

**ANKARA ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ**

DOKTORA TEZİ

**TAKVİYE ÖĞRENME KULLANARAK MERKEZİ OLMAYAN TOPLU
ROBOT NAVİGASYONU**

Tekin ALPTÜRK

BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI

**ANKARA
2022**

Her hakkı saklıdır

ÖZET

Doktora Tezi

TAKVİYE ÖĞRENME KULLANARAK MERKEZİ OLMAYAN TOPLU ROBOT NAVİGASYONU

Tekin ALPTÜRK

Ankara Üniversitesi
Fen bilimleri Enstitüsü
Bilgisayar Mühendisliği Anabilim Dalı

Danışman: Doç. Dr. Mehmet Serdar GÜZEL

Bu tez çalışmasında, kısmen dağınık ortamlarda merkezi olmayan çok ajanlı sistemlerde gezinme sorunu ele alınmaktadır. İlgili problemi çözmek için özgün bir makine öğrenimi tabanlı yaklaşım bu çalışma kapsamında önerilmektedir. Bu yaklaşım temelinde, yeni Q-Öğrenme tabanlı sağlam ve esnek bir model önerilmiştir. Bu yöntem sürekli uzay problemini ele almaktadır. Takviyeli Öğrenme algoritmalarında olduğu gibi, Q-Öğrenme bir çevre modeli gerektirmez. Ayrıca, Q-Öğrenme, basit tasarım ve hızlı olma avantajlarına sahiptir. Bununla birlikte, Q-Öğrenmenin bir dezavantajı, çok büyük miktarda belleğe ihtiyaç duyması ve durum alanına eklenen her yeni özellik ile bu bellek ihtiyacının katlanarak büyümesidir. Bu araştırmada, kısmen dağınık ortamlarda sürekli bir alan sorununu çözmek için ajan düzeyinde merkezi olmayan çarpışmadan kaçınma için düşük maliyetli bir model tanıtıyoruz, ardından boyutunu önemli ölçüde (%70) azaltmak için örtüşmeyen Q-Öğrenme özelliklerini birleştirmek için bir yöntem tanıtıyoruz. Ek olarak, bu çalışmada denetleyici tarafından kullanılan duyusal verileri en aza indirmek için başka bir yöntem önerilmiştir. Bu yöntemlerin bir birleşimi, en az 18 adet robotla, düşük bellek maliyetiyle sürü gezinimin üstesinden gelebilir. Bu yöntemler aynı zamanda derin q-öğrenme mimarilerine uyarlanabilir ve performanslarını artırmak ve aynı zamanda öğrenme süresini azaltmak için kullanılabilir. Deneyler, önerilen yöntemin karmaşık senaryolarda çok ajanlı sistemler için de yüksek derecede doğruluk sağladığını ortaya koymaktadır.

Temmuz 2022, 82 Sayfa

Anahtar Kelimeler: Makine Öğrenmesi, Takviyeli Öğrenme, Uyarlamalı Algoritma, Sürekli Uzay problemi, Çok ajanlı sistemler, Q-Öğrenme

ABSTRACT

Ph.D. Thesis

DECENTRALIZED SWARM NAVIGATION USING REINFORCEMENT LEARNING

Tekin ALPTÜRK

Ankara University
Graduate School of Natural and Applied Sciences
Department of Computer Engineering

Supervisor: Assoc. Prof. Mehmet Serdar GÜZEL

This thesis addresses the problem of navigating decentralized multi-agent systems in partially cluttered environments and proposes a new machine-learning-based approach to solve it. On the basis of this approach, a new robust and flexible Q-learning-based model is proposed to handle a continuous space problem. As in reinforcement learning (RL) algorithms, Q-learning does not require a model of the environment. Additionally, Q-Learning (QL) has the advantages of being fast and easy to design. However, one disadvantage of QL is that it needs a massive amount of memory, and it grows exponentially with each extra feature introduced to the state space. In this research, we introduce an agent-level decentralized collision avoidance low-cost model for solving a continuous space problem in partially cluttered environments, followed by introducing a method to merge non-overlapping QL features in order to reduce its size significantly by about 70% and make it possible to solve more complicated scenarios with the same memory size. Additionally, another method is proposed for minimizing the sensory data that is used by the controller. A combination of these methods is able to handle swarm navigation low memory cost with at least 18 number of robots. These methods can also be adapted for deep q-learning architectures so as to increase their approximation performance and also decrease their learning time process. Experiments reveal that the proposed method also achieves a high degree of accuracy for multi-agent systems in complex scenarios.

July 2022, 82 pages

Key Words: Artificial Intelligence, Reinforcement learning, Machine learning, Adaptive algorithm, Continuous Space Problem, Multi-Agent Systems, Q-Learning.

ÖNSÖZ VE TEŞEKKÜR

Eğitimim süresince bu tez çalışmasının hazırlanması ve şekillenmesi birçok kişinin yardım ve destekleriyle gerçekleşmiştir. Öncelikle tez çalışmasının her aşamasında bana destek olup yol gösteren, mesleki bilgilerini ve deneyimlerini öğrencilerinden esirgemeyip önerileri ile katkıda bulunan değerli tez danışmanım Sayın Doç. Dr. Mehmet Serdar GÜZEL (Ankara Üniversitesi Bilgisayar Mühendisliği Anabilim Dalı) hocama, her zaman tecrübeleri ile bana yol gösteren Prof. Dr. İman ASKERBEYLİ (Ankara Üniversitesi Bilgisayar Mühendisliği Anabilim Dalı) hocama teşekkürlerimi sunarım.

Ayrıca, beni her konuda destekleyen, yetiştirip bu günlere gelmemi sağlayan, üzerimdeki emeklerini hiçbir zaman ödeyemeyeceğim kıymetli Annem, babam ve kardeşlerime sonsuz teşekkürlerimi sunarım.

Tekin ALPTÜRK
Ankara, Temmuz 2022

İÇİNDEKİLER

TEZ ONAYI

ETİK.....	i
ÖZET.....	ii
ABSTRACT	iii
ÖNSÖZ VE TEŞEKKÜR.....	iv
KISALTMALAR DİZİNİ	vii
ŞEKİLLER DİZİNİ	viii
ÇİZELGELER DİZİNİ	ix
1. GİRİŞ	1
1.1 Çalışmanın Katkısı.....	3
1.2 Benzer Çalışmalar	6
2. LİTERATÜR.....	9
2.1 Q-Öğrenmeyi İyileştiren Çalışmalar	10
2.2 Q-Öğrenmeyi kullanan çalışmalar	11
3. MATERIAL VE YÖNTEM	16
3.1 Takviyeli Öğrenme.....	16
3.1.1 Takviyeli öğrenme temelleri.....	17
3.1.2 Ödül fonksiyonu	18
3.1.3 Takviyeli öğrenmede “Durumlar”	19
3.1.4 Tic-Tac-Toe problemi	20
3.1.5 Araba-direk (CartPole) problemi.....	21
3.1.6 Araba-direk (CartPole) problemi sürekli ortamda	23
3.1.7 Takviyeli öğrenmede ajan	24
3.1.8 Markov karar süreci.....	27
3.1.9 Q-Öğrenme	29
3.1.10 Çok ajanlı Q-öğrenme	31
3.1.11 Derin öğrenmeye giriş.....	33
3.1.12 Derin öğrenmede mimari	38
3.1.13 Evrişimli sinir ağları—Vizyon için derin öğrenme	42
3.2 Problem	46
3.2.1 Senaryo tanımı.....	46
3.2.2 Normal durum uzaylı denetleyicisi (NDUD)	49

3.3 Yöntem	51
3.3.1 SDUD için model özelliklerinin tanımlanması	52
3.3.2 Sıkıştırılmış durum uzaylı denetleyici (SDUD)	54
3.3.3 Genelleştirilmiş SDUD	55
3.3.4 Bir bakışta genel akış	57
4. DENEYLER	59
4.1 SDUD ve NDUD Karşılaştırılması	59
4.2 SDUD sonuçları ve öğrenme grafiği ve karmaşıklık analizi	62
4.3 Gezinim (Navigasyon) Sonuçları	63
4.4 NDUD ve SDUD Öğrenme Grafikleri	67
4.5 Genelleştirilmiş SDUD	68
5. SONUÇ VE TARTIŞMA.....	72
5.1 Gelecek Çalışmaları	73
KAYNAKLAR	75
ÖZGEÇMİŞ.....	81

KISALTMALAR DİZİNİ

X	sonlu sayıda olası çevre durumu
U	sonlu ajan eylemleri kümesi
f	Durumlar arası geçiş için olasılık fonksiyonu
ρ	ödül fonksiyonu
R	beklenen indirimli getiri
h	politika
α	öğrenme oranı
γ	indirim faktörü
τ	keşif takası
Q	kalite fonksiyonu
d_g	hedefe uzaklık
a_g	hedefin göreceli açısı
a_r	aktif robotun açısı
c_r	çarpışma özelliği
g_r	hedef özelliği
d_{r_i}	i 'inci robota olan uzaklık
a_{r_i}	i 'nci robotun göreceli açısı
t	zaman adımı sayısı
r_g	gol ödülü
r_s	adım ödülü
r_c	çarpışma ödülü
s_c	adım sayısı

ŞEKİLLER DİZİNİ

Şekil 1.1 Sürü robotları için önerilen merkezi olmayan gezinim stratejisinin akış diyagramı (Güzel ve diğerleri, 2019).....	3
Şekil 2.1 Q-Öğrenme	9
Şekil 3.1 Takviyeli Öğrenmenin Şeması.....	17
Şekil 3.2 Tic-Toc-Toe oyunu	21
Şekil 3.3 Araba-direk problem örneği.....	22
Şekil 3.4 Biyolojik Nöron (kaynak Wikipedia)	33
Şekil 3.5 Yapay nöronun işleyişi	34
Şekil 3.6 Derin öğrenme ağı.....	36
Şekil 3.7 Bir CNN'nin grafiksel gösterimi	42
Şekil 3.8 Gezinme	46
Şekil 3.9 Test senaryoları: (a) dağınık U-şekilli engeller içeren bir senaryo; (b) bir binanın iç mekân haritasına benzeyen bir senaryo. (c) bir oda ve dağınık engeller içeren bir senaryo.	48
Şekil 3.10 Özellik ayırıklaştırma akış şeması	52
Şekil 3.11 Aktif robot ve etrafındaki robotların önceliği	56
Şekil 3.12 Önerilen yöntemin akışı bir bakışta	58
Şekil 4.1 Üç robotla SDUD kullanımının gezinim sonuçları: (a) senaryo 1; (b) senaryo 2; (c) senaryo 3.....	65
Şekil 4.2 Öğrenme grafiği: (a) senaryo 2 için NDUD öğrenme grafiği; (b) senaryo 2 için SDUD öğrenme grafiği.	67
Şekil 4.3 GSDUD kullanarak dokuz robot gezinimi sonuçları	70

ÇİZELGELER DİZİNİ

Çizelge 4.1 Q-Tablosu eleman sayısı ve bellekteki gerçek boyutla ilgili çok ajanlı gezinim problemleri için farklı ayırıştırımlarla NDUD ve SDUD arasındaki boyut	61
Çizelge 4.2 Farklı sayıda robotla yapılan deneylerin performans sonuçları	71



1. GİRİŞ

Robotik araştırmacılar, yoğun bir şekilde sürü zekâsı olarak adlandırılan çalışmalarında çoklu robotlu sistemleri kullanırlar. Bunu yapmak için grup düzeyinde etkileyici ve karmaşık eylem getiren hayvan kümelenmesinin biyolojik çalışmalarından ilham alıyorlar (Bode vd. 2011, Flierl vd.1999), çok robotlu sistemler, ajanlar ve çevre arasındaki yerel bireysel etkileşimlerden arzu edilen bir kolektif davranışın ortaya çıkacağı şekilde tasarlanabilir (Bahgeçi ve Sahin 2005). Bu çalışma alanı, sınırlı zamana sahip karmaşık görevleri yapmak için robotları planlamak ve koordine etmek üzere sürü zekâsı disiplini ve kolektif robotiği birleştirir. Başka bir deyişle, çok ajanlı sistemler, bazı yönlerden sürü zekâsı alanına benzeyen özerk ajanların kümeleridir (Brambilla vd. 2013, Chamanbaz vd. 2017). Çoklu robotlu sistemlerin diğer önemli avantajları, ölçeklenebilirlik, sağlamlık ve esnekliktir ki (Bahgeçi ve Sahin 2005, Camazine vd. 2003, Costa vd. 2016, Mohan ve Ponnambalam 2009) tartışılmıştır. Sağlamlık, bir cihazın kısmi arıza veya diğer anormal koşullar karşısında işlevselliğini sürdürme derecesidir (Senanayake vd. 2016, Tan ve Zheng 2013). Sonuç olarak, Sürü robotiğinde sağlamlık, robot sürüsünün bazı bireysel arızalar veya ortamın öngörülemez olması durumunda bile çalışabilme kapasitesini ifade eder. Ayrıca gruptaki robot sayısı değiştirilebilir olduğu için ölçeklenebilirler. Sürü robotiği, değişen çalışma koşullarına hızla uyum sağlayan ve çok çeşitli ortam ve koşullarda çalışabilen sistemlerin oluşturulmasını kolaylaştıran esnek sistemler oluşturmak için kullanılabilir. Birçok sayıda uygulama farklı alanlarda çoklu robot sistemleri kullanmıştır. Belirli bir geometrik desenle merkezi olmayan bir şekilde toplu hareket (Matignon vd. 2007, Savkin vd. 2016)'de incelenmiştir. Kapsama problemini çözmek için mobil otonom robotik algılayıcılar (Cheng ve Savkin 2011)'te tartışılmaktadır.

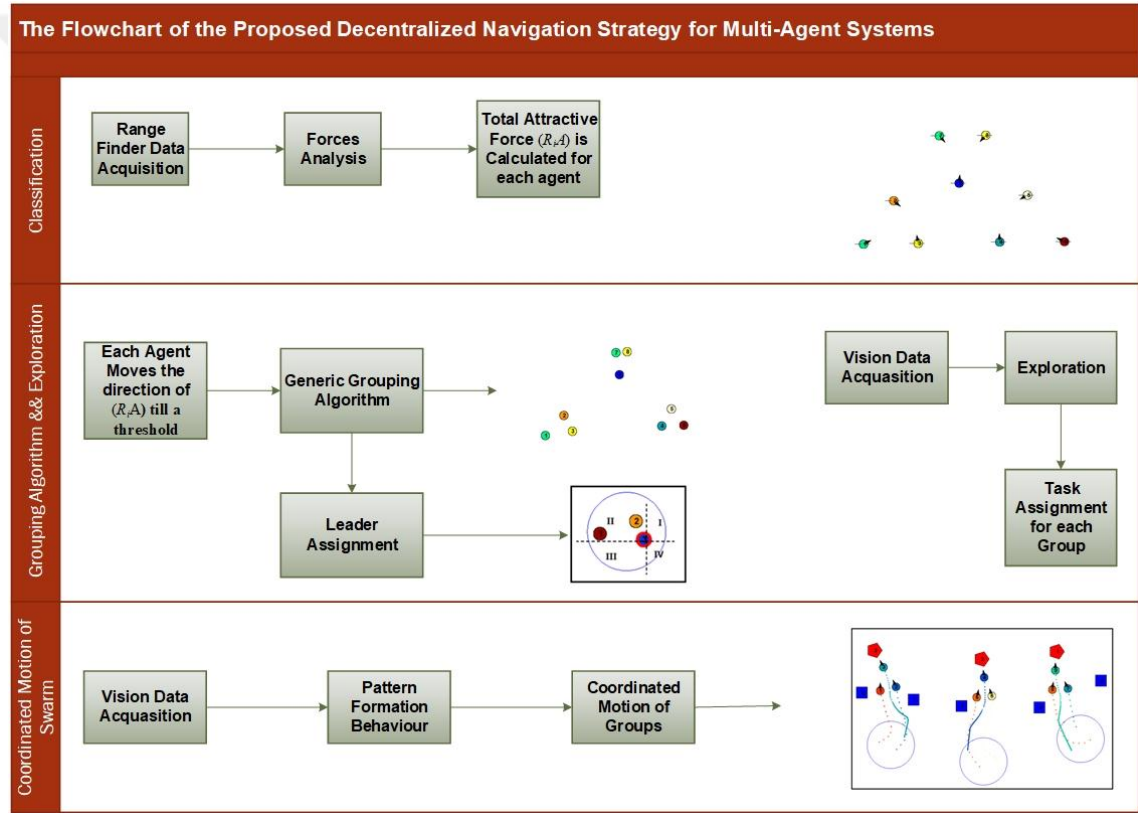
Sürü robotiği, harici altyapıdan veya merkezi kontrolden bağımsız olarak çalışan robot grupları oluşturma çalışmasıdır. Bir sürüdeki robotların toplu davranışı, robotlar arasındaki ve ayrıca robotlar ve içinde çalıştıkları ortam arasındaki yerel etkileşimlere dayanır. Sürü zekâsı kavramları, robot sürülerinin oluşturulmasını rehberlik eder ve yönlendirir. Bu ilkeler, hataya dayanıklı, ölçeklenebilir ve çok yönlü sistemlerin geliştirilmesini kolaylaştırır. Birden fazla görevin aynı anda tamamlanması

gerektiğinde, yüksek yedeklilik ve tek bir arıza noktasının olmaması gerekir. Robotları merkezi bir şekilde çalıştırmak için gerekli altyapıyı kurmak teorik olarak mümkün değildir; literatürde, sürü robotiği bu durumda umut verici bir çözüm olarak ele alınmaktadır (Güzel ve Kayakökü 2017). Mayın temizleme, arama ve kurtarma, gezegen veya su altı keşfi ve gözetim, sürü robotiğinden yararlanabilecek görevlerdir (Al 2014).

Q-Learning (Christopher vd. 1989), çevreyi doğrudan modellemeden politikaları çıkaran, modelsiz bir takviyeli öğrenme algoritmasıdır. Diğer bir deyişle sistem modelinin bilinmediği durumlarda kullanılan uyarlanabilir, adaptif, bir kontrol yöntemidir (Bertsekas 2005, Richard vd. 2018). Bu nedenle model tabanlı algoritmalara göre daha basit ve daha esnek bir algoritmadır (Jin vd. 2018), ayrıca, en uygun Q fonksiyonuna yakınsadığı kanıtlanmıştır (Borkar ve Meyn 2000, Jaakkola vd. 1994, Christopher vd.1992a). Öte yandan deneysel araştırmalar, öğrenmenin daha fazla zaman ve inceleme örnekleri gerektirdiğini ortaya koymaktadır (Deisenroth ve Rasmussen 2011, Schulman vd. 2015).

Takviyeli öğrenme ajanları, karmaşık ortamlarıyla etkileşime girerek öğrenirler (Kaelbling vd. 1996, Richard vd.1998, Weiss 1999). Ajan, her zaman biriminde ortamın durumunu gözlemler ve ortamın yeni bir duruma geçmesine neden olan bir eylemde bulunur. Durumdaki her değişimin kalitesi, bir skaler ödül sinyali ile değerlendirilir ve kümülatif ödülü maksimize etmek, ajanın etkileşim süreci üzerindeki hedefidir. Takviyeli öğrenme geribildirim (ödül), aracıya tam olarak yapması gereken uygun eylemlerin verildiği denetimli öğrenmeden daha az aydınlatıcıdır (Cherkassky ve Mulier 2007). Öte yandan, sonuçlarla ilgili net bir geri bildirimin olmadığı denetimli öğrenmede, takviyeli öğrenme olan geri bildirim daha aydınlatıcıdır (Hinton ve Sejnowski 1999). Buna ek olarak, tek ajanlı takviyeli öğrenme görevi, iyi anlaşılmış, doğrulanabilir yakınsak (convergent) algoritmalar kullanılarak çözülebilir. Bu, basitlik ve genelleme özelliği birleştiğinde, takviyeli öğrenmeyi çok ajanlı öğrenme için çekici hale getirir.

Çok ajanlı bir sistemdeki bireylere önceden programlanmış davranışlar sağlanabilse de temsilcinin veya çok ajanlı sisteminin çıktısının zaman içinde iyileşmesi için genellikle çevrimiçi olarak yeni davranış kalıplarını öğrenmeleri gerekir (Stone ve Veloso 2000, Weiss 1999). Bu genellikle ortamın karmaşıklığından kaynaklanır, bu da iyi ajan davranışlarının önsel tasarımını imkânsız kılmasa da zorlaştırır. Ayrıca, değişen bir ortamda içgüdüsel davranış uygunsuz hale gelebilir. Genel olarak, hedefe dayalı gezinim problemi için, çok etmenli sistemlerin kolektif ve işbirlikçi davranışları için merkezi olmayan bir yöntem oluşturmak düşünülebilir (Güzel vd. 2019). Ekte ki Figürü ekleyelim.



Şekil 1.1 Sürü robotları için önerilen merkezi olmayan gezinim stratejisinin akış diyagramı (Güzel vd. 2019)

1.1 Çalışmanın Katkısı

Robotların hedefin konumu hakkında bilgi sahibi olup olmadıkları açısından bakıldığında iki bilgili (Celikkanat ve Şahin 2010) ve bilgisiz (Ferrante vd. 2012)

gezinme türü vardır. Bu çalışmada tasarladığımız gezinim, merkezi olmayan ve bilgili sayılır. Bu yöntem akın etme yöntemine benzer, ancak akın etme yönteminden farklı olarak bu yöntemde cazibe için ekstra bir ödül tanımlanmamaktadır. Bu yöntemin temel amacı sürüyü hedef noktasına mümkün olduğunca hızlı yönlendirmektir ve cazibe özelliği hızın düşmesine sebep olabilir, bu yüzden bu yöntemde cazibe tasarlanmamıştır ve sağlandığında da ekstra ödül verilmemektedir. Bireylerin zaman zaman sürüden ayrılmaları ve farklı yollar seçmeleri, diğerlerine yol açarak, tasarlanan sistemin genel performansı iyileştirebilir.

Örnek bir çalışmada (Buşoniu vd. 2010), ifade edilen çoklu ajanlı yöntemi kullanılmaktadır. Bu yöneme göre, her robot diğer robotların durumunu gözlemler ve onların durumlarına göre tepki verir. Bunu sağlamak için robotların durum uzaylarına diğer her robot için yeni özelliklerin eklenmesine ihtiyaç vardır. Bu yaklaşım, dağınık çalışma alanlarındaki sürüler için yerel çarpışmasız eylemi verimli bir şekilde belirleyebilme avantajına sahiptir (Long vd. 2018). Öte yandan bu yöntem Q-Öğrenmen'in büyük bellek ihtiyacı olan temel sorununu daha da büyütür. Durum uzayına herhangi bir yeni özelliğin eklenmesi, Q-tablosunun boyutunu artırmakta ve bununla beraber, bellek boyutu da önemli ölçüde artmaktadır.

Literatürde bilgilerimiz dâhilinde ilgili problemi çözecek bir çalışma mevcut değildir. Bu eksiklik bu çalışmanın temel motivasyonunu oluşturmaktadır. Tez çalışması, Q-Öğrenme tabanlı bir modelle kısmen dağınık ortamlarda merkezi olmayan çok ajanlı sistemlerde gezinme sorununa odaklanmıştır. Bu kapsamda Q-Öğrenmen'in büyük bellek ihtiyacına istinaden ve Q-tablo boyutunu önemli ölçüde azaltmak için "Sıkıştırılmış durum uzaylı denetleyici" veya SDUD adlı düşük maliyetli yeni bir yöntem sunulmuştur. Aynı miktarda bellekle daha büyük sorunları çözmek, fizibiliteyi artırmak ve maliyeti düşürmek uygulamanın kullandığı bellek miktarını azaltmanın avantajlarındandır.

Deneylerde, önerilen Q-Öğrenme tabanlı yaklaşımı kullanarak statik ve dinamik engellerin olduğu farklı ortamları içeren senaryolarda robot sürüleri ile başarılı bir şekilde gezdirmeyi başararak, önerilen yöntemin doğrulanması sağlanmıştır. Geliştirilen

yöntemi (SDUD) kullanarak robot grubuna üçüncü bir robot ekleyebildik; ancak normal bir bilgisayarın (bizim test makinemiz) bellek sınırlamaları, ortama daha fazla robotun eklenmesini kısıtladı.

