement learning. In *International conference on machine learning*, pages 5571–5580. PMLR, 2018.

[76] Y. Yu. Towards sample efficient reinforcement learning. In *IJCAI*, pages 5739–5743, 2018.

[77] A. Zhang, H. Satija, and J. Pineau. Decoupling dynamics and reward for transfer learning, 2018.

[78] S. Zhang, J. Cao, L. Yuan, Y. Yu, and D.-C. Zhan. Self-motivated multi-agent exploration. In *Proceedings of the 2023 International Conference on Autonomous Agents and Multiagent Systems*, pages 476–484, 2023.

[79] Y. Zhu, R. Mottaghi, E. Kolve, J. J. Lim, A. Gupta, L. Fei-Fei, and A. Farhadi. Target-driven visual navigation in indoor scenes using deep reinforcement learning. In *2017 IEEE international conference on robotics and automation (ICRA)*, pages 3357–3364. IEEE, 2017.

[80] L. Zintgraf, S. Devlin, K. Ciosek, S. Whiteson, and K. Hofmann. Deep interactive bayesian reinforcement learning via meta-learning. *arXiv preprint arXiv:2101.03864*, 2021.

[81] L. Zintgraf, K. Shiarlis, M. Igl, S. Schulze, Y. Gal, K. Hofmann, and S. Whiteson. Varibad: A very good method for bayes-adaptive deep rl via meta-learning. *arXiv preprint arXiv:1910.08348*, 2019.