Bu tezin önerdiği yöntem, merkezi olmayan bir stratejiye dayalı olarak tasarlanmıştır. Her robot, çevreyi yerel olarak gözlemler ve robotlar arasında eşleme için merkezi bir kontrolöre güvenmeden eylemi gerçekleştirir, böylece diğer robotların amaçları hakkında doğrudan bir bilgiye sahip olmaz. Bununla birlikte, öğrenme sürecini büyük ölçüde hızlandırmak, dallanma faktörü problemini hafifletmek (Amado ve Meneguzzi 2018) ve durum uzayındaki durumların çoğunu birkaç kez ziyaret eden öğrenilmiş politikalara daha fazla güven kazanmak için tüm robotların gözlemlerini yalnızca öğrenme aşamasında paylaşmak önerilen bir yöntemdir. Bu yöntem merkeziyetçilik kavramına zarar vermez çünkü sömürü veya modeli kullanma aşamasında merkezi bir ünite kullanarak veri paylaşımına ihtiyaç duyulmamaktadır. Gözlemleri paylaşma işleminin tezde önerilen sıkıştırma yöntemi için gerekli olmadığını unutmamalıyız.

Deneylerde, önerilen Q-Öğrenme tabanlı yaklaşımı (SDUD) kullanarak statik ve dinamik engellerin olduğu farklı senaryolarda üç robottan oluşan grubun engelleri atlatarak başarılı bir şekilde gezinmesini tamamladığı gösterilmiştir. Bu yöntem sayesinde deney robot grubuna normal bir bilgisayarda (bizim test makinemiz) üçüncü bir robot ekleyebilsek de bir sürü oluşturmak için üç robot yeterli değildir. Bunun için, araştırmayı bir adım daha ileri götürdük ve robot popülasyonunu herhangi bir ölçeğe çıkarmak için yeni bir yöntem geliştirdik, yani “Genelleştirilmiş SDUD” veya GSDUD.

GSDUD, girdi olarak alınan özellik sayısını önemli ölçüde azalttığı için hem normal hem de derin Q-Öğrenme için fayda sağlayacak özgün bir yöntemdir. Derin q-öğrenme kullanan yöntemlerde ise SDUD etkileri daha fazla araştırma gerektirirken, GSDUD’yi kullanmanın, robot sayısını artırma ve girdi özellik sayısını ve öğrenme süresini azaltma anlamında sürü robotiği sistemleri için faydalı olacaktır.

Önerilen yöntemler, üç farklı senaryo üzerinden test edilmiş ve doğrulanmıştır. Birinci senaryo kapsamında, SDUD’nin uygulanabilirliğini ve performansını, SDUD desteği

olmayan sistemle, “Normal Durum Uzaylı Denetleyici” (NDUD) karşılaştırarak gösterilmiştir. Sonraki senaryo ve ilgili deney, GSDUD’yi doğrulamak ve yöntemin başarılılığını göstermek için gerçekleştirilmiştir.

1.2 Benzer Çalışmalar

Gezinim açısından ele alındığında, bu tez çalışmasında kullanılan yöntem literatürdeki sürü hareketine (flocking) ile benzerlik göstermektedir. Sürü hareketi doğada görülebilen bir toplu davranış biçimidir, başka deyişle, bir grup katılımcının ortak bir yönde planlanmış hareketidir (Ferrante vd. 2012, Güzel vd. 2019). Sürü kullanan yöntemler, bireyler arasında çekme ve itmeyi kullanırlar, bazıları, davranışlarına karar vermek için hizalama bileşenlerini de kullanır (Ferrante vd. 2012, Reynolds 1987). Güncel bir araştırmada yapay sinir ağlarını kullanılarak merkezi olmayan gezinim problemi üzerine çalışılmıştır (Liv vd. 2020). Ancak, bu araştırmadaki yöntem yerel iletişim ve öğrenilen özelliklerin yakındaki ajanlar tarafından paylaşılması için merkezi bir birime ihtiyaç duyar, bu nedenle kısmen merkezi bir yapıya sahiptir.

Q-Öğrenme, envanter yönetimi (Giannoccaro ve Pontrandolfo 2002), robotikte davranış tasarımı veya gezinme (Millán vd. 2002, Matignon vd. 2007, Kober vd. 2013) ve oyun oynama (Hu ve Wellman 2003, Patel vd. 2011) gibi çeşitli problemlerde kullanılır. Ayrıca, Q-Öğrenme parametrik fonksiyon yaklaşım yöntemleri (Millán vd. 2002, Kumar vd. 2019) kullanılarak sürekli durum uzayı problemleri çözümünde de kullanılır.

Durum uzayını sıkıştırmaya ve boyutluluğu azaltmaya çalışan başka araştırmalar da mevcuttur (Vincent vd. 2008, Amado ve Meneguzzi 2018). Bu araştırmalar, temel olarak bellek kullanımını azaltmaktan ziyade genelleme yeteneği kazanmak amacıyla sinir ağlarını kullanarak durum uzayını sıkıştırmak için derin otomatik kodlayıcılar kullanırlar. Bundan farklı olarak, bu çalışmada kullanılan yöntem son durumlar üzerinde bir genelleme yapar ve durumların çoğunu aynı şekilde tutarak bellek kullanımını azaltmak için uzman bilgisini kullanır. Bellek azaltma yüzdesi hakkında bilgi bulunmadığı için yöntemleri karşılaştırmak mümkün olmamıştır.

Başka bir çalışmada (Gehring vd. 2018), yazarlar beklenen sonuçları tahmin etmek için doğrusal eylem modellerini bu sonuçlara uyduran bir yöntem sunmuşlardır. Daha sonra bir strateji ile söylenen doğrusal eylem modellerini sıkıştırmışlar, bu strateji, mevcut alan miktarını öğrenme modelleri için azaltır ve Q-Öğrenme'ye göre bir avantaj sağlar. Diğer bir yöntem, bilgi sıkıştırma için durum toplamadır (SAIC) (Mostaani vd. 2020). Bu yöntem, merkezi sisteme sahip sınırlı üye ile çok ajanlı sistemde ajanlar arasındaki mesajlarını sıkıştırmayı amaçlar. Bir diğer çalışma Siber-Fiziksel Sistemler (CPS) verilerini analiz etmek için kullanılan bir derin öğrenme tekniği olan İyileştirilmiş DeepRT'yi (IDEepRT) küçültmek için bir sıkıştırma şeması önermektedir (Rajalakshmi ve Sathappan 2020). Bu işlem, model çalışma süresini azaltmak için Bayes matrisini çarpanlarına ayırma yöntemini kullanılarak yapılır. Bu çalışmada, IDEepRT'deki özellik uzayını sürekli olarak düzeltmek için takviyeli öğrenme ve IDEepRT'yi birleştirilmiştir. Başka güncel bir çalışma bant genişliği ve güç tüketiminden tasarruf etmek için ağ üzerinden akan verilerin boyutunu küçültmeyi amaçlamıştır (Kim ve diğerleri, 2019). Bu çalışma deneyin boyutunu küçültmek için kodlayıcı ve kod çözücü kullanan bir sıkıştırma stratejisi uygulamaktadır.

Q-tablosunu paylaşmak için (Jaidee ve Muñoz-Avila 2012, Amado ve Meneguzzi 2018), tüm robotların q-tablosunu her öğrenme bölümünden sonra birleştirmek için bir normalleştirme adımını kullanırlar. Bu işlemi yapmak için her ajan geçtiği durumların ziyaret sayısını tutmalıdır, bu sebepten dolayı bu yöntem ekstra bellek ve ekstra işleme ihtiyaç duyar. Örneğin, bu tabloyu “Python” programlama dilinde ziyaret tablosu olarak adlandıırırsak, Q-Tablo için “FLOAT” tipini (16bit) ve ziyaret tablosu için “INT” tipini (12 bit) seçersek, her bir ajan için hafıza kullanımı %75 artacaktır. Bu sistem, büyük ajan sayısına sahip sistemler için oldukça maliyetlidir ve hatta bazı durumlar için pratik değildir. Buna karşın, gerçek zamanlı olarak her bölüm sırasında öğrenebilmek bu yöntemin avantajıdır. Ancak, bizim yöntemimizde her robot sırayla Q-tablosunu her bölüm yerine her zaman adımında eşzamanlı olarak kullanır ve günceller, bu nedenle her bir ajanın ziyaret sayısını tutmak için ekstra normalleştirme ve birleştirme aşamasına veya ekstra belleğe ihtiyaç duymamaktadır. Bu sebepten dolayı, genel olarak daha hızlıdır ve birçok problem için uygundur fakat bazı gerçek zamanlı problemlere uygulanamayabilir.

İletişim veri boyutuna göre bakılırsa, her iki yöntemin de yalnızca eylem, mevcut durum ve eylemin sonucuyla ilgili verileri bant genişliğini düşürecek şekilde göndermesi gerekir. Bizim örnekte bu miktar bir gerçek sayı tipi ve dokuz int tipi boyutunun toplamına eşit olacaktır.

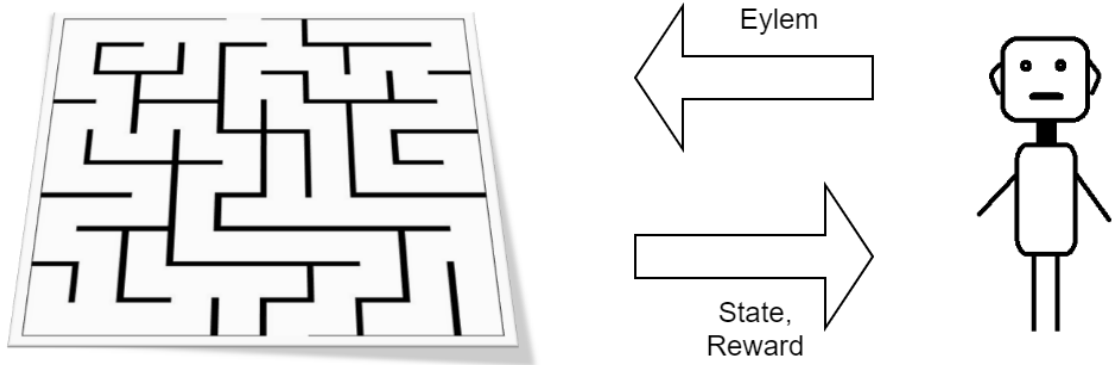


2. LİTERATÜR

Bu bölümde, Q-Öğrenmeyi kullanan veya performansını artırmak için yöntemler sağlayan araştırmalar ve ilişkili çalışmalar ile ilgili literatür çalışmaları sunulmuştur.

Q-Öğrenme, ilk olarak (Christopher vd. 1989) tarafından tanıtıldı. Bu yöntem modelsiz Takviyeli (pekiştirmeli) öğrenme yöntemidir. Ayrıca bir asenkron dinamik programlama (DP) yöntemi sayılır. Bu yöntem Ajanlara, etki alanlarının haritalarını oluşturmalarına gerek kalmadan eylemlerin sonuçlarını deneyimleyerek Markovian etki alanlarında en uygun şekilde hareket etmeyi öğrenme yeteneği sağlar. Bu yöntemle benzer çalışmalar Watkins'ın çalışmasından önce de yapılmıştır, (Richard ve Sutton 1988, Richard vd. 1984) çalışması zamansal farklılıklar (Temporal Difference) yöntemine benzer şekilde ilerler: bir ajan, belirli bir durumda bir eylemi dener ve sonuçlarını, aldığı ani ödül veya ceza ve onun değerine ilişkin tahmini değerlendirir. Tüm durumlardaki eylemleri tekrar tekrar deneyerek, genel olarak en iyi olanı öğrenir ve uzun vadeli indirimli ödüle göre değerlendirilir. Ardından 1989'da Q-Öğrenme Watkins tarafından tanıtılır.

Q-Öğrenme, çok karmaşık uygulamaların temeli olarak çalışsa da aslında ilkel bir öğrenme biçimidir. Q-Öğrenmenin yakınsamasının ispatı (Melo 2001, Christopher vd. 1992b)'de yapılmıştır.



Şekil 2.1 Q-Öğrenme

2.1 Q-Öğrenmeyi İyileştiren Çalışmalar

Q-Öğrenme’de eylem değerlerinin fazla tahmin edilmesinden dolayı Q-Öğrenme performansı düşmektedir. Bu sorunu çözmek için “Çift Q-Öğrenme” adlı çalışma (Hasselt 2010), stokastik ortamlarda Q-Öğrenme algoritmasının performansını iyileştirmeyi amaçlamaktadır. Q-Öğrenmenin abartılı tahminleri maksimum beklenen eylem değeri hesaplamada pozitif bir önyargıya sebep oluyor. Bunun sebebi ise maksimum eylem değerini kullanmasıdır. Araştırmacılar bu çalışmada istenilen herhangi bir tesadüfi değişken kümesi için maksimum beklenen değere yaklaşmayı amaçlayarak alternatif bir yol sunarlar. Bu yöntemle elde edilen çift tahmin yönteminin, bazen beklenen maksimum değeri fazla tahmin etmek yerine olduğundan az tahmin ettiği gösterilmiştir. Yeni bir politika dışı takviyeli öğrenme algoritması olan çift Q-Öğrenme’yi oluşturmak için Q-Öğrenme’ye çift tahmin edici uygulanır. Bu çalışmada yapılan deneylerle yeni algoritmanın en uygun politikaya yakınsadığını ve Q-Öğrenmenin fazla tahmin edilmesi nedeniyle düşük performans gösterdiği bazı ayarlarda daha iyi performans sergilediği gösterilmiştir.

“Zap Q-Öğrenme” (Devraj ve Meyn 2017a) çalışmasında tanıtılan Zap Q-Öğrenme algoritması, Watkins’ın orijinal algoritmasının ve rakiplerinin çeşitli açılardan geliştirilmiş halidir. Asimptotik varyansı optimal olacak şekilde tasarlanmış bir matris kazanç algoritmasıdır. Ayrıca, bir ODE analizi, bu makalede kullanılan geçici davranışın deterministik bir Newton-Raphson uygulamasına yakın bir eşleşme olduğunu ileri sürülmektedir.. Bu, matris kazanç dizisi için iki zaman ölçeği güncelleme denklemi ile mümkün olmaktadır. Bu çalışmada yapılan analiz, önerilen yaklaşımın ideal olmayan parametrelili sistem ayarları için bile kararlı ve verimli sonuçlara yol açacağını öne sürüyor ve hızlı yakınsamayı doğruluyor.

Bir diğer yöntem ise belirsiz bilgileri tutmak için Bayesian bir yaklaşım benimser, yazarlar “Bayesian Q-Öğrenme” (Dearden vd. 1998) adlı çalışmada Q değerleri üzerinde olasılık dağılımlarını koruyarak ve yayarak Watkins’ın Q-Öğrenmesini

genişletirler. Dağılımlar, her eylem için bilgi değerine miyop bir tahmin¹ hesaplamak ve dolayısıyla keşif ve sömürüyü en iyi dengeleyen eylemi seçmek için kullanılır. Ardından yazarlar bu çalışmada algoritmalarının yakınsama (convergence) özelliklerini oluşturarak önerilen modelin diğer iyi bilinen modelsiz keşif stratejilerine göre önemli iyileşmeler gösterebileceğini deneysel olarak kanıtlamışlardır.

2.2 Q-Öğrenmeyi kullanan çalışmalar

Glorennec vd. “Fuzzy Q-Öğrenme” adlı çalışmada (Glorennec ve Jouffe 1997), Watkins’in hem eylemlerin hem de Q-fonksiyonlarının bulanık kurallardan çıkarıldığı Fuzzy çıkarım sistemler için Q-Öğrenme adlı bir uyarlamasını önerirler. Bu yaklaşım, araba merkezleme (Cart-Centring) probleminde Genetik Algoritma yönteminden daha iyi sonuçlar üretmiştir.

Azar vd, tarafından yapılan “Süratli Q-Öğrenme” çalışması (Azar vd. 2011) Q-Öğrenmenin yavaş yakınsama sorununu azaltmak için yeni bir yaklaşım önerir. Yazarlara göre Q-Öğrenmenin yavaş yakınsamasının ana nedeninin, örnek tabanlı stokastik tahminin (ki azalan bir öğrenme oranını kullanıyor) ve Bellman operatörünün bilgiyi tüm eğitim süresi boyunca yaydığı gerçeğinin (özellikle γ değeri 1’e yakın olduğunda) birleşimi olduğunu söylüyorlar. SQL (Speedy Q-Learning) her zaman adım için eylem değer fonksiyonundan aldığı iki ardışık yaklaşık değeri kullanır. Bu yöntem uzay karmaşıklığını standart Q-Öğrenmeye kıyasta iki kat olur. Böylece, SQL’in güncelleme kuralındaki terimlerden biri daha saldırgan bir öğrenme oranı kullanabilir ve bunun sonucu olarak yakınsama hızı oran olarak standart Q-Öğrenmesinden daha hızlıdır.

Kayda değer başka bir çalışma ise “Artımlı Çok Adımlı Q-Öğrenme” (Peng ve Williams 1994). Bu araştırma Q-Öğrenmeyi, tipik olarak aktör-eleştirel (actor-critic) öğrenmede kullanılan, TD(A) geri dönüş tahmin süreci ile birleştiren artımlı bir algoritma sunar. İyi bilinen bir dinamik programlama tabanlı takviyeli öğrenme yöntemi

¹ Myopic approximation: uzağı göremeyen anlamındadır

Aktör-eleştirel öğrenmsidir. “A” parametresi, krediyi eylem dizileri boyunca dağıtmak için kullanılır, bu da daha hızlı öğrenmeye yol açar ve ayrıca kaba(coarse) durum-uzay nicelemenin Markovyen olmayan etkisini hafifletmeye yardımcı olur. Bu işlem sonucu ortaya çıkan algoritma, Q-Öğrenme, $Q(\lambda)$ -öğrenme ve aktör-eleştirmen öğrenme paradigmalarının bazı iyi özelliklerini birleştirir.

Sürekli alanlarda çalışan bir Q-Öğrenme yöntemi Millan vd, “Sürekli-Eylem Q-Öğrenme”de (Millán vd. 2002) sunuyorlar. Giriş uzayını bölmek için artımlı bir topoloji koruma haritası (ITPM) kullanır ve öğrenme sürecini başlatmak ve varsayılan değer belirlemek için önyargı (bias) içerir. ITPM birimleri, giriş uzayını M olası ayrık eylemlerin Q değerlerine eşler. Bu yöntemin ayrık eylem sürümüne göre daha iyi performansa sahip olduğu deneysel sonuçlarla gösteriliyor.

“Takviyeli Öğrenmenin Genelleştirilmesi için Bulanık Q-Öğrenme” (Berenji 1996), artımlı dinamik programlama yapmak için bir yöntem olan GARIC-Q’yu tanıtmaktadır. Yazarlar, Fuzzy Q-Learning tarafından en üst düzeyde kontrol edilen ve yerel düzeyde, her ajanın GARIC’e dayalı olarak öğrendiği ve çalıştığı akıllı ajanlar topluluğu kullanıyorlar. Bulanık kurallar kullanarak ve girdi alanının genelleştirilmesi yoluyla GARIC-Q bulanık Q-Öğrenmenin hızını ve uygulanabilirliğini geliştirir. Bu yöntem Q-Öğrenme ile kural tabanlı akıllı sistemler arasındaki boşluğu kapatır.

“Maxmin Q-Öğrenme” (Lan vd. 2020)’e göre fazla tahmin yanlışlığı, Q-Öğrenmenin, maksimum tahmin edilen eylem değerini kullanarak maksimum eylem değerine yaklaştığına işaret eder. Maxmin Q-learning’de yazarlar, yanlışlığın performansla nasıl etkileşime girdiğini ve mevcut algoritmaların önyargıyı ne ölçüde azalttığını anlamaya çalışırlar. Bu araştırma, aşırı tahmin yanlışlığının öğrenme verimliliği üzerindeki etkisinin çevreye bağlı olduğunu vurgular ve önyargıyı esnek bir şekilde kontrol etmek için bir parametre sağlayan Maxmin Q-Öğrenme adı verilen bir Q-Öğrenme genelleştirmesi önerir; ek olarak, teorik olarak, Maxmin Q-Öğrenme için normal Q-Öğrenmeden daha düşük bir yaklaşım varyansı ile tarafsız tahmine yol açan bir parametre seçimi ile mümkün olduğunu gösterir. Yazarların iddiasına göre bu yöntem

tahmin yanlılığını, oyuncak ortamlarında daha iyi kontrol ediyor ve ayrıca bu yöntemle çeşitli karşılaştırma problemlerinde üstün performans elde ediliyor.

Takviyeli öğrenmedeki bir diğer ikilem, bu çalışmada tanıtılan “geriye dönük öğrenme” yönteminin çözmeye çalıştığı sömürü ve keşif arasında seçim yapma stratejisidir. Q-Öğrenme ve Sarsa algoritmalarının farklı özellikleri vardır. Genel olarak konuşursak, Sarsa algoritmasının avantajı daha hızlı yakınsama özelliğidir, öte yandan Q-Öğrenme algoritması da daha iyi bir nihai performansa sahiptir. Bununla birlikte, Sarsa algoritması yerel minimumları geçemeyip takılabilirken, normal Q-Öğrenme de eğitimini tamamlamak için daha uzun zamana ihtiyaç duyar. “Geriye doğru Q-Öğrenme” (Wang vd. 2013) makalesi, Q-Öğrenmenin Sarsa algoritması ile nasıl birleştirileceğine odaklanmakta ve Sarsa algoritması ve Q-Öğrenmede uygulanabilen geriye dönük Q-Öğrenme adı verilen bir yöntem sunmaktadır. Doğrudan Q değerlerini ayarlayan geriye dönük Q-Öğrenme algoritması adlı bir eylem seçim stratejisi önerir ve ardından Q değerleri, eylem seçim politikasını dolaylı olarak etkiler. Bu nedenle, önerilen algoritmalar hızı arttırdığı için nihai performans iyileşir. Bu çalışmada yapılan tüm simülasyonlar, araştırmanın önerdiği geriye dönük Q-Öğrenme tabanlı RL algoritmasının, normal Q-Öğrenme ve Sarsa algoritmalarından daha iyi performans gösterdiğini göstermektedir.

“Enterpolasyon tabanlı Q-Öğrenme” (Szepesvári ve Smart 2004), çalışmada yazarlar, yerel fonksiyon yaklaşımı yöntemleriyle birleştirilmiş toplam beklenen indirimli maliyet ölçütü altında sürekli durum uzayları için bir Q-Öğrenme çeşidi önermektedir. Fonksiyon tahmincisinin belirli enterpolasyon özelliklerini sağlaması koşuluyla, elde edilen algoritmanın bir olasılık ile yakınsadığı gösterilir. Araştırmada limit fonksiyonunun, sabit nokta operatörünün keşif politikasının durağan dağılımına ve fonksiyon yaklaşımı yöntemine bağlı olduğu Bellman tipi sabit noktalı bir denklemi sağladığı gösterilmiştir.

Diğer ajanların deneyim ve bilgilerini kullanmayı amaçlayan bir çalışma ise “Uzmanlığa dayalı işbirlikçi Q-Öğrenme” adlı çalışmadır (Ahmadabadi ve Asadpour 2002). Bunu yaparak bir öğrenme aşamasında olan ajan daha az hata yapabilir, daha

hızlı öğrenebilir ve daha iyi genelleyebilir (görünmeyen durumlar için bazı kurallar oluşturabilir). Bu avantajlar, öğrenmekte olan ajan, diğer ajanların bilgisinden kendi gereksinimleri için kurallar çıkarabiliyorsa yararlı olacaktır. Bunu yapmanın olası bir yolu, öğrenen ajanın diğer ajanlara bazı uzmanlık değerleri atamasını ve bilgilerini buna göre kullanmasını sağlamaktır. Yazarlar bu çalışmada işbirlikli bir öğrenme yöntemi olan “Ağırlıklı Strateji Paylaşımı” (weighted strategy sharing-WSS) adlı yöntemi ve ayrıca takviyeli öğrenmede ajanların bilgi seviyesini ölçmek için kullanılan bir yöntemi sunarlar. Bu yöntemlerle her bir ajan, takım arkadaşlarının uzmanlığını ölçerek ve onların bilgisine ağırlık vererek onlardan öğrenir.

Sadece sömürü yapmak, ajanın yerel olarak en uygun politikalara hızlı bir şekilde ulaşmasına neden olurken, aşırı keşif yapmak, öğrenme sürecini hızlandırırsa ve yerel olarak en uygun politikalardan kaçınmaya izin verse bile Q-Öğrenme algoritmasının performansını düşürür. “Metropolis ölçütüne dayalı yeni bir Q-Öğrenme algoritması” adlı çalışmada (Guo vd. 2004), Q-Öğrenmenin keşif ve kullanımını dengelemek için benzetimli tavlama(Simulated Annealing) algoritmasının Metropolis kriteri tanıtılmış ve bu kriterle dayalı değiştirilmiş Q-Öğrenme algoritması SA-Q-Öğrenme sunulmuştur. SA-Q-Öğrenme, Q-Öğrenme ve Boltzmann keşfinden daha hızlı yakınsar ve aşırı keşif nedeniyle performans düşüşü yaşamaz.

“Dinamik bulanık Q-Öğrenme kullanarak bulanık çıkarım sistemlerinin çevrimiçi ayarlanması” adlı çalışma (Er ve Deng 2004), bulanık çıkarım sistemlerini (Fuzzy Inference Systems - FIS) çevrimiçi olarak ayarlayabilen dinamik bir bulanık Q-Öğrenme (Dynamic Fuzzy Q-learning - DFQL) yöntemi sunar. Bu yöntem kendi kendini organize eden çevrimiçi öğrenme algoritmasında yapı ve parametre tanımlaması, yalnızca Q-Öğrenmesine dayalı olarak otomatik ve eşzamanlı olarak gerçekleştirir. Bu yöntem, sürekli değerlere sahip durumlar ve eylemlerle baş edebilmeyi sağlar. Bulanık kurallar, hızlı pekiştirmeli öğrenme için önyargı bileşenlerini dahil etmenin doğal bir yolunu sağlar.

“Gelişmiş Q-Öğrenme kullanarak bir mobil robotun en uygun yol planlamasını çözme” adlı başka güncel bir çalışmada (Low vd. 2019), yazarlar, Q-Öğrenme’nin yavaş

yakınsama problemini iyileştirmeye çalışıyorlar. Bunun için bu çalışmada kısmen yönlendirilmiş Q-Öğrenme kavramı tanıtılmaktadır, burada Q-Öğrenmenin başlatılmasını iyileştirmek için çiçek tozlaşma algoritması (Flower Pollination Algorithm - FPA) kullanılmakta, bunun da öğrenme sürecini hızlandırdığı gözlemlenmiştir.

Abdoos vd, "Takviyeli Öğrenmenin Verimli Bir Donanım Uygulaması: Q-Öğrenme Algoritması " adlı çalışmada (Spano vd. 2019), gerçek zamanlı uygulamalar için uygun olan Q-Öğrenme algoritmasını uygulayan verimli bir donanım mimarisi önermektedir. Donanımın ana özellikleri yüksek verim, düşük güç ve sınırlı olmasıdır. Algoritma, yaklaşık çarpanlara dayalı bir teknik ile donanım karmaşıklığını azaltmayı öneriyor.

“Q-Öğrenme için En Hızlı Yakınsama” adlı bir diğer çalışmada Devraj ve Meyn (Devraj ve Meyn 2017b) tarafından yapılmıştır. Bu makalede tanıtılan Zap Q-Öğrenme algoritması, Watkins’in orijinal algoritmasının ve ayrıca son rakiplerinin çeşitli açılardan geliştirilmiş halidir. Başka deyişle, asimptotik varyansı optimal olacak şekilde tasarlanmış bir matris kazanç algoritmasıdır. Ayrıca, bir ODE analizi, geçici davranışın deterministik bir Newton-Raphson uygulamasına yakın bir eşleşme olduğunu öne sürülür Bu, matris kazanç dizisi için iki zaman ölçekli güncelleme denklemi ile mümkün olmaktadır. Bu yöntem, ideal olmayan parametrelili ayarlar için bile kararlı ve verimli hesaplamaya yol açar, ayrıca ideal olmayan durumlarda bile hızlı bir yakınsamaya sahiptir.

“Çoklu ajan Q-Öğrenmesine dayalı sabit olmayan ortamlarda trafik ışığı kontrolü” adlı çalışma (Abdoos vd. 2011), ise trafik ışığı kontrolü için Q-Öğrenmeyi kullanır. Yazarlar, çok ajanlı bir sistem olarak büyük bir trafik ağını modelliyorlar ve bunun için çok ajanlı takviyeli öğrenme tekniklerini kullanıyorlar. Bu çalışmada, Q-Öğrenme, ortalama kuyruk uzunluğuna sahip durumları tahmin etmek için kullanılmaktadır. Eylem uzayının parametrik bir temsili, yöntemi farklı kavşak türlerine genişletilebilir hale getirmiştir ve sonuçlar, yöntemlerin farklı trafik talepleri altında sabit zaman yöntemine göre üstünlüğünü göstermektedir.

3. MATERIAL VE YÖNTEM

Kısmen dağınık ortamlarda merkezi olmayan çok ajanlı sistemlerde gezinmek zorlu bir sorundur (Güzel vd. 2018). Bu çalışma kapsamında ele alınan senaryolarımızda, robotların, tümü aynı denetleyiciyi kullanarak herhangi bir dinamik (yani diğer robotlar) veya statik engellerle çarpışmadan kısa bir yoldan bir hedef durumuna gitmeyi başarmalıdır. Robotları koordine etmek için merkezi bir kontrol ünitesinin olmaması, dinamik engellerle çarpışmayı önlemeyi ve aynı zamanda en uygun yolu planlamayı zorlaştırmaktadır. Ayrıca, problemin gerçek durum uzayı süreklidir ve çok büyük bir Q-Tablosu gerektirmektedir. Bu çok önemlidir, çünkü bu problem için durum uzayını büyük birimlere bile ayırarak Q-Tablo hafıza boyutu sınırlarına kolayca ulaşılmaktadır. Birkaç ekstra özelliğin eklenmesi, Q-Öğrenme kullanmayı pratik problemlerde imkânsız hale getirmektedir. Öte yandan Q-Öğrenme, eğitim öncesinde ortam modelinin sağlanmasını gerektirmiyor ve modelsiz bir algoritmadır.

Bu bölümde ilk önce takviyeli öğrenmenin temellerinden bahsedilecek ve ardından bu araştırmadaki problem modeli ve ayrıntıları anlatılacak. Son olarak önerilen yöntem detaylandırılacaktır.

3.1 Takviyeli Öğrenme

Takviyeli Öğrenme, verilen ortamda maksimum ödülü elde etmek için en uygun davranışı öğrenmekle ilgilidir. Başka deęişle Takviyeli Öğrenme karar verme bilimi sayılır. Bu en uygun davranış, çevre ile etkileşimler ve nasıl tepki verdiğine dair gözlemler yoluyla öğrenilir, tıpkı çocukların çevrelerindeki dünyayı keşfetmelerine ve bir hedefe ulaşmalarına yardımcı olan eylemleri öğrenmelerine benzer şekilde.

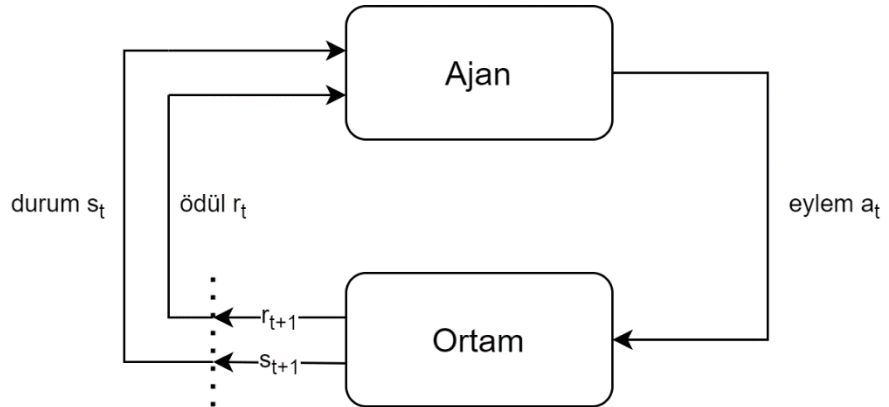
Bir gözetmen yokluęunda, ajan(öğrenci), bağımsız olarak ödülü en üst düzeye çıkaran eylemlerin sırasını keşfetmelidir. Bu keşif yapma süreci, deneme yanılma ile aramaya benzer. Eylemlerin kalitesi, gecikmiş ödül denilen gelecekte getirebilecekleri ödülle ve ayrıca aldıkları anlık ödülle ölçülür. Takviyeli öğrenme başarılı sonuca varan eylem

sıralarını gözetmen yardımı olmadan görünmeyen bir ortamda yapabildiği için çok güçlü bir algoritmadır.

3.1.1 Takviyeli öğrenme temelleri

Takviyeli öğrenmenin çok temel bir tasarımı “öğrenme” ve “eylem” döngüleri ile Şekil 3.1’deki diyagramda verilmiştir. Burada, daha önce de anlatıldığı gibi her ajan t adım-zamanında içinde bulunduğu duruma (S_t) göre en iyi eylemi seçmeyi öğrenmek için ortamla etkileşime girer. Ajanın eylemi, sırayla, çevrenin durumunu S_t ’den S_{t+1} ’e değiştirir (Şekil 3.1’de gösterildiği gibi) ve ajan için bir ödül r_t üretir. Daha sonra ajan bu yeni durum için mümkün olan en iyi eylemi seçer (S_{t+1}), böylece bir ödül r_{t+1} alır. Bir yineleme periyodu boyunca ajan aldığı ödülleri kullanarak ortamın belirli bir durumunda yapılabilecek "en iyi eylemin hangisi olduğuna dair kararını geliştirmeye çalışır.

Dolayısıyla buradaki ortamın rolü, ajana, ajanın tepki vermesi gerekebilecek problemde var olabilecek farklı olası durumlar veya bunların temsili bir alt kümesini sunmaktır. Ajana öğrenme aşamasında yardımcı olmak için şemadaki ortam (çevre), mevcut durumda ajan tarafından seçilen eylem kararlarının karşılığında alınan cezayı (olumsuz bir ödül) veya ödülü de verir. Dolayısıyla ödül, yalnızca eylemin değil hem eylemin hem de durumun bir işlevidir. Bu, aynı eylemin farklı durumlar altında farklı bir ödül alabileceği anlamına gelir.



Şekil 3.1 Takviyeli öğrenmenin şeması

3.1.2 Ödül fonksiyonu

Belirli bir durumda yapılan doğru bir eylemin gerçek ödülü hemen gerçekleşmeyebilir. Örnek olarak, dışarı çıkıp oynama ya da şimdi oturup yaklaşan sınavlarınız için çalışma ve sınavlardan sonra oynama seçeneğiniz olan gerçek bir hayat senaryosu hayal edelim. Şimdi oynamak, olumlu bir sonuç olarak değerlendirilirse, takviyeli öğrenme açısından bir ödül olarak değerlendirilebilecek küçük bir adrenalin veya istek anında verilebilir. Oysa şimdi çalışmak kısa vadede sıkıcı görünebilir, ancak uzun vadede muhtemelen çok iyi sonuç verebilir, bu daha büyük bir ödül olarak düşünülebilir, ancak ödül gelecekte gerçekleşir. İndirim faktörü, karşılaştırma yapmak ve gelecekteki ödülleri şimdiki zamana indirmek içindir ve bu sorun için kullanılan oldukça iyi kurulmuş çözümlerden birisidir.

Takviyeli öğrenmedeki diğer bir karmaşıklık yaratan konu, ödüllerin rastgele olma (olasılıklı) doğası veya ödüllerdeki belirsizliktir. Daha sonra bir ödül (iyi notlar) için şimdi çalışmanın aynı örneğini ele alalım. Diyelim ki kursta 10 bölüm var ve soruların bu bölümlerden sadece altısından geleceğini biliyoruz, ancak soruların hangi belirli altı bölümden geleceğini ve her birinin ağırlığının ne kadar olacağını bilmiyoruz. Seçilen altı bölüm sınavlarda yer alacaktır. Ayrıca, ajanımızın bir açık hava oyunu oynamak için harcayabileceği 3 saatlik bir zaman diliminde, 10 bölümden sadece herhangi birini çalışabileceğini varsayalım.

Dolayısıyla, gelecekteki ödülün algılanan değerinin oynamak yerine üzerinde çalışmaya değer olduğunu varsaysak bile, şimdiki zamanı hiçbir soru sorulmayan belirli bir bölümü çalışarak geçirip geçiremeyeceğimizden emin değiliz. Seçilen bölümün önemli olduğunu düşünsek bile, bu bölümden gelecek soruların puanlarının özgül ağırlığını bilmiyoruz. Bu nedenle, şimdiki çalışmanın getirileri yalnızca gelecekte gerçekleşmekle kalmayacak, aynı zamanda olasılıklı veya belirsiz de olabilir.

Bir diğer önemli husus, ödüllerin geçmişte yapılan belirli eylemlere atfedilmesidir. Yukarıdaki örnekle devam edersek, 10 bölümden veya karşılık gelen 10 birimden oynayacağını varsayalım, ajanın 10 bölümden herhangi birini çalışacağı, altı örneği

rastgele seçeceğine karar veriyoruz (rastgele seçildiğini varsayın) ve kalan dört yuvada oynayacaktır. Belirli seçimler küçük ama yakın ödül ve daha büyük ama gelecekte olma olasılığı olan ödüllerden oluşan, nihai toplam ödülü en üst düzeye çıkarma amacı ile yapılır.

Ajanın yaptığı seçimlerle sonunda sınavında %50 puan aldığını varsayalım. Ayrıca, çalışmayı seçtiğimiz altı bölümden, soruların sırasıyla %5, %8, %12 ve %15 puan ağırlıklarıyla yalnızca dördünden geldiğini ve kalan %50 soruların yerine oynamaya karar verdiğimiz için hazırlayamadığımız dört bölümden geldiğini varsayalım, sonuç olarak ajan bu senaryoda yanıtladığı sorulardan 40 puan almıştır.

Şimdi, bu ödülün atfedilmesi için bir çözüm, bu ödülü, temsilcinin oynamak yerine çalışmaya karar verdiği altı alanın her birine eşit olarak bölmek olabilir. Bir başka çözüm de her belirli (çalışılmış ve sınavda çıkmış) soruya, ajanın o soruya çalışmaya karar verdiği bölümden gelen soru ağırlığı kadar ödül vermemiz olabilir.

3.1.3 Takviyeli öğrenmede “Durumlar”

Bir Takviyeli Öğrenme eğitimi sırasındaki durum, ortamın eyleme geçmesi için ajana sunduğu bir bağlamı temsil eder ve ardından bu durum bağlamında gerçekleştirilen eylem için bir ödül oluşturur. Böyle bir durum, en iyi ihtimalle, ajanın gerçek hayattaki belirsizliklerle uğraşırken üretim/dağıtımda¹ karşılaşacağı gerçek ortam senaryosunun basitleştirilmiş bir temsidir (Sewak vd. 2020).

Gerçek hayat senaryosuna uygulanmış bir takviyeli öğrenme ajan durumu, ajanın tüm algılayabileceği yönlerle temsil edilir. Senaryonun sonucunu etkileyebilecek (ulaşılabilir durum ve belirli bir durumda bir eylemde bulunulduğunda alınan ödül) ve ölçülebilen her şey ideal olarak duruma dahil edilmelidir. Öte yandan bir sürü faktör sürecin sonucunu etkileyecek olabilir, fakat önceden bunların bazıları bilinemez veya

¹ Production/deployment

ölçülemez. Bu faktörleri bilemedikçe veya belirli bir doğruluk derecesi ile ölçemedikçe, tüm bu faktörleri ve oluşumları bundan böyle gürültü olarak adlandırıyoruz.

Bir eğitim ortamından bir test/doğrulama veya diğer ön üretim ortamlarına (ajanın hala gerçek dünya ortamıyla karşı karşıya olduğu, ancak eylemlerinin sanal alanda olduğu ve gerçek dünyadaki durumu değiştirmek için kullanılmadığı senaryo) geçiş yaparken, ajanı sürekli olarak eğitmek ve herhangi bir üretim dağıtımına hazırlamak için veriler toplanacaktır. Eğitim ve üretim öncesi ortamlar arasında çok fazla farklılık olduğu zamanlarda, eğitim durumlarının yeniden yapılandırılması ve bazı aşırı basitleştirilmiş varsayımların yeniden ele alınması gerekebilir, böylece eğitim sırasındaki durumlar gerçek yaşam koşullarını daha iyi temsil edebilirler. Ajanlar daha sonra doğrulama yeniden oluşmadan önce eğitim ortamından yeni durum dizileri ve ödüller ile yeniden öğrenmeleri sağlanır.

Bazen durum, gerçek zamanlı olarak ortamın veya ajanın erişebildiği tüm girdilerin (sensörler ve diğer girdiler) yığılmış bir matrisinden oluşabilir. Ancak çoğu zaman durumlar, bir ajanı eğitmek için en uygun olmayabilir ve daha sonra, girdilerden, gözlemlerden elde ettiğimiz tüm gerçek zamanlı ve tarihsel verilerden durumu nasıl tezahür ettirileceğinin düşünülmesi gerekebilir.

3.1.4 Tic-Tac-Toe problemi

Tic-tac-toe (Şekil 3.2) gibi basit bir oyun örneğini alırsak, durumumuz n 'nin temsil ettiği basit bir $[n, 9]$ (veya bu konuda $[n, 3, 3]$) matrisinden oluşabilir. Burada n olayların veya deneylerin sırasını göstermektedir. Dizideki belirli bir olay için, durum uzayı bir dizi veya $[9]$ uzunlukta veya alternatif olarak $[1, 9]$ matrisi (veya bir $[3, 3]$ matrisi) olabilir, burada dizinin/matrisin her hücresi bir tanesini gösterir. Gerçek hayattaki tic-tac-toe probleminde hücrenin değeri, hücre boş ise 0, içinde "x" varsa 1, "o" varsa 2 olabilir. Bu basit örnek, ajan için elimizde yeterli sayıda örnek deney (bölüm) ile onu eğitmek istediğimiz tarafa ("x" veya "o") karşılık gelen stratejileri öğrenmesi için yeterli olmalıdır.

X		O
X	O	
O		

Şekil 3.2 Tic-Toc-Toe oyunu

Durumun (olay alanı) $[1, 9]$ veya $[3, 3]$ matrisi olması, her hücre gerçek hayattaki bir tic-tac-toe'daki değeri temsil ettiği sürece fark etmez. Ayrıca, olayların önceki sırası da fark yaratmaz. Önemli olan tek şey matrisimizin dokuz hücresinin her birindeki (0, 1 veya 2) değerleridir ve ajanımız bu durumdan farklı eylemler için ödül verilen en iyi stratejiyi öğrenebilmelidir. Böyle bir ajan oynamak istediği hücrenin (boş ise) konumuna karşılık gelen eylemi çıktı olarak verecektir. Ajandan gelen eylem, öncekine benzer bir sonraki durumu belirleyecek ve yönlendirecektir, fakat bu sefer bir boş hücrenin (0 değeri) ajanımız tarafından devralınması dışında. Ajan kazanırsa, ödülü alır, aksi takdirde sonraki durum (eylem seçmek), oynaması için insana veya düşman ajana sunulur. İnsanın/düşmanın geçerli dönüşünde, ortaya çıkan durum, eylem için tekrar ajanımıza geri gönderilir ve süreç, bir kazanan olana veya oynayacak boş hücre kalmayana kadar devam eder.

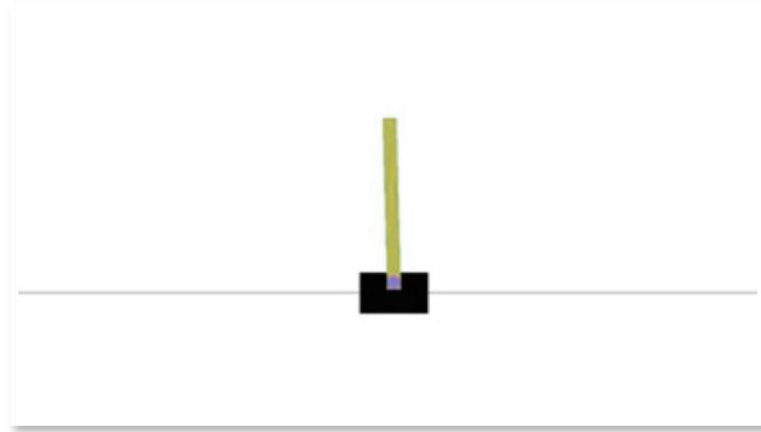
Bu basit örnek, gerçek yaşam senaryosunu ve gözlemleri, ajan için anlamlı olacak matematiksel gösterime ve bilgisayar bilimi veri yapılarına nasıl dönüştüreceğinin kavramsallaştırmasının sağlanması için belirlenmiştir. Seçilen durum formülasyonu gerçek hayattaki tüm önemli senaryoların kapsamını yeterli bir şekilde temsil eder ve böylece veri sağlandığında ajan iyi bir şekilde eğitilebilir.

3.1.5 Araba-direk (CartPole) problemi

Araba-direk problemi Takviyeli Öğrenmede tipik bir başlangıç problemidir ve bu problemde, bir arabaya sabitlenmiş düz bir direğimiz vardır. Arabayı dengeleyerek

direğin düşmesini engellememiz gerekiyor. Bunu, arabayı ileri (diyelim ki +1 çıktı ile gösterilir) veya geri (-1 çıktı ile gösterilir) veya sabit kalarak (0 çıktı ile) hareket ettirerek başarabiliriz. Arabayı tek bir dönüşte her iki yönde de yalnızca belirli bir miktarda hareket ettirebileceğiniz ($\pm$ sabit bir diyelim 1) uygulamalar olabilirken, diğer uygulamalarda hızı da ayarlayabiliriz (sürekli + v'den -v'ye her birim zaman adımıyla) biraz daha karmaşık hale getirir.

Bu değişiklikler daha çok ajanın eylemiyle ilgilidir. Öte yandan bu aynı zamanda ajanın iyi bir eylemde bulunabilmesi için iyi bilgilendirilmiş olması gerekiyor ve bu durumu nasıl formüle etmeniz gerektiğini de etkileyecektir. Bazı ayrık durum uzayına sahip olan uygulamalar hareket etmeden yerinde kalma seçeneği de verebilir. Bu durumda 0 yerinde kalmak ve hiçbir şey yapmamak anlamına, -1 geri gitmek ve +1 ileri gitmek için kullanılabilir. Bazı diğer uygulamalar ikili bir eyleme sahip olurlar ve bu durumda 1 ileri gitmek ve 0 geri gitmek için kullanılabilir. Şekil 3.3, bir hat üzerinde yatay olarak hareket edebilen bir araba üzerinde direğin dengeli görüldüğü tipik bir araba-direği örneğini göstermektedir.



Şekil 3.3 Araba-direk problem örneği

Ayrıca, direğin dengelenmesi, sisteme beslenmesi gereken fizik, yerçekimi, momentum ve açısal hızların iyice öğrenilmesini gerektirir, ancak bu ayrıntılar, öğrenmesi doğrudan veya dolaylı olarak ajanın görevidir. Burada yapacağımız tek şey durumu gözlemlediğimiz şekliyle kodlamak ve bunları tasarladığımız ortamı beslemek için kullanmak olacaktır. Gözlemlediğimiz, sadece direğin arabaya göre -90° (direk saat

yönünün tersine düştü) ve $+90^\circ$ (kutup saat yönünde düştü) arasındaki açıdır, burada 0° mükemmel merkezli bir kutbu temsil eder.

3.1.6 Araba-direk (CartPole) problemi sürekli ortamda

Yukarıdaki bilgiler, bir miktar salınım kabul edilebilir olduğu durumlarda, ayrık bir eylem uzayına sahip ajanı eğitmek için yeterlidir. Ancak sürekli eylem uzayına sahip bir ajan için veya salınımı en aza indirmek için ve açısal hızı hesaba katmak için daha fazla bilgi gerekebilir. Açısal hız belirtildiği gibi ajanın öğrenmesi gereken bir parametredir fakat çoğu durumda sistem bu girdiye sahip olmayabilir. Ajanın bu ayrıntıları öğrenebileceği bir simülasyon ortamı, bu öğrenme işi için yararlı olabilir.

Bu örnekte açısal hız hakkındaki bilgileri içeren en basit örnek, mevcut zaman adımından bir birim zaman önce direğin konumu (açı olarak) olabilir (bir milisaniye veya bir tepki-salınım dengesi gibi anlamlı olan herhangi bir birim olarak düşünülebilir). Eğer ayrıca, bu birim zaman ajanın dönüşler arasındaki tepki vermesine gereken süre ise, tam olarak önceki durumumuz bu ek girdidir. Unutmayalım ki daha önce gördüğümüz örnekte önceki zaman adımında (veya durum farkı ile adım farkı eşitse önceki durumda) girdi olarak direğin açısal konumunu verdik. Bu nedenle, esasen, sürekli eyleme sahip bir ajanı eğitmek için (sadece mevcut durum/çısal konum yerine) sadece iki ardışık durumu göndermek yeterli oluyor (açısal konumun yanı sıra) ve böylelikle ayrık eyleme sahip ajanının aksine, diğer ilgili karmaşıklıkları dahil etmemize gerek kalmamaktadır.

Bu örnekte fark ettiğiniz gibi, işleri etkili ve ayrıca basit yapmak için ve takviyeli öğrenmede kullanacağımız ajanın durumunun matematiksel formülünü çıkarmak için yakalaması daha kolay ve kullanması daha basit olan gerçek dünyanın bazı simülasyonlarını kullanabiliriz. Ajan, doğru algoritmalar ve yeterli sürede yapılan eğitim deneyleri ile başarılı sonucu elde edem doğru hareketle ilgili fiziği dolaylı olarak öğrenebilir. Eğer söylenen koşullar altında yapılırsa, durum gerçek zamanlı gözlemlerin dönüşümü olmayabilir (tic-tac-toe aksine), ancak biraz ustalıkla sorun basitleştirilebilir. Bu yöntemin, ajanın verimliliğini ve doğruluğunu büyük oranda etkiler.

3.1.7 Takviyeli öğrenmede ajan

Ajan, herhangi bir durumda kararlar almak ve en uygun eylemi önermek için zekayı içerdiğinden, takviyeli öğrenmenin açık ara en önemli parçasıdır. Ajanın temel zekasını eğitmek amacıyla, karşılaşılabilecek benzer bir ortamın temsilleri yaratılır ve ajan için ortamı ve bağlamı uygun bir biçimde formüle edilir. Ajan çok önemli olduğu için, öğrenme mimarisi ve ilişkili modelleri üzerine çok sayıda araştırmalar yapılmıştır (Sewak vd. 2021).

3.1.7.1 Değer fonksiyonu

Ajan, her duruma geldiğinde o duruma göre yapabileceği en iyi eylemin ne olduğuna karar vermelidir. Bu kararı öğrenmesi için esasen iki yolu vardır. İlki, ajanın daha önce bu belirli durumdayken aldığı mevcut ve gelecekteki ödülleri kullanarak (geçmişteki tecrübelerinden yola çıkarak) sonraki en iyi durumun hangisi olduğunu belirlemeye çalışmaktır. Bu mantık benzer durumlara genişletebilir. Esasen tüm bu tekniklerde, gözlemlerimize dayanarak, herhangi bir görünen veya görünmeyen durumun (veya durum ve eylem birleşimini), “Değerini” (veya ödülünü) öngörmeye çalışıyoruz. Bu değer, mevcut durumla ilişkilendirilebilecek tüm şimdi aldığı mevcut ve gelecekteki (indirimli) ödüllerin bir fonksiyonu olabilir.

“(Durum) Değer Fonksiyonu” $V(s)$ ile gösterilir, burada s alt simgesi bu V ’nin (değer) bir durum fonksiyonu olduğunu gösterir. Bu değer fonksiyonu, bahsedilen çeşitli sorunları göz önünde bulundurur. Örneğin farklı ödülleri, ajanın öğrenmeye çalıştığı bir duruma bağlı homojen bir fonksiyona dönüştürerek gelecekteki/gecikmeli ve olasılıklı/belirsiz ödülleri yönetmek için ele alır. Burada temelde, ajanın kararı en karlı durum belirlenerek verilir ve yeni durumu geldiğinde tekrar aynı süreç tekrar edilir. Ajanın eğitim sürecinde asıl amacı, her bir durum için mümkün olan en yakın “Değerini” verebilecek “Değer Fonksiyonunu” öğrenmeye çalışır.

3.1.7.2 Eylem-değer/Q-fonksiyonu

Değer Fonksiyonu, daha önce bahsettiğimiz gibi, ajan tarafından içinde bulunulan en iyi duruma karar vermek için kullanılır ve daha sonra bu karara dayanarak o duruma ulaşma şansını en üst düzeye çıkarmak için uygun bir eylemde bulunur. Ancak bu, en iyi eyleme karar vermenin dolaylı bir yoludur, ya da belirli bir durumdayken mümkün olan en iyi eylemi belirleyerek doğrudan böyle bir karar alabiliriz. $Q(s, a)$ ile gösterilen “Eylem-Değer Fonksiyonu” bu durumda yardımcı olur. Gösterimi için kullanılan Q sembolünden ve ayrıca değer fonksiyonuna referans nedeniyle karışıklığı önlemek için, eylem-değer fonksiyonuna “ Q -Fonksiyonu” da denir. “Değer Fonksiyonu yalnızca durumun bir fonksiyonu olduğuna dikkat etmeliyiz, Q -Fonksiyonu hem durumun (s) hem de eylem a ’nın bir fonksiyonudur.

3.1.7.3 Sömürü ve keşif ikilemi

Eylem-Değer fonksiyonuna en uygun değeri eğitmek için, başlangıçta varsayılan değer olarak rastgele değerler veya belirli sabit değerler verebiliriz. Bu değerler eğitim sırasında çeşitli bölümler ve senaryolardan toplanan verilerle iyileştirilir.

Eğer ajan herhangi bir varsayılan değere veya herhangi bir aracı Eylem-Değer fonksiyonuna yatkın hale gelirse, kesinlikle bu şekilde geliştirilmiş Eylem-Değer fonksiyonu ya da başka bir deyişle önceden öğrenilmiş bilgiyi “Sömürü” yaparak kararlar alabilir, ancak yeni bilgileri sömürerek bu fonksiyonu daha fazla geliştiremez. Bu yüzden bir mekanizma yaratarak ajanın karar ölçütünü seçmesini sağlamamız gerekiyor. Bunun yapılabileceği temelde iki yaklaşım vardır, yani bir sonraki bölümde açıklayacağımız Politika İçi ve Politika Dışı yaklaşımlar.

3.1.7.4 Politika içi ve politika dışı yöntemler

“Politika” öğrenme mekanizmasının ve nihai modelin içinde olduğu duruma göre seçilmesi gereken en iyi eylemi bulmak için kullandığı taktike denir. Politika “ π ”

sembolü ile gösterilir. Politika, “ $\pi_{(s)}$ ” bir fonksiyon olarak ifade edilir ve durumu parametre olarak alır. İşlevi ise belirli bir durumda yapılacak en iyi eylemi belirlemektir. Oluşturulan politika tüm öğrenme süresince geçerlidir. Ancak dağıtım¹ sırasında, temelde keşif ve kullanım arasındaki dengeyi sağlamak yerine daha basit bir strateji yani sadece sömürüyü tercih edebiliriz.

“Q-Fonksiyonunu” öğrenmek için, her bir eylemin iyi olma (döndüreceği ödül) olasılığını hesaplayacağımız ve ardından hepsinin toplamını bire ölçeklendireceğimiz ve farklı eylemlerin ilgili ölçeklenmiş olasılıklarına dayalı olarak bir eylemi stokastik olarak seçeceğimiz bir yaklaşım benimseyebiliriz. Bu politikaya göre, ödül olasılığı en yüksek olan eylem aynı zamanda seçilme olasılığı en yüksek, diğerlerinin de benzer sırayla seçilme olasılığı daha düşüktür. Dolayısıyla, seçtiğimiz “Politika”, keşif ve sömürünün olasılıksak doğasına sahiptir çünkü zaten olasılıksal ortamda inşa edilmiştir. Bu yöntem kullanılırsa “Politika İçi” öğrenme denir.

Öte yandan, örneğin $\epsilon = 0.15$ gibi bir olasılık değeri ile, eylemleri seçmede sömürü yerine “keşif” edeceğimiz bir mekanizma ile öğrenebiliriz. “Keşif” aşamasında, basit olması adına, şimdilik eylemi rastgele seçtiğimizi varsayalım (mevcut durumda tüm olası eylemler için eşit olasılıkla). Ancak “keşif” sürecini ilerletmenin birçok çeşidi ve hatta daha iyi yolu da vardır.

Keşfetmediğimiz geri kalan durumlarda (yani, $(1 - \epsilon) = 0.8$ olasılıkla), en iyi “Q-değeri”ne sahip eylemi yapmak için “Sömürür” veya başka deyişle “Açgözlü” bir karar alırız. Bu “Q-değeri” ise farklı mekanizmalar/algoritmalar kullanılarak öğrenilebilir. Bu nedenle, iki ayrı fonksiyon kullanıyoruz, birisini “Q-fonksiyonu”muzun güncellemesi için ve diğerini ajanın “Davranışı” için kullanıyoruz. Sömürünün “Tahmin” politikasında gömülü olmadığı ve dolayısıyla bunun için ayrı bir politikanın gerekli olduğu bu yaklaşımlara “Politika Dışı” öğrenme yaklaşımı denir.

¹ Deployment

3.1.8 Markov karar süreci

Markov Karar Süreci¹ (MDP), tüm takviyeli öğrenme işleminin temel taşıdır. Başka deyişle şu ana kadar tartıştığımız her şeyin özeti MDP'dir diyebiliriz. MDP, iki bölümden oluşmaktadır: “Markov” ve “Karar Süreci”.

“Markov” terimi, MDP'nin bir formu olduğu “Markov Zinciri” olgusunun altında yatan ilke olan “Markov Özelliği”ni ifade eder. Markov Özelliği, olasılıksal ve belirsiz sistemler için “hafızasız” özellik olarak da adlandırılır. Eğer bir süreç Markov Özelliğini takip ediyorsa ve şimdi belirli bir durumdaysa ve ayrıca birkaç durumdan geçmişse, olası sonraki durumun koşullu olasılık dağılımı yalnızca mevcut duruma bağlı olacaktır. Bu, sürecin bu belirli mevcut duruma ulaşması için geçtiği durumların sıralamasından bağımsız olacaktır. Bu nedenle, belirli bir duruma ulaşmanın birkaç yolu (dizisi) olsa bile, sürecin belirli bir duruma ulaşması için hangi belirli yolu (dizi) benimsediği önemli değil, bu belirli durumdan sonraki durumların koşullu olasılık dağılımı aynı kalır.

Bir dizi olasılıksal olay Markov özelliğine bağlanırsa Markov Zinciri oluşur. Bir sonraki olayın olasılığının yalnızca önceki olayda elde edilen duruma dayandığı bir dizi olaydan oluşan stokastik bir modele atıfta bulunur. Bir Markov Zincirindeki olayların sırası, ayrık veya sürekli zamanda meydana gelebilir, ancak bunlar sayılabilir durum uzaylarından oluşur. Son bölümdeki açıklamadan ve başlangıçtan itibaren, durumları tanımlarken benzer bir veri yapısı olarak model izlediği ya da başka bir şekilde açık şekilde onu tek tip bir modele dönüştürürüz. Ajanımızı eğitmek için bazı uzunluklarda dizilerin gerekli olduğu sürekli araba direği örneği gibi. Bu örnek için, tek bir durumda biriktirilmiş ilgili sayıda önceki konumumuz vardı, öyle ki durum dizimiz kendi başlarına mevcut durum dışında herhangi bir önceki duruma bağlı değildir.

Markov Karar Süreci, ayrık durum uzayında kullanılan olasılıksal bir kontrol sürecidir. MDP, belirli bir karar sürecine Markov Zinciri özelliğini uygular. Takviyeli Öğrenme bağlamındaki karar süreci, ajanın herhangi bir durumdayken “Politika” yardımı ile

¹ Markov Decision Process

seçilecek en iyi eylemi belirlemesine yardımcı olur. Markov Karar yöntemi, sonuçları seçtiğimiz eylemlerden dolayı kısmen bizim kontrolümüzde olan ve kısmen rastgele eylemlerden oluşan süreci matematiksel modellemek için bir temel sağlar.

Markov karar sürecinin resmi modeli, tek ajanlı Takviyeli Öğrenme için kullanılır (Buşoniu ve diğerleri, 2010):

Tanım 1. $\langle X, U, f, \rho \rangle$ Demet¹, bir sonlu Markov karar sürecidir ve X 'in mümkün olan çevre durumlarının sonlu sayısını gösterir. U , sonlu ajan eylemleri kümesini belirtir $f: X \times U \times X \rightarrow [0, 1]$ durumların geçişi için olasılık fonksiyonudur ve $\rho: X \times U \times X \rightarrow \mathbb{R}$, ödül fonksiyonudur.

Ortam veya çevre, her ayrık k zaman adımında $x_k = X$ durumu ile tanımlanır. Ajan durumu gözlemler ve değerlendirir ve $u_k \in U$ eylemini gerçekleştirir. Sonuç olarak, f : tarafından verilen geçiş olasılıklarına göre çevre durumu $x_{k+1} \in X$ 'e değişir: x_k 'da u_k yürütüldükten sonra, x_{k+1} 'e varma olasılığı $f(x_k, u_k, x_{k+1})$. Ödül fonksiyonuna göre $\rho: r_{k+1} = \rho(x_k, u_k, x_{k+1})$, robot aracısı bir skaler ödül $r_{k+1} \in \mathbb{R}$ alır. Eylemin o anki etkisi u_k ödülle, yani çevre durumunun x_k 'den x_{k+1} 'e dönüştürülmesiyle değerlendirilir. Ancak, seçilen eylemin uzun vadeli sonuçlarının ne olacağı ile ilgili hiçbir şey söylemez. Varsayım ödül fonksiyonunun sınırlı olduğudur.

f veya geçiş olasılığı fonksiyonu, deterministik sistemler için daha basit bir geçiş işlevi olan $\bar{f}: X \times U \rightarrow X$ ile değiştirilir. Buna göre, ödül tam olarak yapılan eyleme ve mevcut duruma bağlıdır: $r_{k+1} = \bar{\rho}(x_k, u_k)$, $\bar{\rho}: X \times U \rightarrow \mathbb{R}$. Markov karar sürecinde birkaç terminal durum mevcuttur, bu durumlar girildikten sonra çıkılamayan durumlardır. Terminal durumun ödülü sıfır olur. Bu süreç boyunca, öğrenme süreci genellikle belli bir durumda başlayan ve bir terminal durumda biten yörüngeler olan farklı denemelere (bölümlere) bölünür.

¹ Tuple

Ajanın davranışını politikası belirler ve bu politika mevcutta olduğu durumu göz önüne alarak eylemlerin nasıl seçildiğini belirler. Politika belirleyici¹ $\bar{h}: X \rightarrow U$ veya olasılıksal², $h: X \times U \rightarrow [0, 1]$ olabilir. Zamanla değişmeyen politikalara durağan(stationary) denir. Ajanın amacı, herhangi bir x durumundan beklenen indirimli getiriye maksimize eden bir politika keşfetmektir:

$$R^h(x) = E \left\{ \sum_{k=0}^{\infty} \gamma^k r_{k+1} | x_0 = x, h \right\} \quad (3.1)$$

Burada h politikası altında, olasılıksal durum geçişleri üzerinden beklenti alınır ve $\gamma \in [0,1)$ indirim faktörüdür. Toplam uzun vadeli ödül ajan tarafından kompakt bir şekilde getiri R ile temsil edilir. İndirim faktörü γ , gelecekteki ödüller hakkında artan belirsizliği kodlamanın bir yolu veya aksi takdirde sınırsız büyüyecek bir toplamı sınırlamanın bir yolu olarak düşünülebilir.

3.1.9 Q-Öğrenme

Ajan uzun vadeli ödülü/başarıyı optimize etmeye çalışırken yalnızca tek adımlı performansı (ödül) hakkında geri bildirim alır. Bunu yapabilmek için Q-fonksiyonu veya başka deyişle ideal durum-eylem değeri fonksiyonunu hesaplayarak yapabilir. Herhangi bir durum-eylem çiftinden h politikası tarafından alınan beklenen getiri, Q-fonksiyonu $Q^h: X \times U \rightarrow \mathbb{R}$ tarafından verilir:

$$Q^h(x, u) = E \left\{ \sum_{k=0}^{\infty} \gamma^k r_{k+1} | x_0 = x, u_0 = u, h \right\} \quad (3.2)$$

Bellman denklemi, Q-Öğrenmenin temel ögesidir. Daha basit, özyinelemeli(recursive) alt problemlere bölerek ve onların en uygun çözümlerini bularak karmaşık bir problem için en uygun çözümü bulmaya yardımcı olur. Optimum Q-fonksiyonu $Q^*(x, u) = \max_h Q^h(x, u)$ ile gösterilir ve Bellman optimizasyon formülü bu formül kullanılarak hesaplanır

¹ Deterministic

² Stochastic

$$Q^*(x, u) = \sum_{x' \in X} f(x, u, x') \left[\rho(x, u, x') + \gamma \max_{u'} Q^*(x', u') \right] \quad \forall x \in X, u \in U \quad (3.3)$$

Denkleme göre, beklenen en yakın ödül artı gelecekteki indirimli ödül (bir sonraki durumdan elde edilebilecek en uygun değer), u 'yu x 'te almanın optimal değeridir. X sonludur ve bu yüzden beklenti bir toplam olarak ifade edilir.

Q^* belirlendikten sonra, her durumda en yüksek optimal Q değerine sahip eylem seçilerek optimal bir politika (yani maksimum getiriyi veren politika) hesaplanabilir:

$$\bar{h}^*(x) = \arg \max_u Q^*(x, u) \quad (3.4)$$

Birkaç eylem en yüksek Q değerine ulaştığında, aralarından herhangi bir politika seçilebilir. Bu durumda, 'arg' operatörü hem burada hem de devamında eşit derecede iyi çözümlerden sadece birini döndürür. Q -fonksiyonunu maksimize etmeye çalışan bu politika açgözlü politika sayılır. Sonuç olarak, optimal bir politika bulmak, önce Q^* 'ı hesaplamayı ve ardından açgözlü olarak Q^* 'da hesaplamayı gerektirir.

Denklem (3.3), etkileşimli bir yaklaşım prosedürüne dönüştürülür (Watkins vd. 1992a). Q -Öğrenme keyfi ve rastgele bir Q -fonksiyonu ile başlar, geçişleri $(x_k, u_k, x_{k+1}, r_{k+1})$ gözlemler ve her adımda Q -fonksiyonunu değiştirir.

$$Q_{k+1}(x_k, u_k) = Q_k(x_k, u_k) + \alpha_k \left(r_{k+1} + \gamma \max_{u'} Q_k(x_{k+1}, u') - Q_k(x_k, u_k) \right) \quad (3.5)$$

(x_k, u_k) 'nin optimal Q değeri ile mevcut $Q_k(x_k, u_k)$ tahmini ve güncellenmiş tahmin $r_{k+1} + \gamma \max_{u'} Q_k(x_{k+1}, u')$ arasındaki fark olan zamansal farkı (temporal difference) kullanır. Bu yeni tahmin, durum-eylem (x_k, u_k) çiftinde Q_k 'ye uygulanan Bellman denkleminin (2) sağ tarafının bir örneğine dayanmaktadır. Örneğimizde x' , gözlemlenen sonraki durum x_{k+1} ile değiştirilir ve $\rho(x_k, u_k, x')$ yerine gözlemlenen ödül r_{k+1} konur). $\alpha \in (0,1]$ olan öğrenme oranı genellikle zamanla değişir ve genellikle zamanla azalır.

(Jaakkola vd.1994, Tsitsiklis 1994. Christopher vd.1992a)'e göre, aşağıdaki koşullar altında, Q_k dizisi kanıtlanabilir şekilde Q^* 'a yakınsar:

İlk olarak, her durum-eylem çifti sonsuz, genellikle asimptotik olarak deneyimlenir. Daha sonra, her durum-eylem çifti için, Q -fonksiyonunun açık, farklı değerleri depolanır ve değiştirilir. Ek olarak, $\sum_{k=0}^{\infty} \alpha_k$ sonsuz iken, $\sum_{k=0}^{\infty} \alpha_k^2$ toplam ifadesi sonludur.

İlk ölçütün karşılanması için, diğer koşulların yanı sıra, ajan tüm sıfırdan büyük olasılığa sahip durumlarda, eylemlerin tamamını denemeli. Bu keşif olarak bilinir ve her adımda olasılık $\varepsilon \in (0,1)$ ve açgözlü bir eylem $(1 - \varepsilon)$ olasılıklı rastgele bir eylem seçilerek başarılabılır. Bunu yaparak üçüncü ölçüt olan ε -greedy keşif prosedürü elde edilir. Çoğu durumda, ε olasılığı zamanla azalır.

3.1.10 Çok ajanlı Q-öğrenme

Markov karar yöntemi, bazen stokastik oyun olarak adlandırılan çok etmenli duruma genelleştirilmiştir (Buşoniu vd. 2010).

Tanım 2. Bir $\langle X, U_1, \dots, U_n, f, \rho_1, \dots, \rho_n \rangle$ kümesi, sonlu bir çevresel durum kümesinin X ile gösterildiği stokastik bir oyundur. n , dahil olan ajanların sayısını belirtir. Ajanlar tarafından erişilebilen sonlu eylem kümeleri $U_i, i = 1, \dots, n$, ile gösterilir, ortak eylem kümesi $U = U_1 \times \dots \times U_n$, oluşturulur, durum geçiş olasılığı fonksiyonu $f : X \times U \times X \rightarrow [0,1]$, ve ajanlar için ödül fonksiyonları $\rho_i : X \times U \times X \rightarrow \mathbb{R}, i = 1, \dots, n$ olarak gösterilir.

Ödül işlevlerinin sınırlı olduğu varsayılır ve durum geçişleri, çok ajanlı bir sistem durumunda birlikte çalışan tüm ajanların sonucudur; yani, $u_k = [u_{1,k}^T, \dots, u_{n,k}^T]^T, u_k \in U, u_{i,k} \in U_i$ burada T vektör devriktir(transpose). $h_i : X \times U_i \rightarrow [0,1]$ politikaları, h ortak politikasını şekillendirmek için birleşir. Temsilcilerin $r_{i,k+1}$, ödülleri ortak eyleme bağlı olduğundan, sonuçları ortak politikaya bağlıdır:

$$R_i^h(x) = E \left\{ \sum_{k=0}^{\infty} \gamma^k r_{i,k+1} | x_0 = x, h \right\} \quad (3.6)$$

Her ajanın Q-fonksiyonu, $Q_i^h(x, u) = E \{ \sum_{k=0}^{\infty} \gamma^k r_{i,k+1} | x_0 = x, h \}$ ile birlikte $Q_i^h: X \times U \rightarrow \mathbb{R}$, ortak eylemi ve politikası tarafından belirlenir. Ortak stokastik oyunlarda tüm ajanlar için ödül işlevleri aynıdır: $\rho_i = \dots = \rho_n$. Dahası, hepsinin ortak getiriyi maksimize etme hedefi aynı ve çıktılar da aynı, $R_1^h = \dots = R_n^h$.

Q-Öğrenme algoritması algoritma 3.1’de verilmiştir:

Algoritma 3.1 Q-Öğrenme

Gerekenler:

- Durum kümesi $X = \{x_1, \dots, x_n\}$
- Durum kümesi $U = \{u_1, \dots, u_n\}$
- Ödül fonksiyonu $\rho: X \times U \rightarrow \mathbb{R}$
- Öğrenme oranı $\alpha \in (0,1]$
- İndirim faktörü $\gamma \in (0,1]$
- Keşif takası $\tau \in (0,1]$

```

1: Procedure Q-Öğrenme ( $X, U, \rho, \alpha, \gamma, \tau$ )
2:   Initialize  $Q: X \times U \rightarrow \mathbb{R}$ 
3:   While Yakınsanmamış do
4:      $x_k \in X$  durumunda başla
5:     While durum terminal değil do
6:        $\epsilon$ -açgözlü politikasını kullanarak  $\tau$  göre  $u_t \in U'$ ’yu seçin
7:        $u_k$  çalıştırın ve  $r_k$  ödülünü ve  $s_{k+1}$  yeni durumu gözlemleyin
8:        $Q_{k+1}(x_k, u_k) = Q_k(x_k, u_k) + \alpha_k(r_{k+1} + \gamma \max_{u'} Q_k(x_{k+1}, u') -$ 
9:          $Q_k(x_k, u_k))$ 
9:        $x_k = x_{k+1}$ 
10:    End while
11:  End while
12:  Return  $Q$ 
13: End procedure

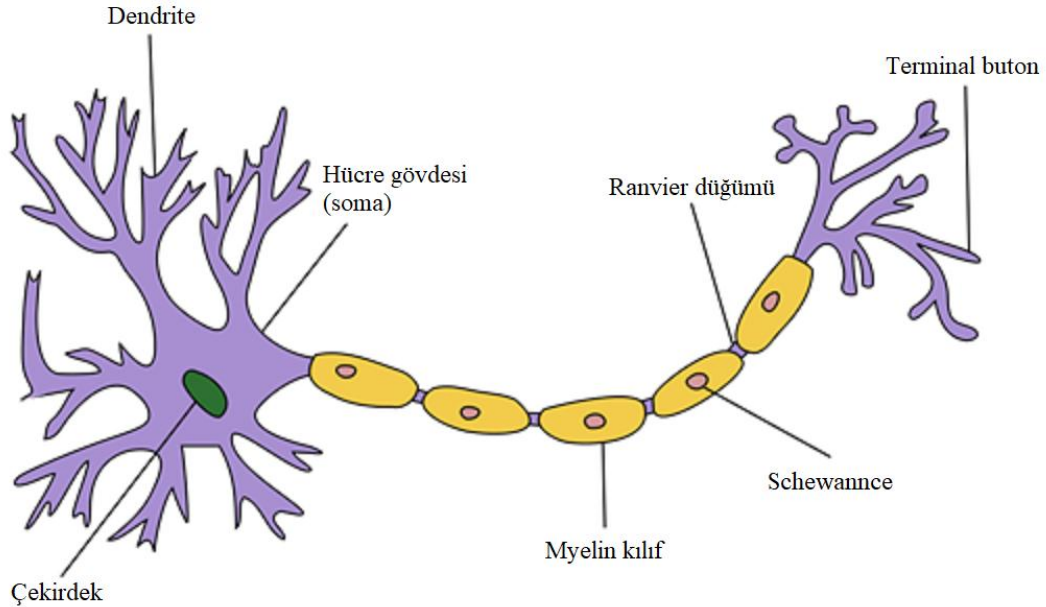
```

3.1.11 Derin öğrenmeye giriş

Bu bölümde Derin Öğrenmenin temelleri anlatılacak. Bunun için çok katmanlı algılayıcı¹ ve derin yapay sinir ağları² gibi derin öğrenme ağının temel mimarisi ve çalışma mekanizması anlatılacak. Ayrıca bu bölümde derin öğrenme ağlarından olan Evrişimli Sinir Ağları³ anlatılacak.

3.1.11.1 Yapay nöronlar

Yapay nöronlar derin öğrenmenin yapı taşları sayılırlar. Bir sinir ağı, birçok yapay nörondan oluşur. Yapay nöron en temel hali ile, gerçek bir biyolojik nörona benzer ve biyolojik nöronun çalışması ve oluşumundan esinlenerek modellenmiştir (Şekil 3.4). Biyolojik bir nöron, birçok diğer nörona ve beynin bölümlerine bağlıdır ve bu bölümlerden sinyaller alınır. Kendi sırasında bu girdilere dayalı olarak farklı çıktılar üretilir. Bu çıktı ise başka nöronları veya bazı kasları aktif etmek için kullanılır.



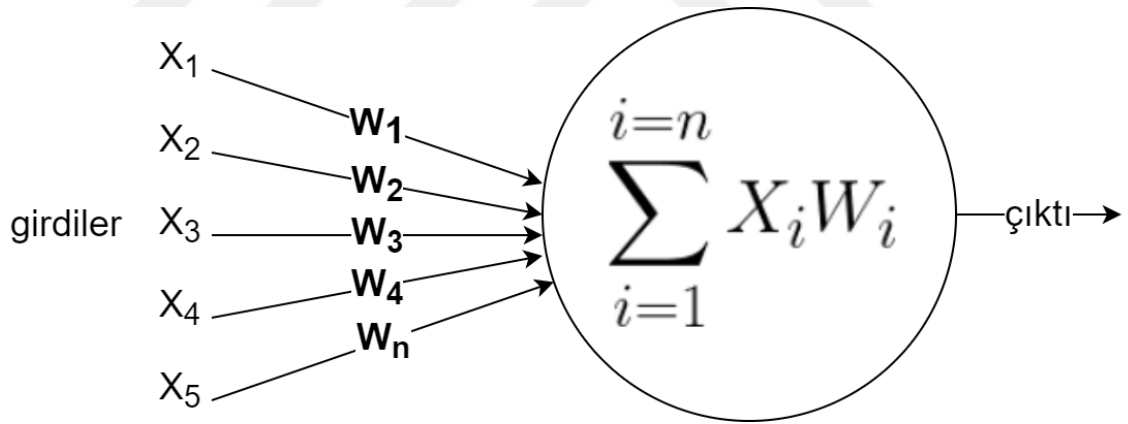
Şekil 3.4 Biyolojik Nöron (Wikipedia)

¹ Multi-layer Perceptron (MLP)

² Deep Neural Network (DNN)

³ Convolutional Neural Network (CNN)

Şekil 3.5 matematiksel olarak modellenen yapay bir nöronu gösterilmektedir. Her nöron için girdi olarak $x_1, x_2, \dots, x_n$ değerleri sağlanır. Girdiler, $X = \{x_1, x_2, \dots, x_n\}$ olacak şekilde veya sadece X vektörü olarak temsil edilebilir. Bu girdiler doğrudan ham verileri temsil edebilir, ancak etkili çalışması için uygun şekilde ölçeklendirilmelidir. Bazen bir toplu iş genelinde girdi özelliklerinin normalleştirilmesi gerekebilir (birim ortalamasına ve birim varyansına dönüştürülür). Girdiler veri yapısı olarak ağırlık vektörü şeklinde tanımlanır. Ağırlık vektörü ise $w_1, w_2, \dots, w_n$ öğelerine sahiptir. Ağırlıklar W vektörü olarak gösterilir, öyle ki $W = \{w_1, w_2, \dots, w_n\}$. En iyi sonucu elde etmek için ağırlık vektörünü güncellememiz veya başka deyişle eğitmemiz gerekiyor. Bu işlem için vektörün ağırlık elemanlarının optimal değerlerini ödülü en üst düzeye çıkaracak şekilde bulmamız gerekiyor. Bu işlemde en iyi çıktı, deneyerek elde edilen veri setindeki çeşitli örnekler arasında hata/kaybı (gerçek ve tahmin edilen değerler arasında) en az üreten çıktıdır. Gerçek çıktı y olarak ve nörondan tahmin edilen çıktı $\hat{y}$ olarak adlandırılacaktır. Ağırlıklara ek olarak, nöronun gerçek bir sayı olan bir önyargısı (örneğin b) olabilir.



Şekil 3.5 Yapay nöronun işleyişi

Nöronun içindeki işlem, “Aktivasyon Fonksiyonu” adı verilen bir fonksiyon kullanılarak yapılır. Aktivasyon fonksiyonunun girdisi, $w_1x_1 + w_2x_2 + \dots + w_nx_n + b(W^T X + b)$ ye eşit olan Z olarak adlandırılır. Buna ek olarak amaçlanan hedef ve işleve göre uygun bir aktivasyon fonksiyonuna karar verilmelidir. Nöronun çıktısı $\hat{y} = \text{Aktivasyon}(Z)$ olarak gösterilir. Aktivasyonu belirleyen önemli bir faktör de

eğitildikten sonra beklenen tahminlerin(çıktının) kategorik (sınıf olasılığı) veya sürekli olmasıdır.

Yukarıdaki tanım, ileri geçiş¹ olarak bilinir ve bu örnekte tek bir veri ögesi için yapıldı ama bir ağı ağırlıklarını eğitmek için daha fazla veri ögesine gerek var. Eğitim yaparken, değerlendirme sonrası oluşan veri seti üzerindeki kayıpları mümkün olduğu kadar en aza indirmeye çalışıyoruz. Kayıp miktarı, kayıp fonksiyonu kullanarak gerçek çıktı (y) ve tahmin edilen çıktı ($\hat{y}$) parametreleri ile hesaplanır. Aktivasyon fonksiyonunda olduğu gibi, aralarından seçim yapabileceğimiz birden fazla kayıp fonksiyonu vardır ve bunların arasından uygulamaya ve girdi verilerine en uygun olanı seçilebilir.

3.1.11.2 İleri beslemeli derin sinir ağları (DNN)²

Yapay bir nöron nadiren tek başına kullanılır. Beyinde olduğu gibi, nöronlar bireysel olarak çok güçlü değildir, ancak milyonlarca nöron birleştiğinde, beynin tüm gücünü sağlıyorlar. Benzer şekilde, yapay bir nöronu tek başına kullanmak yerine, birleşimleri bir Yapay Sinir Ağı (YSA) oluşturmak için kullanılabilir. Bir yapay sinir ağı katmanları üç türdür, ilk katmana giriş katmanı, daha sonrakilere gizli katman ve en son katmana da çıkış katmanı denir. Giriş katmanı sistemin özellik vektöründeki özellik kadar nörona sahiptir. Çıkış katmanı da aynı şekilde, tasarımda düşünülen çıktı sayısı kadar özellik ve nörona sahiptir. Sınıflandırma probleminde çıkış sayısı ağdan tahmin edilecek sınıf sayısı kadar olabilirken, regresyon probleminde çıkış sayısı bir olabilir. Gizli katmandaki nöron sayısı, eldeki uygulamaya göre değiştirilebilir ve bu sayı için en uygun değere ulaşmak biraz deneyim gerektirebilir.

YSA'ların evrensel tahmin edici olduğu bilinmektedir. Uygun bir aktivasyon fonksiyonu verildiğinde, bu özellik gerçek sayılı³ özelliklere sahip kompakt bir veri alt kümesindeki herhangi bir sürekli fonksiyonu yaklaşık olarak tahmin edebileceği

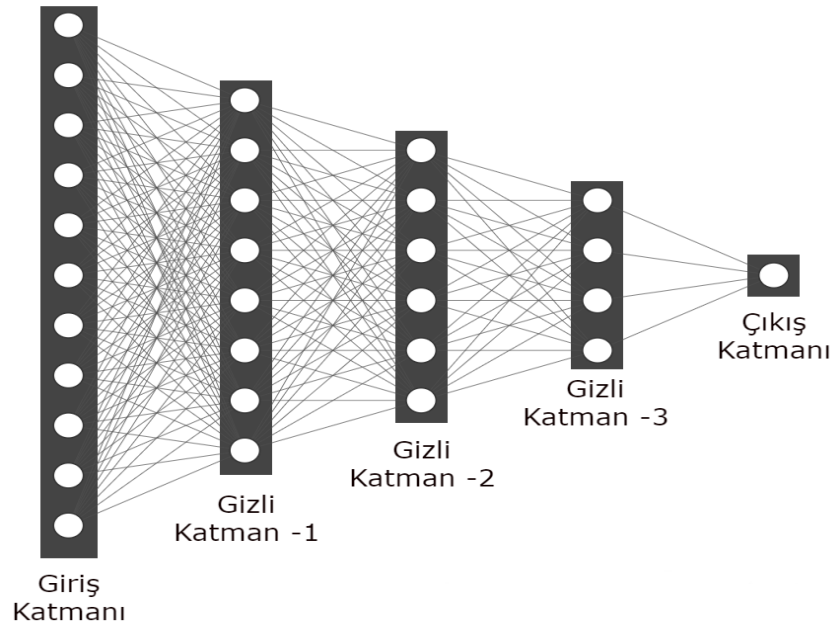
¹ Forward pass

² Feed-Forward Deep Neural Networks (DNN)

³ Real number

anlamına gelir. Ancak bu tahmin edici özelliğe rağmen, tek bir katmanla öğrenilen bir fonksiyon, karmaşık gerçek hayat verilerini modelleyemez. Derin Sinir Ağlarına (DNN) burada ihtiyaç duyuluyor. Bunlar bazen Çok Katmanlı Algılayıcı (MLP) tabanlı derin sinir ağı (MLP-DNN) olarak da adlandırılır; burada MLP terimi, bu ağlardaki altta yatan yapay nöronun türünü belirtir.

Derin Sinir Ağları (DNN'ler) birden fazla gizli nöron katmanına sahiptir (Şekil 3.6). Bir süre öncesine kadar, çok derin bir sinir ağı oluşturmak zordu. Bu, sinir ağlarında “Kaybolan Gardiyan¹” adı verilen bir fenomenden kaynaklanıyordu. Derin öğrenmede her katmanın ideal ağırlıklarını bulmak(eğitmek) için, son katmanın ağırlığından başlayarak her katmanın ağırlıkları güncellenir ve bu işlem katman katman geriye doğru yayılır. Bu katmanların ağırlıkları, kayıp fonksiyonunun² gradyanı alınarak güncellenir, böylece ağırlıklar, kaybın en aza indirilebileceği bir yönde iyileştirme sağlanır. Kayıp fonksiyonunda bir minimuma hareket edilerek kayıplar minimize edilebilir ve bu minimuma kayıp fonksiyonlarının bu şekilde hesaplanan gradyanları yönünde hareket edilerek ulaşılabilir.



Şekil 3.6 Derin öğrenme ağı

¹ Vanishing Gradient

² Loss function

Bu eğitim yöntemine “Geri Yayılım” denir. Hatalar, gardiyanları bulmak ve dolayısıyla ağı optimize etmek için ağda geri yayılır. Türevlerin zincir kuralına göre geri yayılım mümkündür.

Yani $(n - 2)$ 'inci katman ile son katman (n 'inci katman) arasındaki gradiyan, $(n - 1)$ 'inci katman ile (n) 'inci katman arasındaki gradiyanın çarpımından gelir. Hatayı derin bir ağda çıktı katmanından başlangıç katmanlarına yaymak, uzun gradiyan çarpımları zinciri gerektirir. *Sigmoid* ve *Tanh* gibi optimal olmayan bir aktivasyon fonksiyonunun kullanılması, sonuçtaki gradiyanın küçük bir mutlak değerine ($\ll 1$) dönüşmesine yol açar; bu, zincir kuralında olduğu gibi bir dizi benzer şekilde küçük mutlak değerlerle sürekli olarak çarpıldığında kaybolan gradiyan eğilimine yol açar. Buna kaybolan Gradiyan etkisi denir.

Geoffrey Hinton, kaybolan gradiyan probleminin üstesinden gelmek için bir sürü aktivasyon fonksiyonu yöntemi önerdi ve bunların yardımı ile sinir ağlarını derin yapmayı mümkün kıldı. Bu sebepten dolayı Derin Öğrenmenin babası olarak da biliniyor. Derin bir sinir ağı, çeşitli katmanları sayesinde birden fazla işlevi öğrenebilir. Bu katmanların birleşiminin sayesinde çok güçlü bir model oluşabilir. Derin Sinir ağları çok karmaşık yapılandırılmış veriler veya konuşma ve görme gibi yapılandırılmamış verilerle çalışabilir. Yapılandırılmış verilerle çalışmak ve tahmin yapmak için örnek bir derin sinir ağı Şekil 3.6'da gösterilmektedir. Bu ağ 3 gizli katmana sahiptir.

3.1.11.3 Derin sinir ağlarında ileri besleme mekanizması

İleri Beslemeli Derin Sinir Ağındaki aktivasyon da bir önceki bölümde anlatılan tek nöron aktivasyonuna benzerdir. İleri beslemeli bir ağda, sinyal bir katmandaki nöronlardan sonraki katmandaki nöronlara yalnızca ileri yönde yayılır. Bir DNN'nin gizli katmanında, belirli bir katmandaki tüm nöronların girdileri, önceki katmandaki tüm nöronlara (yoğun veya tam bağlı katmanlar olarak da bilinir) bağlıdır ve çıktıları, sonraki katmanın nöronlarının her birine bağlıdır. Girdi katmanındaki nöronlar, bire bir eşleme ile girdi verilerindeki özelliklere bağlanır; çıktı katmanı ise regresyon veya ikili sınıflandırma için bir çıktı özelliğine bağlanır ve çok terimli sınıflandırma için sınıf

sayısı kadar özelliğe bağlanır. Çıkış katmanının aktivasyon fonksiyonu Giriş ve gizli katmanlardan farklı olur.

Bir DNN’de, belirli bir katmandaki tüm nöronlar farklı ağırlıklara sahip olabilir ve bu nedenle, her katmandaki tüm nöronların tüm ağırlıklarının eğitilmesi gerekir, bu nedenle eğitim sürecini hesaplama açısından maliyetli hale getirir. Evrişim Sinir Ağı ile ilgili daha sonraki bir bölümde, ağırlıkların belirli bir katmandaki nöronlar arasında paylaşılabileceği bir mekanizma hakkında bilgi edineceğiz, böylece özellikle aşağıdaki gibi uzamsal olarak ilişkili girdi verileriyle çalışmak için tasarlanmış bir ağ için eğitimin hesaplama karmaşıklığını azaltacağız.

3.1.12 Derin öğrenmede mimari

Derin öğrenme katmanların sayısı ve türü mimariyi belirler. Bu da Derin Öğrenme ağının türüne ve özel kullanımına bağlıdır. Bu bölümde, bu tür ağların çeşitleri ve aralarında paylaşılan ortak noktalar tartışılacaktır.

3.1.12.1 Aktivasyon fonksiyonları

Bir önceki bölümde tartışıldığı gibi, nöronun ağırlıklı girdisini çıkışına dönüştürmek için nöronlar bir aktivasyon fonksiyonuna kullanırlar. Ayrıca Derin Sinir Ağı bölümünde, aktivasyon fonksiyonundaki değişikliğin Derin Öğrenmenin başarısına nasıl yol açtığını ve sinir ağlarının ölçeklenmesine nasıl yardımcı olduğunu tartıştık. Bu bölümde, sık kullanılan bazı aktivasyon fonksiyonlarında bahsedeceğiz.

SoftMax Aktivasyonu

Derin öğrenme ağı sınıflandırmak için kullanılacaksa çıkış katmanının nöron sayısı verilerdeki sınıf sayısı kadar olmalıdır. Sadece bir nöron gerektiren ve Sigmoid Aktivasyonu kullanabilen ikili sınıflandırma dışında, çok terimli sınıflandırma tabanlı derin öğrenme ağlarının çoğu çıktı katmanında SoftMax Aktivasyonu kullanır.

İncelenen her sınıf için girdinin o sınıftan olma olasılığı SoftMax Aktivasyon katmanı kullanılarak elde edilir (Goodfellow, Bengio ve Courville, 2016).

$$P(y = c|x) = \frac{e^{x^T w_j}}{\sum_{k=1}^K e^{x^T w_k}} \quad (3.7)$$

Doğrusal ve Identity Aktivasyonu

Sınıf olasılıkları gerektiren sınıflandırma problemlerinden farklı olarak, regresyon problemleri çıktı katmanında sadece tek bir nörona sahip olabilir. Bu ağlar en başta eğitim data setinde sağlanan gerçek çıktıdaki değişkenin tahminine karşılık gelen bir değeri çıktı olarak verebilir. Bahsedilen bu değer genellikle kullanılmadan önce ölçeklendirilir ve mutlak bir temsil kullanılmaz. Takviyeli Öğrenme Doğrusal aktivasyonunun kullanılması gereken durumlardan birisidir. Identity aktivasyonu denklem (3.7)'de verildiği gibidir. Doğrusal etkinleştirme işlevini elde etmek için normal Kimlik işlevine ölçekleme ve önyargı¹ faktörünü eklememiz yeterli olacaktır (Sewak ve diğerleri, 2021).

$$Identity(x) = \sum_i x_i w_i \quad (3.8)$$

Doğrultucu Doğrusal Birim² (ReLU) ve Varyantları

Daha önceki bölümde çıktı katmanının aktivasyon yöntemlerini tartıştık bu kısımda da girdi ve gizli katmanların aktivasyon fonksiyonlarını tartışacağız. Kaybolan gradyan problemi tartışılmıştı, bu sorunu çözmek için daha önce, literatürde Geoffrey Hinton tarafından önerilen Doğrultucu Doğrusal Birim veya ReLU kullanılır. Bu fonksiyon denklem (3.9)'te verildiği gibidir. Daha sonra, Leaky ReLU denklem (3.10), Randomize Leaky ReLU ve Parametriized Leaky ReLU gibi bazı varyantlarında diğer araştırmacılar tarafından önerilmiştir. ReLU aktivasyonu çok yaygındır ve uygulamaların çoğu, giriş/gizli katmanlarda bu fonksiyonu kullanır.

¹ Bias

² Rectifier Linear Unit

$$ReLU(x) = \max(0, x) \quad (3.9)$$

$$Leaky ReLU(x) = \begin{cases} ax & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases} \quad (3.10)$$

Üstel Doğrusal Birim¹ (ELU)

ReLU aktivasyonu, kaybolan gradiyan problemini ve aktif olmayan veya ölü nöronlarla ilgili diğer bazı problemleri çözse de kendisinin “Ortalama Kayma”² olarak adlandırılan bir problemi vardır. Derin öğrenme ağ eğitim sırasında yaşanan bir sorunun sebebi ReLU’daki aktivasyonların sıfır merkezli olmamalarından kaynaklanıyor bu da ReLU’nun pozitif çıktılara eğimli olmasından kaynaklanıyor. Çözüm olarak “Toplu Normalleştirme” yöntemi katmanlar arasına uygulanabilir. "Ortalama kayma" sorununu yükseltmenin bir yolu, giriş/gizli katmanlarda ReLU aktivasyonunu Üstel Doğrusal Birim (ELU) aktivasyonu ile değiştirmektir. Denklem (3.11)’de ELU verilmiştir:

$$ELU(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha (\exp(x) - 1) & \text{if } x \leq 0 \end{cases} \quad (3.11)$$

3.1.12.2 Derin öğrenmede kayıp fonksiyonları³

Bir ağı eğitirken, her katmandaki ağırlıkları/hiperparametreleri optimize etmek için hatayı geri yaymamız gerekir. Bu kaybı hesaplamak için kayıp fonksiyonu kullanılır, bu fonksiyon kayıp değerini hesaplamak için gerçek çıktı ve tahmin edilen değerleri girdisi olarak alır ve çıktı olarak kayıp miktarını verir. Kayıp fonksiyonları genellikle, L1 kaybı (gerçek ve tahmin edilen değer arasındaki mutlak farka dayalı olarak) ve L2 kaybı (gerçek ve tahmin edilen değerler arasındaki farkın karesine dayalı olarak) olmak üzere iki kategoriye ayrılır. L2 tabanlı kayıp fonksiyonlarının optimize edilmesi matematiksel olarak kolaydır ve ayrıca bunlar sürekli türevlenebilir fonksiyonlar. Öte yandan L1 tabanlı kayıp fonksiyonları daha sağlamdır ve ters olan veya aykırı değerlerden fazla etkilemez.

¹ Exponential Linear Unit (ELU)

² Mean Shift

³ Loss functions

Gerçek tahmin edilen değerler (L1 ve L2 kayıpları) arasındaki mutlak fark ve kare fark ile ilgili farkın yanı sıra, hesaplamadaki diğer bazı değişiklikler farklı senaryo türlerinde yardımcı olabilir. L1 ve L2 varyantları ve ayrıca diğer tür kayıp fonksiyonları denklem (3.12) - (3.18)'de verilmiştir. Sınıflandırma problemleri için çapraz entropi kaybı ve regresyon problemleri için L2 (Mean Square Error) —MSE çok popülerdir.

$$L2 \text{ loss: } \|y - o\|^2 \quad (3.12)$$

$$\text{Beklenti Kayıp Fonksiyonu: } \|y - p(o)\| \quad (3.13)$$

$$\text{Düzenli Beklenti Kayıp Fonksiyonu: } \|y - \sigma(o)\|^2 \quad (3.14)$$

$$\text{Hinge Kayıp Fonksiyonu: } \sum_j \max(0, 1/2 - \hat{y}_j o_j) \quad (3.15)$$

$$\text{Kare Hinge Kayıp Fonksiyonu: } \sum_j \max(0, 1/2 - \hat{y}_j o_j)^2 \quad (3.16)$$

$$\text{Çapraz Entropi Kayıp Fonksiyonu: } - \sum_j y_j \log \sigma(o)_j \quad (3.17)$$

$$\text{Kare Log Kayıp Fonksiyonu: } - \sum_j [y_j \log \sigma(o)_j]^2 \quad (3.18)$$

3.1.12.3 Derin öğrenmede optimize ediciler

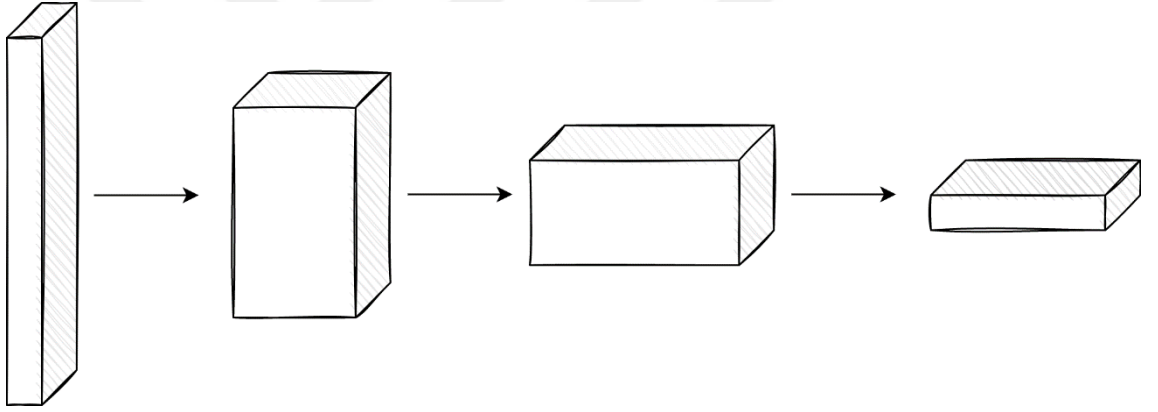
Derin öğrenmede kayıp fonksiyonlarının rolü, kaybı optimize eden/en aza indiren farklı hiperparametrelerin ideal kombinasyonunu seçmektir. Bu teorik olarak kayıp fonksiyonunun gradiyanını bularak yapılır ve eğer kayıp fonksiyonu doğada dışbükey ise (orijine doğru kavisli bir minimuma sahipse) o zaman ağırlıkları kayıp fonksiyonunun gradiyanına doğru güncellemek bizi bir minimuma götürecektir.

Ancak pratikte bu, derin öğrenmede bazı makine öğrenimi problemleriyle karşılaştırıldığında o kadar basit değildir. Buna sebep olan ana şeyler, hiperparametre veya parametre alanındaki yüksek sayısı ve optimize edilmesi gereken fonksiyonun karmaşıklığıdır. Gradiyan hiperuzayda eğitim alırken, yerel bir minimumda takılıp

kalmak veya optimizasyon sırasında vadiler¹ ve eyerlerle² karşılaşmak gibi çeşitli sorunlarla karşılaşılır. Optimize edici bu sorunlara karşı sağlam ve ayrıca verimli ve hızlı olmalıdır. Derin öğrenmede yaygın kullanılan optimize edicilerden ikisi RMSProp ve ADAM'dir (Uyarlamalı Moment Tahmini).

3.1.13 Evrişimli sinir ağları—Vizyon için derin öğrenme

Bir Evrişimsel Sinir Ağı (CNN) veya ConvNet, çok katmanlı derin sinir ağının çok özel bir türüdür. CNN, görsel kalıpları doğrudan görüntülerden çıkarmak için tasarlanmıştır ve bunun için minimum işlem gücünü kullanır. Bu ağın görseli şekil 3.7'te gösterilmektedir.



Şekil 3.7 Bir CNN'nin grafiksel gösterimi

Bir özellik/öznitelik vektörü olarak temsil edilebilecek yapılandırılmış verilerin aksine, görüntü genellikle 3 boyutlu bir gerçekte sayı matrisidir. Bir görüntüdeki genişlik, yükseklik ve renk kanalları (derinlik) bu üç boyutla gösterilir. Giriş matrisinin her bir elemanı, resmin o konumdaki görüntü pikseline karşılık gelen yoğunluğu gösterir.

Özellikle daha yüksek çözünürlüklü görüntüler, karmaşık bir veri yapısını temsil eder. Yüksek boyutlu bir veriyi modellemek için daha önce ele alınana ileri beslemeli DNN'i kullanmaya çalışırsak, çok büyük ağırlık matrisi eğitimi ve birçok katman gerektiren

¹ ravines

² saddles

çok derin bir sinir ağı lazım olur. Bu da kendi sırasında görüntü işlemedeki hatayı artırır ve hesaplamaları çok karmaşık bir görev haline getirir. Neyse ki, görüntüler uzamsal olarak ilişkili verileri temsil eder. Bu, bir görüntüde birlikte bulunan piksellerin yoğunluk değerlerinin çok benzer (ilişkili) olduğu ve bunlar arasındaki bilgilerin kademeli olarak değiştiği anlamına gelir. Bu özellik, derin öğrenme ağının görüntüler için işleme yükünü azaltabilir. Bu, bir Evrişimsel Sinir Ağı'nın Çok Katmanlı Algılayıcı (MLP) tabanlı Derin Sinir Ağı (MLP-DNN) muadiline üstün olduğu yerdir.

CNN'ler, MLP-DNN'ler gibi bir girdi katmanına sahiptir. Ancak bu vektör tek boyutlu değildir ve görüntünün boyutlarını taklit eder. Bu nedenle, 3 boyutlu bir görüntü için CNN girişi 4 boyutlu bir tensördür (çok boyutlu bir dizi), burada tensörün 3 boyutu görüntünün 3 boyutunu temsil eder ve her boyut görüntününkiyle aynı kardinaliteye sahiptir (genişlik x yükseklik x renk kanalları) ve 4. boyut, mini toplu¹/toplu ortamda eğitim için görüntü örneğini temsil eder. Görüntüler, özellikle yüksek çözünürlüklü olanlar, genellikle boyutu küçültmek için yeniden ölçeklendirilir ve renk kanalları modelle uyumluluk için ayarlanır (veya küçültülür).

Giriş katmanı ile çıkış katmanı arasında kaç farklı türde gizli evrişim sinir ağı katmanı vardır. Bunlar evrişim katmanı², havuzlama katmanı³ ve tam bağlantılı katmandır⁴. Bir CNN ağı, farklı sırada düzenlenmiş bu katman türlerinden birden fazlasına sahip olabilir. Evrişimsel ve havuzlama katmanları çoğunlukla aralıktır ve ardından tam bağlantılı katmanlar gelir. Bu, son olan çıkış katmanına kadar devam eder.

3.1.13.1 Evrişimsel katman

CNN'deki Evrişimsel terimi, matematikteki evrişim işlevinden sonra adlandırılan Evrişimsel katmanın kullanımına atıfta bulunur. Evrişim, matematiksel bir işlemdir ve girdi olarak iki fonksiyon alır ve üçüncü bir fonksiyonu oluşturur. Aslında kıvrımlı

¹ Batch

² Convolution layer

³ Pooling layer

⁴ Fully connected layer

fonksiyon, orijinal fonksiyonlardan birinin kıvrılmış halidir. Bu işlemin sonucunda oluşan fonksiyon, girdi olarak verilen fonksiyonların noktasal çarpımının integralini verir. Sonuç, orijinal fonksiyonlardan birinin çevrildiği büyüklüğün¹ bir fonksiyonudur. Evrişim katmanları aslında çapraz korelasyonlar kullanır. Aslında bu işlem evrişim işlemine çok benziyor.

Evrişim katmanının çıktısına özellik haritası ismi verilir. Bu çıktı çekirdek filtresi tarafından verilir ve bir dizi şeklindedir. Oluşturulan özellik haritaları, bir sonraki katmana girdi olarak kullanılabilir. Unutmamalıyız ki çekirdek filtre sayısı her evrişim katmanı için farklı olabilir.

3.1.13.2 Havuzlama katmanı

Görüntü verileri yüksek oranda ilişkilidir ve verimli hesaplama için maksimum bilgiyi koruyacak şekilde sıkıştırılmalıdır. Bu aynı zamanda her bir evrişim katmanından üretilen evrişim haritalarının her biri için de geçerlidir. Sonraki herhangi bir evrişim katmanına girdi boyutunu azaltmak, hesaplama yükünü önemli ölçüde azaltabilir, tam olarak havuzlama katmanının yardımcı olduğu yer burasıdır.

Genellikle evrişim katmanları sonrasında havuzlama katmanları yerleştirilir. Bir havuzlama katmanı, önceki evrişim katmanından oluşturulan evrişim haritasının $m \times n$ boyutlu alt bölgesine uzanır ve ardından tüm haritayı kapsayacak şekilde evrişim haritasının genişliği ve derinliği boyunca ilerler. Daha sonra, bir maksimum havuzlama veya bir ortalama havuzlama tekniği kullanılarak her adımdan tek bir temsili değer seçilir, böylece sonraki evrişim katmanlarının hesaplama karmaşıklığını azaltmak için çıktı boyutu küçültülür. Evrişimsel haritada, her giriş katmanında bağımsız bir havuzlama katmanı çalışır. Havuzlama katmanının çıktı ve girdisinin derinliğinin aynı olma sebebi de budur. Fakat bu katmandan sonra Evrişimsel haritanın yüksekliği ve genişliği aynı değildir.

¹ Magnitude

3.1.13.3 Düzleştirilmiş ve tamamen bağlı katmanlar¹

Şimdiye kadar, tartıştığımız gibi, giriş katmanı ve bundan sonraki tüm katmanlar, MLP-DNN'dekiyle aynı yapay nörondan oluşsa da yapı olarak MLP-DNN'den bileşim ve çalışma açısından büyük ölçüde farklıydı. Resimlerde olduğu gibi, uzamsal olarak ilişkili verilerden işe yarayan özellikler çıkarmak için, performanslı bir yapıya ihtiyaç duymamız bunun sebebidir. Yapılandırılmış bir veriyi normal bir MLP-DNN'e benzer olarak girişi tek boyutlu bir veri tensörü/vektöre çevirebiliriz. Ancak bunu yapabilmek için giriş verilerini MLP-DNN'lerdeki gibi yapılandırılmış bir formatta vermeleri lazım. Bu görev düzleştirilmiş katmana aittir. Düzleştirilmiş katmanın nöron sayısı, CNN/havuzlamadaki çok boyutlu katmanın nöron sayısı ile eşittir. Bu katman vasıtası ile bir önceki katmanın yapısını çevirmek için bire bir eşleme yapılır.

Bir veya daha fazla tam bağlantılı katman (FC katmanı) dizisi düzeltme katmanına bağlı olur. Bu katmanlar MLP-DNN'deki gizli katmanlar mantığında çalışır ve nöronlar tek tek biri önceki katmanındaki her nöronla eşleşir. Ayrıca herhangi bir FC katmanındaki her bir nöronun çıkışı, yanındaki katmandaki tüm nöronların girdilerine bağlıdır. Bu nedenden dolayı tam bağlantılı katman ismi verilir. Son tam bağlı katman da çıktı katmanına yönlendirilir. Birçok durumda çıktı katmanı, girdisi görüntü olan verilerindeki sınıflarla aynı sayıda düğüme sahip olur. Bunlar SoftMax aktivasyon sınıflandırma katmanı kullanır. DQN ve takviyeli öğrenme, son FC katmanı, bir fonksiyonun değerini tahmin eden doğrusal olarak etkinleştirilmiş tek bir çıkış düğümüne bağlanabilir.

Çıktı olarak farklı tahminlere ihtiyaç olursa düzeltme katmanlarının ardından gelen tam bağlı katmanlar çıktı sayısı kadar farklı ağa bölünebilir ancak bunlardan önceki katmanlardaki temel bilgiler paylaşılabilir. Bu şekilde oluşturulan iki FC ağının her biri, kendi bireysel çıktı katmanında sahip olur.

¹ Flattened and Fully Connected Layers

3.2 Problem

Robotun referans çerçevesine göre kendi pozisyonunu ve oryantasyonunu oluşturma kabiliyetine robot lokalizasyonu denir. Çevresinde gezinme yeteneği tüm mobil cihazlar için, önemlidir. Çarpışmalar ve güvensiz koşullar (sıcaklık, radyasyon, havaya maruz kalma vb.) gibi tehlikeli durumlardan kaçınmak önce gelir, ancak robotun ortamındaki belirli yerlerle ilgili bir amacı varsa, önce o yerleri bulması gerekir. Robot gezinimi, yol planlama yeteneği anlamına gelir. Bu işlemi robot kendi referans çerçevesine göre konum bularak ve daha sonra bir hedefe yönünü belirleyerek yapar. Bunlara ek olarak tanımlandığı çevrede gezinmek için hareketlilik cihazı veya bu örnekte olduğu gibi robot, bir çevre haritasına ve bu haritayı yorumlama yeteneğine sahip olmalıdır.

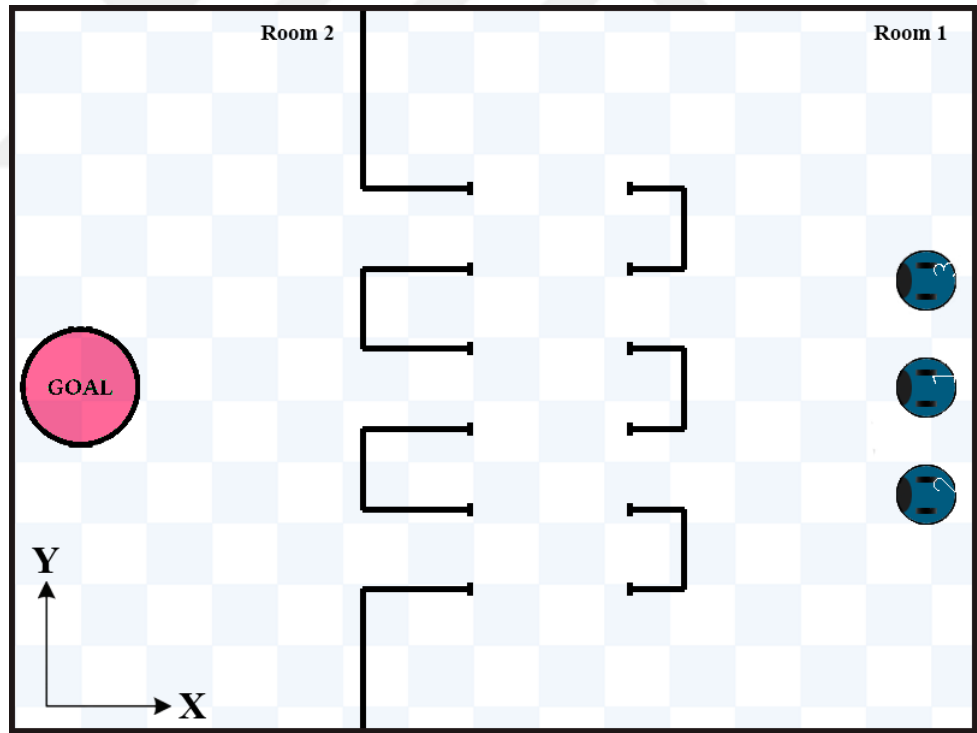


Şekil 3.8 Gezinme

3.2.1 Senaryo tanımı

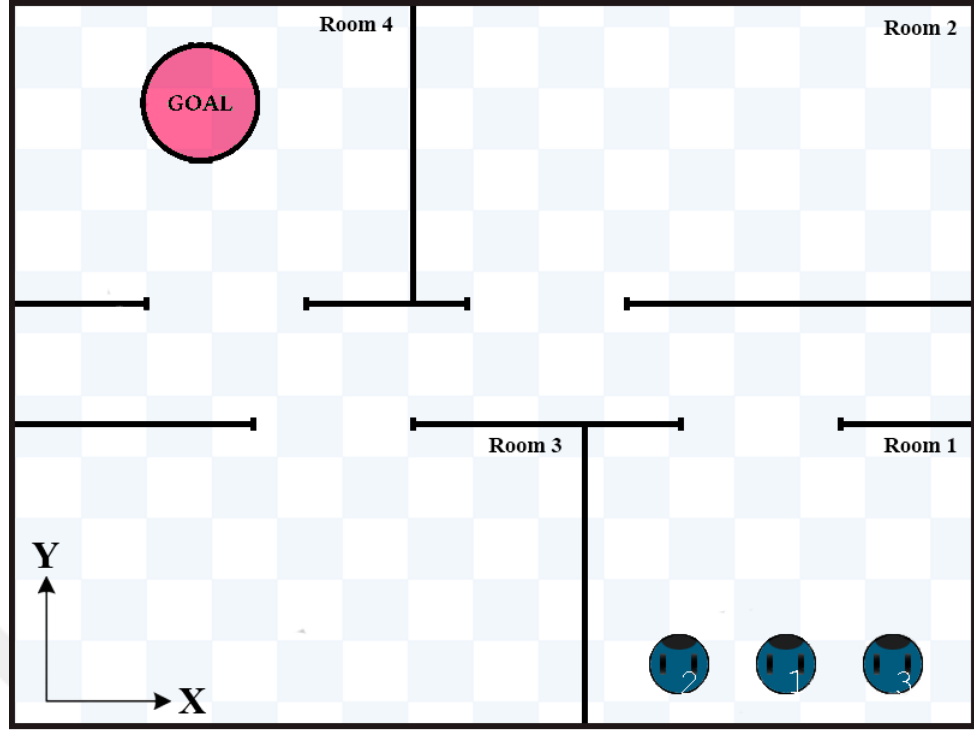
Şekil 3.9a, Şekil 3.9b ve Şekil 3.9c, örnek senaryolarımızdan üçünü göstermektedir. Bu senaryolarda, robotlar başlangıç noktasından hedef noktasına kısa bir yoldan gitmeleri beklenmektedir. Kullanılan robotlar daireseldir ve ajanların eylem alanı “sağa dön”, “sola dön”, “ileri doğru adım at” ve “hiçbir şey yapma” aksiyonlarından oluşur. Bu şekilde, bir robotun belirli bir noktaya gitmesi için önce sola dönüş veya sağa dönüş

eylemlerinden birini seçerek noktaya doğru dönmesi gerekir ve daha sonra ileri adım seçerek hedefe doğru adım atabilir. Ortam engelleri, görüntüsündeki çizgiler ile temsil edilir. Çarpışan robotlar (dinamik veya statik engelle) başarısız sayılarak o anki öğrenme oturumunun kapanmasına ve büyük bir ceza/negatif ödül almasına sebep olacaktır. Hedef noktasına ulaşan robotlar başarılı sayılacak ve büyük bir pozitif ödül alacak ve hedefe varan son robot değilse diğer robotların da gelmesini bekleyecektir. Bölümün sona ermesi için tüm robotlar hedef noktasına ulaşmalı veya geri kalan diğer robotlar çevrede uzun süre maksimum adım sayılarını aşacak kadar dolaşmaları gerekiyor. Robotlar attıkları her adımda küçük bir negatif ödül alırlar. Bu tür bir ödülü vermenin sebebi robotları hedefe mümkün olduğu kadar kısa yoldan ulaşmaya teşvik etmektir. Tasarlanan denetleyici sistem, tüm robotlara ayrı ayrı yerleştirilmiş ve herhangi bir merkezi sisteme bağlı olmadan her bir robot için kısa yoldan çarpışmasız bir yol planlıyor.

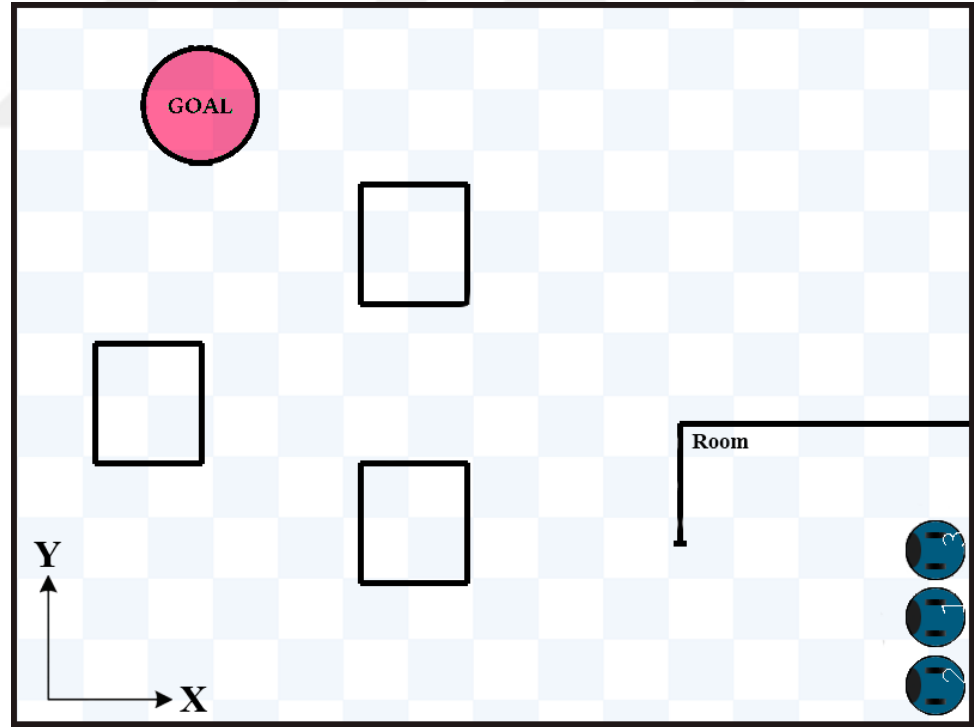


(a)

Şekil 3.9 Test senaryoları



(b)



(c)

Şekil 3.9 Test senaryoları (devam)

(a) dağınık U-şekilli engeller içeren bir senaryo; (b) bir binanın iç mekân haritasına benzeyen bir senaryo.
(c) bir oda ve dağınık engeller içeren bir senaryo.

Bu tez çalışmasının literatüre ve bilimie temel katkısı , bellekteki durum uzayı boyutunun sıkıştırılmasıdır. Durum uzayı, sistemin tüm olası durumlarının kümesidir; başka bir deyişle, değişkenlerle temsil edilen ya da bazen özellikler (features) olarak adlandırılan matematiksel bir modeldir. Bu özellikler benzetim ortamına ve kontrolörümüz için tasarlamak istediğimiz davranışlara göre tanımlanmalıdır. Özellikler, denetleyici için ihtiyacımız olan tüm bilgileri kapsamalıdır. Başka bir deyişle, her durumda bir eylem yaparak sorunun çözülebileceği yeterli özelliğe sahip olduğumuzdan emin olmalıyız. Örneğin, senaryomuza ve ortamımıza göre denetleyicimizin engellerle çarpışmasını önlemek istediğimizi varsayalım. Bu durumda, özniteliklerimize nesnenin gerçek konumunu veya göreceli konumunu dahil etmeliyiz veya koordinat özellikleri ekleyerek denetleyiciye bir konum hissi vererek çarpışmaları ele alabiliriz.

3.2.2 Normal durum uzaylı denetleyicisi (NDUD)

Durum uzayını tasarlarken bir durum için birden fazla eyleme sahip olmak, ekstra bir özellik eklememiz gerektiği anlamına gelir. Detaylandırmak adına, diyelim ki sadece tek robot ve hedeften oluşan bir ortamımız var. Bir kontrol sisteminin robotu ortamdaki herhangi bir noktadan hedef noktasına götürmesi için sadece x , y konum özelliklerini tanıtması yeterli gibi görünebilir. Aslında bu yöntem her adımda tüm yönlere gidebilen robotlar için yararlı olabilir, ancak tanımladığımız problemdeki robotlar belli bir noktaya gidebilmeleri için önce sola veya sağa dönmeli ve yüzlerini hedefe doğru çevirmenin ardından ileri gitmeleri gerekir. Anlatılan bu sistemde sadece iki x , y özelliği doğru eylemi bulmak için yeterli olmayacaktır; çünkü kontrol sistemi robotun mevcut yönü hakkında bir bilgiye sahip değildir ve bir duruma birden fazla eylem koymaya çalışacaktır. Dolayısıyla bunu yapmak için doğru yöntem robota yön özelliğinin eklenmesi olacaktır. Bu durumda ilk yöntemden farklı olarak robot sola dönerse kendini yeni bir durumda bulur ve farklı bir eylem seçebilir.

Ortam ve davranış özelliklerine göre, daha önce bahsettiğimiz senaryolar için standart bir durum uzayı dokuz özelliğe sahip olmalıdır.

- Aktif robot ile hedef noktası arasındaki mesafe (d_g): Bu, kontrol sisteminin bir robotu hedefe doğru yönlendirebilmesi için temel veridir.
- Hedef noktasının aktif robota göre göreceli açısı (a_g): Göreceli hedef açısı, aktif robotun merkezini ve hedef noktasını geçen çizginin açısıdır ve $\arctan2(\nabla y, \nabla x)$ formülü ile hesaplanır. Hedef noktasının açısı olmadan robot, göreceli konum duygusuna sahip olmayacak ve bu nedenle hedefe giden bir yol bulamayacaktır.
- Robotun açısı (a_r): Robotun sola/sağa dönmesi gerekip gerekmediğine karar verebilmesi için kendi yönü hakkında bilgi sahibi olması gerekir.
- Çarpışma özelliği (c_r): Bu özellik çarpışma için ilgili değeri tutar ve iki durumu olabilir.
- Hedef özelliği (g_r): Bu özellik, robotun hedef durumuna ulaşip ulaşmadığını iki değer ile tutar.
- Diğer iki robotun bilgileri: Bu özellikler eklenen her ekstra robot için tekrarlanır.
 - o i 'inci robota olan mesafe (d_{r_i}): Diğer robotlarla çarpışmayı önlemek için aktif bir robotun o robota olan mesafesini bilmesi gerekir.
 - o i 'inci robotun açısı (a_{r_i}): Diğer robotun mesafesine ek olarak, aktif bir robotun diğer robota göre göreceli açığa ihtiyacı vardır. Aksi takdirde aktif robot çarpışmalardan kaçınamayacaktır.

Bu özniteliklerle ilgili ilk sorun, durum özellikleri dışında kalan özniteliklerin sürekli olması ve Q-tablosunda kullanılabilmesi için ayrıklaştırılmalarının gerekmesidir. Eğitim bittikten sonra bir aracı kontrol sistemi kullanarak ayrık değer fonksiyonunu genelleştirmek ve düzgün hareketler şeklinde sürekli hale getirmek mümkündür; bu bir şekilde denetleyicinin daha önce hiç görmediği değerleri tahmin edebildiği denetimli öğrenme gibidir. Ancak, bu çalışmanın odak noktası bu değildir. Bu çalışmanın ele almak istediği bir yön, kısmen dağınık bir ortamda çok ajanlı robot navigasyonunun sürekli durum uzayı problemini çözmek için ayrıklaştırılmış ve daha sonra sıkıştırılmış bir model önermektir. Yukarıdaki özellikler verilen problemi çözebilir ve deneyler bölümünde göreceğimiz gibi önerilen model, bir dizi deney gerçekleştirilerek doğrulanmıştır.

3.3 Yöntem

Özniteliklerin ayrıklaştırılması, sürekli özelliklerin ayırık aralıklara dönüştürülmesi veya bölümlenmesi sürecini ifade eder, öte yandan hassasiyette bir dereceye kadar kayıpla gelir. Başka bir deyişle, önceki ortamdaki bir dizi değer, artık sadece tek değerle, sonlu ayırık bir dizide tutulacaktır. Esas olarak ayrıklaştırma miktarı bilgisayar Birincil hafıza boyutu ile sınırlıdır. Böylece ayrıklaştırma işleminde öncelikle özellikleri önem derecesine göre önceliklendirdik; daha sonra deneme yanılma yöntemi ile hangilerinin daha fazla ayrıklaştırılmaya ve daha fazla belleğe ihtiyaç duyduğuna kara verdik. Buna göre en yüksekten en düşüğe ayrıklaştırma miktarı sırasıyla hedefe olan mesafe, hedefe olan açıya, diğer robotlardan uzaklığa, aktif robotun açısına ve diğer robotlara olan açıya verilmiştir.

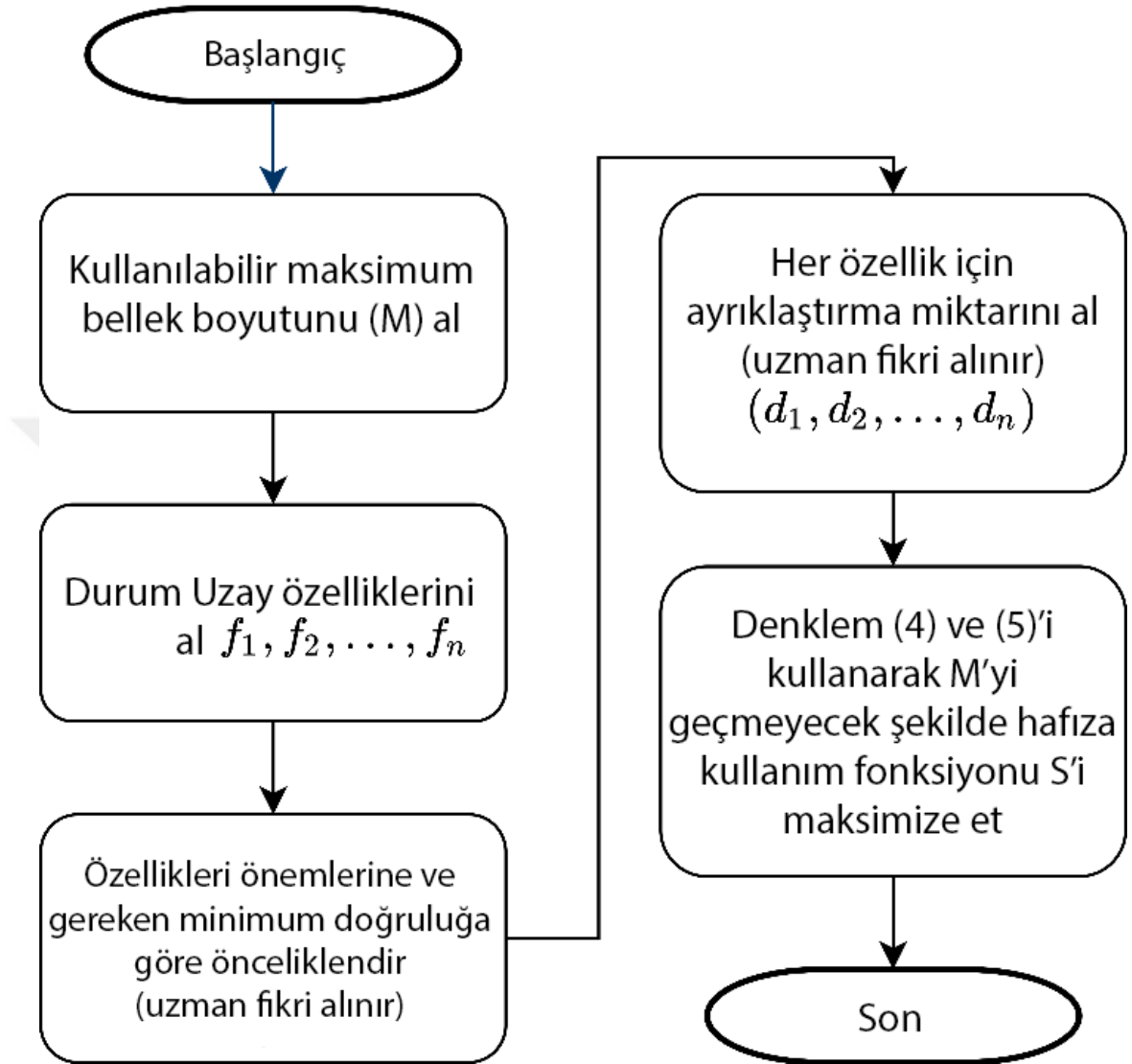
Örneğin, hedefe olan mesafenin maksimum değeri 950cm (ortamın hipotenüsü ile robotun çapının farkına eşittir) ve robotun açısının maksimum değeri 360 derece olabilir. Testlerimizde, hedefe olan mesafe özelliğini on sekiz parçaya ve robot açısını 8 parçaya bölerek sırasıyla adım boyutlarını 52 ve 45'e çıkardık. Bunun tersini yapsaydık, hedefe olan mesafe adım boyutu yaklaşık 119cm olurdu ve bu problemin çözümü için uygun olmazdı ve bazı durumlarda robotlar engelleri geçemezdi. Buna göre, deneme yanılmalar yapılarak, sırasıyla hedefe olan mesafe, hedefe olan açıya, diğer robotlardan uzaklığa, aktif robotun açısına ve diğer robotlara olan açıya en yüksekten en düşüğe ayrıklaştırma miktarı sağlanmış oldu.

$$S(d_1, \dots, d_n) = p.c. \prod_{i=1}^n d_i \quad (3.19)$$

$$(d_1^*, \dots, d_n^*) = \underset{(d_1^*, \dots, d_n^*) \in \mathbb{N}^+}{\operatorname{argmax}} S(d_1^*, \dots, d_n^*) = \{d_1, \dots, d_n | S(d_1, \dots, d_n) \leq M\} \quad (3.20)$$

Ayrıklaştırma işleminin daha resmi tanımı denklem (3.19), (3.20)'te ve şekil 3.10'teki akış şemasında verilmiştir. Q-tablo boyutu, denklem (3.19) kullanılarak hesaplanır. Bu denklemde S , Q-tablo boyut fonksiyonudur, $d_1, \dots, d_n$, durum uzayındaki her özelliğin ayrıklaştırma miktarını temsil eder, p , Q-tablosunda kullanılan veri tipi boyutudur ve c , ortamda tanımlanan toplam eylem sayısıdır. Amaç, denklem (3.20)'ü kullanarak M

maksimum bellek boyutunun altında kalırken ayırıklaştırma miktarını maksimize etmektir.



Şekil 3.10 Özellik ayırıklaştırma akış şeması

3.3.1 SDUD için model özelliklerinin tanımlanması

Daha önce belirtildiği gibi, verilen özellikler, sürekli bir uzay problemde, geniş bir durum uzayı oluşturmaktadır. Ayırıklaştırmadan sonra, boyutunu küçültmenin bir yolunu ararken, örtüşmeyen durumları istiflemenin ve durum uzayı boyutunu küçültmenin bir yolu bu tez çalışması kapsamında geliştirilmiştir. Ek olarak, bazı özelliklerin durum sayısı azaltılmıştır. Burada, örtüşmeyen durumlar ile aynı anda

oluşamayacak durumlar ifade edilmiştir. Örneğin, önerilen senaryoda, çarpışma ve hedefe varma durumları örtüşmez ve aynı anda gerçekleşemez. Buna ek olarak, problemimiz için, çevrede çarpışmanın meydana gelebileceği birçok farklı pozisyon olmasına rağmen, problemin durum uzayında sadece bir çarpışma durumuna ihtiyacı olduğunu biliyoruz. Bunun nedeni, senaryomuz için çarpışma durumları arasında ayırtırmaya ihtiyaç duymamamızdır. Çünkü aynı olumsuz ödüle sahipler, buna ek olarak çarpışmanın konumu bu problem için önemli değildir. Tüm bu nedenler hedefe varma özelliği için de geçerlidir ve o yüzden tek durum bu özellik için de yeterlidir.

Bu nedenleri göz önünde bulundurarak, her özellik için yeni bir değişken eklemek yerine, bazılarını üst üste istifleyebileceğimizi ve bu şekilde durum alanı boyutunu önemli ölçüde azaltabileceğimizi yeni bir yaklaşım bu tez çalışmasında önerilmiştir. Ek olarak, bazı özellikler için özellik ayırıklaştırma miktarı yalnızca bire düşürülebilir. Bu, yöntem ona verilen ortamın durum sayısını azaltmaz ve sadece ortam için kullanılan bellek miktarını azaltır. Bu, Q-Öğrenmede çok kritik bir kazanımdır. Çünkü bu yöntemin ana sorunu, durum uzayına özellikler eklendiğinde veya ayırıklaştırma miktarları arttığında polinom, kuadratik ve hatta daha hızlı büyüyen Q-Tablosu boyutudur. Deneylerimizde, her bir ekstra özelliği ekleyerek veya ayırıklaştırma miktarını artırarak Q-Tablosunun boyutunun megabaytlardan gigabaytlara hatta terabaytlara çoğaldığını ve doldurulmayı veya kaydedilmeyi neredeyse imkânsız hale getirdiğini gözlemledik.

Özellik yığılmanın en önemli amacı kullanılan bellek boyutunu azaltmasıdır. Bunun nedeni, Q-tablo boyutunun her bir özelliğin ayırıklaştırma miktarının çarpımı ile tanımlanmayacağı ve bunun yerine yığılmış özelliklerin toplamının çarpımının sonucu olacağıdır. Örneğin, bir ortamda, ajan için olası “4” eylem tanımlandığını ve sırasıyla “10”, “10” ve “2” parçalara ayrılmış “A”, “B” ve “C” olmak üzere yalnızca üç özelliğimiz olduğunu varsayalım. NDUD kullanan bu kurulumda “Q-tablo”su boyutu, “10×10×2×4” çarpımı ile elde edilir ve bellekte “800” hücre boyutunda hafızaya ihtiyaç duyar. Ancak, SDUD öznitelik yığılmasının kullanılmasına karar verilirse, Q-tablosunun boyutu “10×12×4” çarpımı ile elde edilen “480” durum ile hafızayı işgal

edecektir. Sonuç olarak, SDUD algoritmasının kullanılması bu örnek için bellek boyutunu (bellek maliyetini) %40 oranında azaltmayı başarmaktadır.

3.3.2 Sıkıştırılmış durum uzaylı denetleyici (SDUD)

Bir önceki bölümde söylenenleri göz önünde bulundurarak özellikleri analiz ettik ve örtüşmeyen üç "Hedefe Uzaklık", "Hedefe ulaşıldı" ve "Çarpışma" özelliklerini bulduk. Bu özelliklere aynı anda ihtiyaç yoktur, yani robot çarpıştıktan sonra hedefe olan mesafeye ve hedefe ulaşıldı özelliği değerlerine ihtiyacımız olmayacaktır. Bu nedenle bu özellikler üst üste istiflenebilir. Özellik istifleme dediğimizde, gerçekten özellikleri üst üste koyma kastedilmektedir. Pratikte, birkaç özellik için aynı değişken farklı aralıklar için kullanacaktır. Bunu yapmak için, Q-Tabloda özellik değerlerini biriktiren bir adaptör birimi yapılabilir.

Bu durumda yeni durum alanında özellik istifledikten sonra, yedi yerine beş özellik kalmıştır. İlgili özellikler aşağıda belirtilmiştir.

- $(d_g - s)$ Hedefe olan mesafe, çarpışma ve hedefe ulaşıldı özellikleri aynı anda tutacak Mesafe-Durum çifti,
- Hedef noktasının aktif robota göre göreceli açısı (a_g) ,
- Robotun açısı (a_r) ,
- Diğer iki robotun bilgileri, bu özellikler eklenen her ekstra robot için tekrarlanır.
 - o i 'inci robota olan mesafe (d_{r_i}) ,
 - o i 'inci robotun açısı (a_{r_i})

Bu yeni özelliklerin ihtiyaç duyulan bellek boyutunu önemli ölçüde azaltmasına rağmen öğrenme sürecini ve süresini etkilemediğine dikkat etmeliyiz. Esasında bunun nedeni, durum uzayının gerçekte değişmemesidir. Eğitimin tamamlanabilmesi için tüm çarpışma durumları keşfedilmelidir. Deneyler bölümünde sunulan öğrenme grafikleri bu iddiayı doğrulamaktadır.

3.3.3 Genelleştirilmiş SDUD

Bir diğer önemli problem, Q-Öğrenmeyi bir robot sürüsüne uygulamaktır. Burada, önerilen yöntemi kullanmaya devam edersek, yeni robot başına iki ekstra özellik eklenmesi gerekmektedir. Bu durumda Q-Öğrenme sürü robot üzerinde kullanılamaz, hatta “sıkıştırılmış durum uzaylı denetleyicisi” (SDUD) kullanılsa bile başarılı sonuçlar elde edilemez. Bunun sebebi, eklenen her ajanın beraberinde durum-eylem uzayına getirdiği özelliklerdir. Çok ajanlı takviye öğrenmede ajan sayısı artınca algoritmanın karmaşıklığı da katlanarak büyüyür. Sonuç olarak, iyi bilinen boyutluluk laneti¹ çok ajanlı takviyeli öğrenmede, tek ajanlıya kıyasta daha şiddetlidir (Buşoniu ve diğerleri, 2010).

Bu sorunu çözmek için, SDUD kullanarak herhangi bir sayıda robota sahip sürü gezinimini mümkün kılmak için başka bir yöntem geliştirilmiştir. Bunu yapabilmek için Q-Öğrenme algoritması için sağlanan algısal verileri ikiye sınırlandırılmıştır. Sınırlamak için en başta algısal verileri önceliklendirmek gereklidir. Önceliklendirme süreci, aktif robotun etrafındaki robotlardan hangi iki robot konumunun SDUD tarafından önemseneceğini belirler. Şekil 3.11, mavi ile gösterilen merkezdeki aktif robotun önceliklerini göstermektedir.

¹ Dimensionality curse



Şekil 3.11 Aktif robot ve etrafındaki robotların önceliği

Aktif robot üzerindeki kontrol algoritması, sadece iki komşu robotu görerek karar verebilecek; bu yüzden onları önem derecesine göre önceliklendirme yapmak zorundadır. Önem derecesi, aslında çarpışmaya neden olma olasılığı ile ölçülür. Robotlarımızın hareket mekanizmasına göre, temel olarak öndeki robotlar, aktif robotun arkasındaki robotlardan ve hatta yandaki robotlardan daha önemlidir. Sonraki önceliklendirme ölçütü aktif robot ve diğer robotlar arasındaki uzaklıktır. Aktif robotun arkasındaki robotlar, yakın olsalar bile önünde başka robot olduğu sürece tamamen atlanır. Çünkü aktif robotun kısa vadede yapacağı hiçbir işlem arkasındaki robotlarla çarpışmasına yol açamaz. Buna karşılık gelen adımların sözde kodu¹ Algoritma 3.2’de verilmiştir.

Algoritma 3.2 Önceliklendirme algoritması

Gerekenler: $[(d_{r_1}, a_{r_1}), \dots, (d_{r_n}, a_{r_n})]$ as UA (mesafe-açı tüm dinamik engellerin uzaklı-açı çifti)

Amaç: kontrolöre iletilecek en önemli iki engeli seçmek,

- 1: **Procedure** Prioritize Robots (DA)
- 2: $A = \text{Sırala}(DA, \text{"ön/arkaya göre"})$
- 3: $B = \text{Sırala}(A, \text{"uzaklığa göre"})$
- 4: $C = \text{DatayaAçıÖnceliğiEkle}(B)$

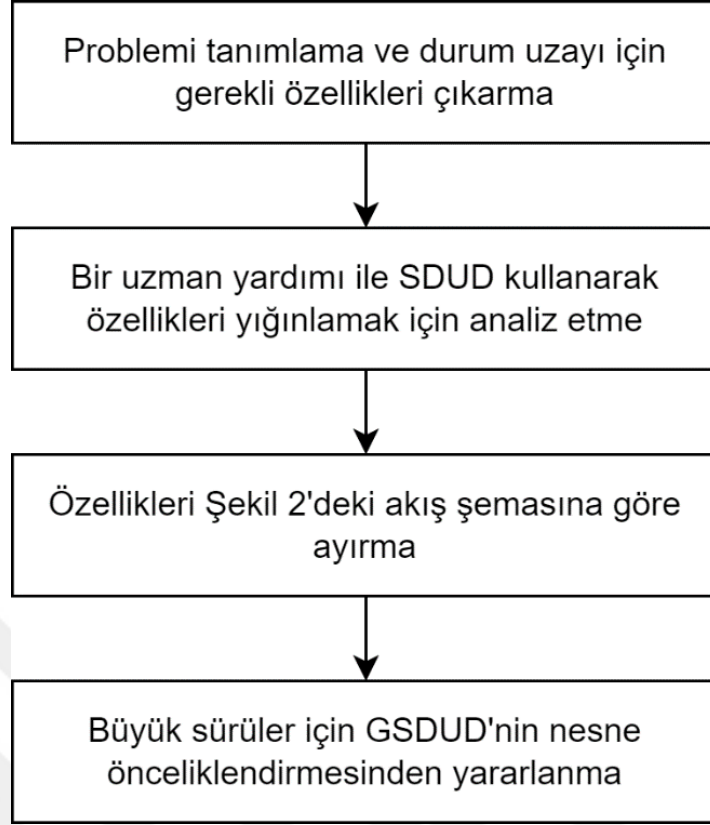
¹ Pseudocode

```
5:  $D = \text{Sırala}(C, \text{"açı önceliğine göre"})$   
6:  $E = \text{EnÜsttenEngeliSeç}(D)$   
7: Return  $E$   
13: End procedure
```

Önceliklendirme algoritmasının girdisi, dinamik engellerin mesafe-açı listesinden oluşur. Bu liste, şekil 3.11’de gösterilen tüm robotlar için bir mesafe-açı özellik çifti içerir. İlk olarak, algoritma giriş listesini engellerin aktif robotun ön veya arkasında olma durumuna göre sıralar, yani arkadaki robotlar listenin sonuna gönderilir. Bu sayede öndeki robotlar, arka sıradakilere göre çok yakın olmasalar da daha fazla tercih edilecektir. Ardından engeller aktif robota olan uzaklıklarına göre sıralanırlar. Bunu takiben, bir fonksiyon kullanarak yeni sıralanmış listeye açı öncelikli boyut eklenir. Açı önceliklendirmesi, şekil 3.11’e göre yapılır. Bunun ardından son aşamada eklenen yeni boyuta göre bir sıralama daha yapılır. Sonuç olarak, algoritma en üstteki iki engelin uzaklık-açı çiftini döndürür.

3.3.4 Bir bakışta genel akış

Şekil 3.12, bu çalışmada önerilen yöntemin genel akışını göstermektedir. Bir GSDUD sistemi tasarlamamanın ilk adımı, problemi tanımlamak ve özelliklerini çıkarmak için özelliklerini detaylandırmaktır. Bir sonraki adım, bir uzmanla istifleenebilir olanları bulmak ve bu özellikleri birleştirmek için onları analiz etmektir. Bundan sonra, şekil 4’teki akış şemasında belirtildiği gibi özelliklerin mevcut belleğe göre ayrıklaştırılması gerekmektedir. Buraya kadar tezde tasarlanan sistem, NDUD’ye kıyasta birçok sorunu daha düşük bellek tüketerek çözebilecektir. Üç robot yeterli değilse bir adım daha ileri giderek, çok ajanlı bir gezinim sisteminde herhangi bir sayıda ajanı işlemeyi sağlamak için GSDUD kullanılabilir.



Şekil 3.12 Önerilen yöntemin akışı bir bakışta

4. DENEYLER

Önerilen yöntemler, uygulanabilirliklerini, etkinliklerini ve optimizasyon oranlarını göstermek için kapsamlı bir şekilde test edilmiştir. Bu bölümde deney sonuçlarını paylaşacağız. Deneyler, Windows işletim sisteminde 16 GB RAM belleğe sahip 2,8 GHz hızında çalışan Intel Dört çekirdekli bir makine ile gerçekleştirilmiştir. Sistem python programlama dili ve Pygame (Shinners, 2011) kullanılarak tasarlanmıştır. Pygame, video oyunları ve simülatörler yazmak için tasarlanmış, bilgisayar grafikleri ve ses kitaplıklarını kapsayan, platformlar arası bir Python modülü setidir.

Bu deney setlerinde, önceki bölümde bahsedilen yöntemler üç farklı test senaryosu altında test edilmiştir. Şekil 1a, 1b ve 1c’de senaryoların başlangıç durumlarını göstermektedir. Ayrıca, önceki bölümde bahsedilen üç farklı denetleyici tasarlanmıştır: “Normal durum uzaylı denetleyici” (NDUD), “Sıkıştırılmış durum uzaylı denetleyici” (SDUD) ve “Genelleştirilmiş SDUD” (GSDUD). Yapılan testlerde İlk olarak NDUD ve SDUD yöntemlerinin bellek ihtiyaçlarını karşılaştırdık. Daha sonra, gezinim problemlerini çözmedeki başarı ve öğrenme hızı açısından her iki yöntemin etkinliğini karşılaştırdık. Daha sonra çoklu robot gezinimi için aynı senaryolar üzerinde SDUD’yi engel önceliklendirme algoritması ile birlikte kullandık (GSDUD).

Q-Öğrenmeyi sürekli durum uzayına sahip bir problem üzerinde kullanmak için, durum uzayının ayrıklaştırılması gerekir. Ayrıklaştırma işleminde, ortamı daha fazla parçaya ayırmak daha yumuşak hareketlere sebep olsa ve daha güvenilir olsa da bunun için bir sınır ve engel vardır. Durum uzayını daha fazla ayrıklaştırılırsa, öğrenme daha fazla sürecek ve daha da önemlisi, ortaya çıkan model daha fazla belleğe ihtiyaç duyacaktır.

4.1 SDUD ve NDUD Karşılaştırılması

Yapılan testlerde ilk olarak NDUD ve SDUD yöntemlerinin bellekteki boyutunu karşılaştırdık. Daha sonra, ortak gezinim problemlerini ve öğrenme hızını çözmeye her

iki yöntemin etkinliğini karşılaştırdık ve gösterdik. Daha sonra engel önceliklendirme algoritmasını (GSDUD) aynı senaryolar üzerinde çoklu robot gezinimi için kullandık.

Q-Öğrenmeyi sürekli durum uzayı üzerinde çalışmaya uygun olmadığı için, özelliklerin ayrıklaştırılması gerekir. Ayrıklaştırma işlemini yaparken, özellikleri daha fazla parçaya ayırmak daha güvenilir ve daha düzgün olsa da ayrıklaştırmanın bir sınırlaması vardır. Durum uzayını ne kadar ayrıklaştırırsak, modeli hafızada tutmak ve kaydetmek de o kadar daha fazla belleğe ihtiyacı duyacaktır ve ayrıca öğrenme süresi de orantılı şekilde artacaktır.

Sürekli durum alanına sahip bir problem için maksimum ayrıklaştırma miktarını iki kriter belirler: gerekli özellik sayısı ve RAM bellek boyutu. Özellikler, esas olarak problem tanımına bağlıdır ve onun tarafından belirlenir; bu nedenle, problemin tanımı değiştirilmeden model özelliklerinin sayıları kolayca değiştirilemez, ama öte yandan ayrıklaştırma miktarı problemin çözümünü etkilemediği sürece azaltılarak hafıza sorununu azaltır. Örneğin, yol planlama problemlerinde yetersiz ayrıklaştırma engellerden kaçınmayı önleyebilir. Bu nedenle, deneme yanılma yoluyla her özellik için ihtiyaçlarımıza göre bir ayrıklaştırma eşiği bulduk. Robotun yaptığı hareketlerin yumuşaklığı farklı yöntemlerle de elde edilebilir, örneğin, bu bir adımı daha küçük adımlara bölerek eylemleri yumuşatma göreviyle ana denetleyici ile simülatör veya robot bellenimi(firmware) arasına basit bir denetleyici birimi yerleştirerek yapılabilir.

Son uygulama tarafından kullanılan bellek boyutu, özellik sayısı, ayrıklaştırma miktarı ve veri türü ile doğrudan ilişkilidir. Çok sayıda deneme yaptıktan sonra, d_g (hedeften uzaklık), a_g (hedefe olan açı), a_r (robotun mevcut açısı), d_{r_i} (diğer robottan uzaklık), a_{r_i} (diğer robota açı) özellikleri için en iyi ayrıklaştırma sırasıyla 18, 18, 8, 18 ve 8 olarak bulunmuştur. Bu değerler bize eylemleri mevcut bellek boyutunu aşmadan gerçekleştirmemizi ve başarılı gezinim elde etmemizi sağlar. Ayrıca bu değerler kabul edilebilir bir pürüzsüzlük sağlar. g_r (hedefe ulaşıldı) ve c_r (çarpışma özellikleri) özellikleri sadece iki duruma sahiptir, bu nedenle ayrıklaştırmadan sonra değişmezler. Eldeki bu değerlerle, her iki yöntem için toplam bellek kullanımı ve toplam durum sayısı tablo 4.1’de verilmiştir.

karşılaştırmalarını göstermektedir. Ayırıklaştırmalar, güçlü bir bilgisayar için nispeten makul miktarda RAM kullanımı artana kadar devam etmektedir. İkinci sütundaki ortamın özellikleri önceki bölümde hem NDUD hem de SDUD için açıklanmıştır. Sonuçlar, SDUD kullanılarak bellek kullanım miktarında önemli miktarda %70 azalma olduğunu göstermektedir. Ayrıca daha fazla robot eklemenin pratik olmadığını belirtmekte fayda var; aksi takdirde ihtiyaç duyulan RAM miktarı makul sınırları aşacak veya ayırıklaştırma miktarı problem çözilemeyecek şekilde düşecektir. Örneğin, dördüncü sıraya fazladan bir robot eklersek, model iki ekstra özelliğe ihtiyaç duyacak ve bu Q-Tablo boyutunu 144 kat artıracaktır ki toplamda yaklaşık 10.566 GB RAM hafıza ihtiyacı anlamına gelmektedir.

Çizelge 4.1 NDUD ve SDUD kullanarak oluşacak toplam durum sayısı ve ihtiyaç duyulan RAM miktarı göstermektedir. İlk sütun, ortamdaki robot sayısını gösterir. İkinci sütun, ortamın her özelliği için farklı ayırıklaştırma miktarlarını gösterir. Sonraki iki sütun, NDUD ve SDUD yöntemleri kullanılarak ayırıklaştırma miktarına göre hesaplanan durum sayısını ifade etmektedir. Sonraki iki sütun, Q-Tablolarının Python ile hesaplanan bellekteki gerçek boyutlarını ifade etmektedir. Son sütun, NDUD yerine SDUD kullanan Q-Tablosunun indirgeme yüzdesini sunmaktadır.

Çizelge 4.1 Q-Tablosu eleman sayısı ve bellekteki gerçek boyutla ilgili çok ajanlı gezinim problemleri için farklı ayırıklaştırmalarla NDUD ve SDUD arasındaki boyut

	Robot sayısı	Özellik Ayırıklaştırma miktarı $(d_g, a_g, a_r, (d_{r_1}, a_{r_1}, \dots, d_{r_n}, a_{r_n}))$	Q-Tablosu elemanlarının sayısı NDUD	Q-Tablosu elemanlarının sayısı SDUD	Bellekteki boyutu NDUD	Bellekteki boyutu SDUD	İndirgeme yüzdesi
1	3	(6, 6, 4, (6, 4, ...))	1,327,104	442,368	5.06MB	1.69MB	66.6%
2	3	(12, 12, 4, (12, 4, ...))	21,233,664	6,193,152	81MB	23MB	70.8%
3	3	(18, 18, 8, (18, 8, ...))	214,990,848	59,719,680	3280MB	911MB	72.2%
4	3	(24, 24, 12, (24, 12, ...))	9,172,942,848	2,484,338,688	34992MB	9477MB	72.9%
5	3	(36, 36, 16, (36, 16, ...))	110,075,314,176	29,047,652,352	419840MB	110592MB	73.6%
6	4	(6, 6, 4, (6, 4, ...))	31,850,496	10,616,832	121.5MB	40.5MB	66.6%
7	4	(12, 12, 4, (12, 4, ...))	1,019,215,872	297,271,296	3888MB	1134MB	70.8%
8	4	(18, 18, 8, (18, 8, ...))	123,834,728,448	34,398,535,680	472064MB	117760MB	72.2%

Ön deney sonuçları dikkate alınarak, son deneylerin yapılması için öğrenme oranı α "0.7" ve indirim faktörü γ "0.9" olarak belirlenmiştir. Deneylerimizde, robotu hedefe yaklaştıran eylemler için ekstra pozitif bir ödül vermenin hem öğrenme sürecini hızlandırdığını hem de robotun optimal olmayan döngülerden (suboptimal loops) daha hızlı çıkmasına yardımcı olduğunu gözlemlenmiştir. Bu ekstra ödülü vermemek, öğrenme sürecini önemli ölçüde artırır. Optimum olmayan bir döngüye yapışan robotlar aynı duruma geri döner ve birkaç kez daha aynı eylemi seçer, ancak bu mekanizma ile sorun uzun sürmez; çünkü bir süre sonra robot kendisini hedefe yaklaştıran hareketi tercih etmeye başlar. Öte yandan, bu mekanizma modelin yerel minimumları (yani, U şeklindeki engeller veya hedefe yakın boş odalar) geçmeyi ve onlardan çıkmayı öğrenmesini uzatır. Ancak, deney sonuçlarımıza göre, genellikle ve çoğu senaryoda bu ihmal edilebilir bir zamandır. Optimal olmayan döngüleri işlemek için kullandığımız bir başka mekanizma, robot tarafından atılan son n adımı tutmak ve aynı öğrenme bölümünde aynı durumda aynı eylemi seçmeyi engellemektir. Bu mekanizma, ajanı oluşan bu durumlarda keşfetmeye teşvik etmeyi amaçlıyor.

SDUD'nin etkinliğini göstermek ve problemi çözmede NDUD ile karşılaştırmak için daha öncede belirtildiği üzere Şekil 1a, 1b ve 1c'de gösterilen üç farklı senaryo tasarlanmıştır. Birinci senaryoda, U şeklinde dağınık engellerin olduğu bir ortam kullanır. İkinci senaryo, koridorlu bir bina içindeki klasik gezinim problemini göstermektedir. Üçüncü ve son senaryo ise dağınık engellerin ve bir odanın birleşimini sunmaktadır. Her üç senaryo engellerin yanı sıra farklı yerel minimum engellere sahiptirler. Bunlar birinci odadaki U şeklindeki duvarlar, ikinci ve üçüncü olandaki boş odalar ve köşelerdir.

4.2 SDUD sonuçları ve öğrenme grafiği ve karmaşıklık analizi

Çalışan her bölümü değerlendirmek ve öğrenmedeki ilerlemeyi görmek için denklem (4.1) kullanılmıştır. Bu formülü kullanarak ödüller sırayla en yüksekten en düşüğe

aşağıdaki gibi sıralanır: En yüksek ödül, robotun hedefe en kısa yoldan ulaştığı bölüme¹ verilir ve ardından robotun uzun bir yoldan hedefe ulaştığı bölüm değerlendirilir. Bir sonraki yüksek ödül, robotun hedefe ulaşmadığı ama aynı zamanda çarpışmadığı bölümlere verilir. Eğitim süresince kazandığımız tecrübelerle bu durumun kısa süre gerçekleştiğini gözlemledik. Bu davranışın sebebi modelin engel konumlarını öğrenmesine rağmen daha hedefe giden yolu öğrenememesidir. Bu durum, ajanın ortamı öğrenmede önemli miktarda ilerleme kaydettiği anlamına gelir; aksi takdirde bir süre gezdikten sonra engele çarpacaktı. Bu tür bölümlerin ardından, robotun bir süre gezindiği ve ardından bir engelle çarptığı bölümlerdeki durumlar meydana gelir. Sonraki ve en kötü durumlar, bölüm başladıktan kısa bir süre sonra robotların çarptığı bölümlerdir. Bu genellikle öğrenmenin başında olur ve modelin pek bir bilgiye sahip olmadığını gösterir. Bu kriterlere göre, ts_r 'nin toplam bölüm ödülünü, g_r 'nin hedef ödülünü, s_r 'nin adım ödülünü, c_r 'nin çarpışma ödülünü ve s_c 'nin adım sayısını temsil ettiği denklem (4.1)'i elde edilir. $\left|\frac{g_r}{s_r}\right|$ ile g_r 'yi çarpmak ve $\left|\frac{c_r}{s_r}\right|$ ile c_r 'yi çarpmak belirtilen durumlar arasındaki farkı yoğunlaştırır ve daha ayırt edilebilir hale getirir.

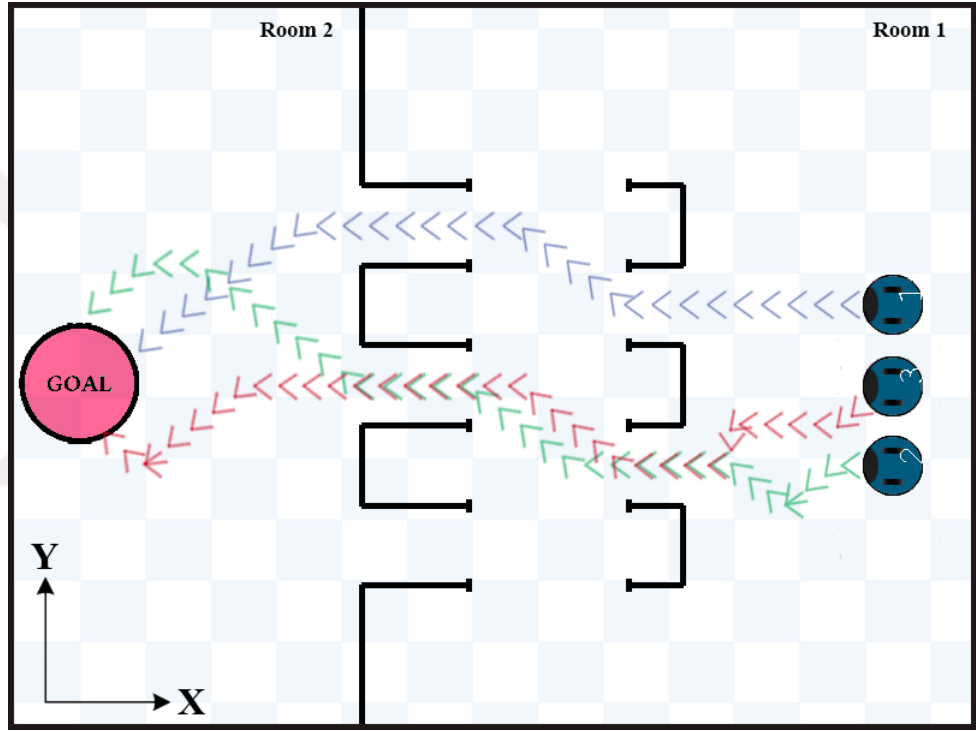
$$e_r = \frac{ts_r + g_r * \left|\frac{g_r}{s_r}\right| + c_r * \left|\frac{c_r}{s_r}\right|}{s_c} \quad (4.1)$$

Karmaşıklık açısından, bizim problemimizde ajanların başlangıçta durum uzayının topolojisinin bilgisine sahip olmadıklarına göre bilinmeyen bir durum uzayı sayılıyor. Ek olarak, bu problemde yinelenen eylemler yoktur ve başlangıçta Q-tablosu sıfır ile doldurulur. Daha önceki çalışmalara göre (Aleliunas, Karp, Lipton, Lovász ve Rackoff, 1979; Koenig ve Simmons, 1993), bu konfigürasyonlarla bir hedefe veya nihai duruma ulaşmanın en kötü durum karmaşıklığı $O(n^3)$ 'tür.

¹ Öğrenme sürecinde, bölüm(episode) başlangıç ve bitiş durumları ve o aralıktaki tüm durumların toplamına söylenir.

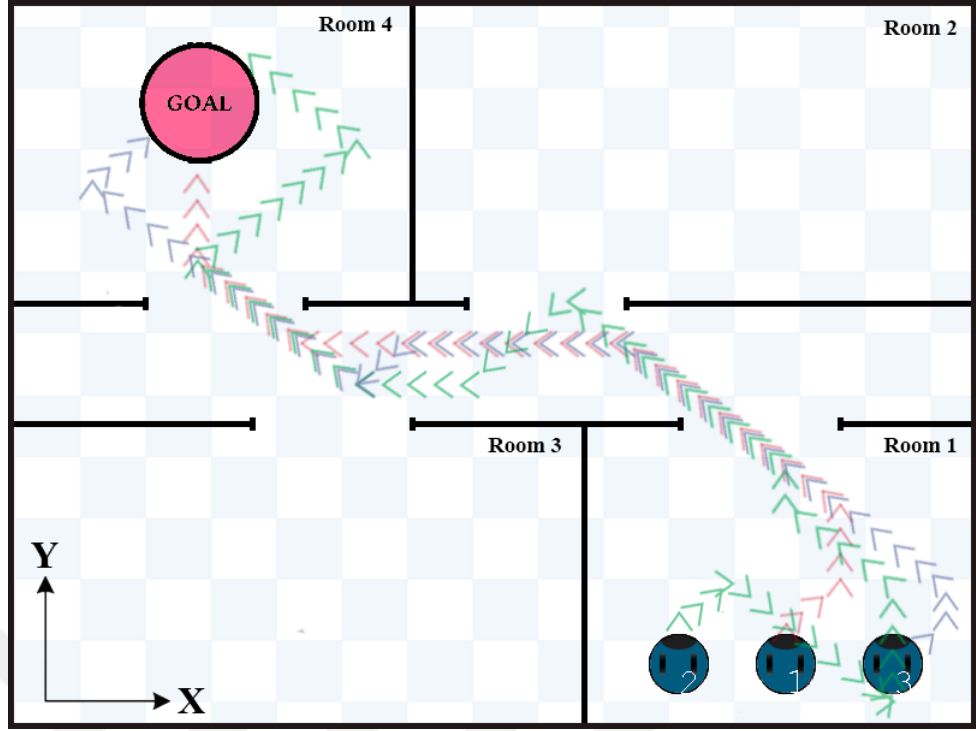
4.3 Gezinim (Navigasyon) Sonuçları

Hem NDUD hem de SDUD ile eğitilmiş modellerden elde edilen sonuçlara baktığımızda, her iki yöntemin de robotu kabul edilebilir kısa ve güvenli bir yoldan hedefe yönlendirmede başarılı olduklarını gördük. Her iki yöntemin de sonuçları çok benzerlik göstermektedir, bu nedenle teze sadece SDUD sonuçlarının resimlerini ekledik. Her senaryo için oluşturulan örnek yol, şekil 4.1a, b ve c’de gösterilmektedir.

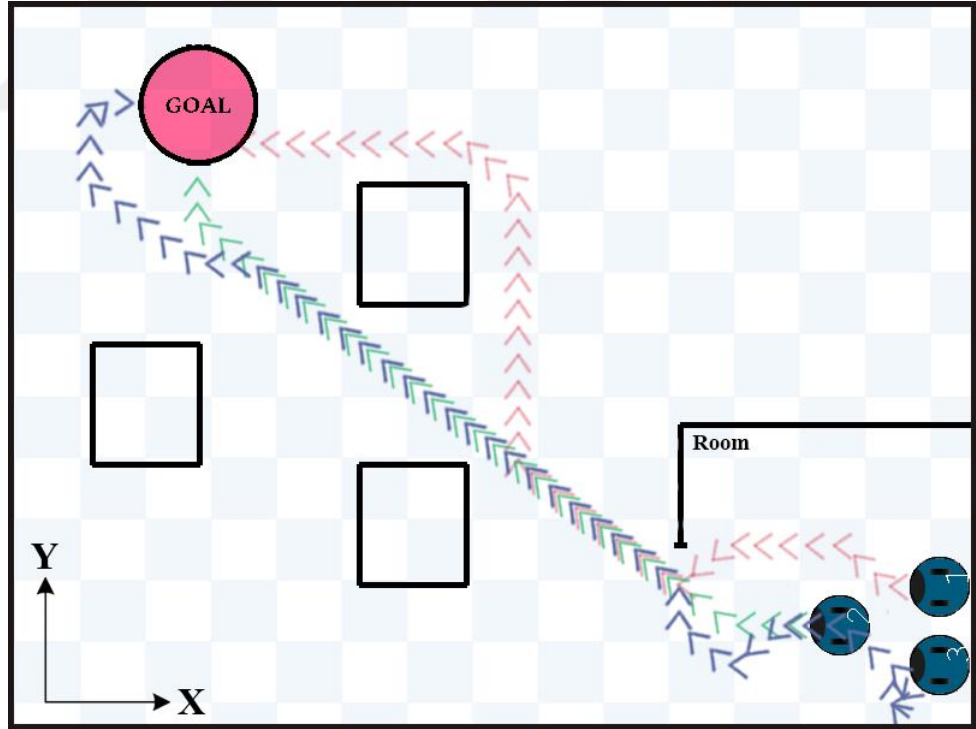


(a)

Şekil 4.1 Üç robotla SDUD kullanımının gezinim sonuçları



(b)



(c)

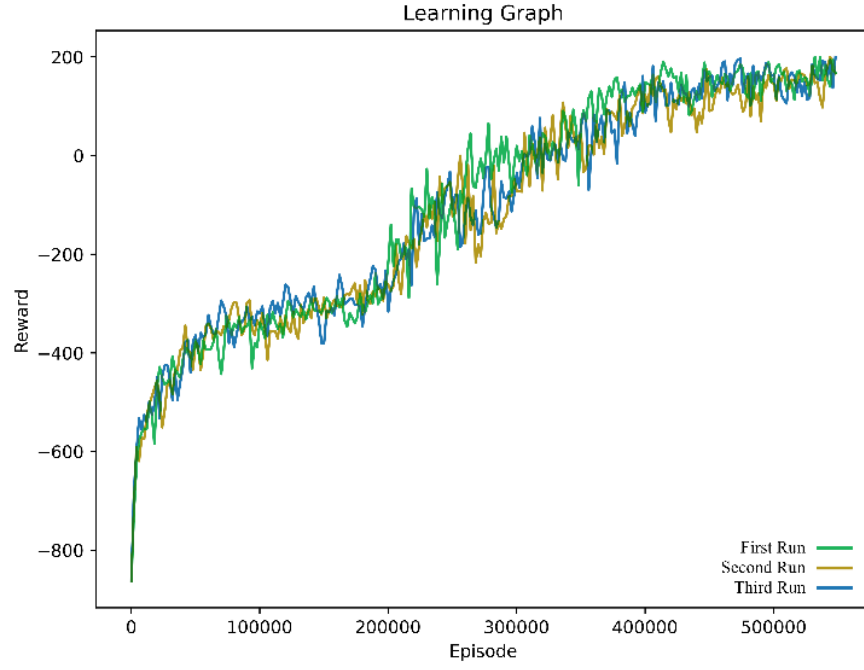
Şekil 4.1 Üç robotla SDUD kullanımının gezinim sonuçları (devam)

(a) senaryo 1; (b) senaryo 2; (c) senaryo 3.

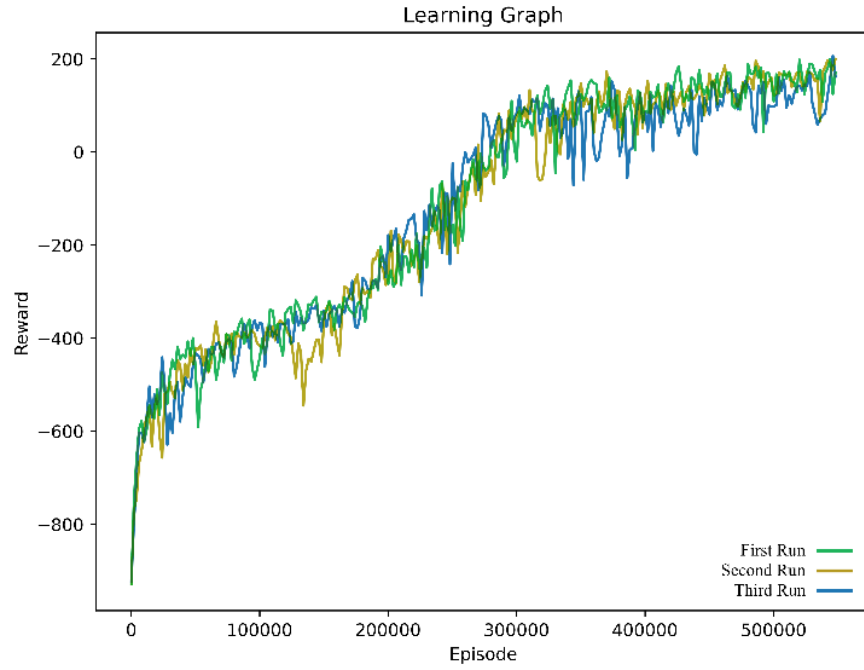
Burada gösterilen yollar, öğrenilen modelden yararlanırken elde ettiğimiz yollardır; ancak tüm robot dönüşlerini dahil edilmemiştir. Bazen, robotlar yolları üzerinde bir sonraki ara noktaya devam etmeden önce hiçbir şey yapma eylemini veya her yol noktasında sola veya sağa dönüş eylemlerini seçer. Ancak, daha fazla okunabilirlik için bu davranışlar ilgili şekle eklenmemiş sadece son eylemler eklenmiştir. Bu ekstra eylemler bir aracı denetleyici tarafından kolaylıkla kaldırılabilir. Deneyler, Bellman denkleminin, bir süre sonra kabul edilebilir bir kısa yol bulmakta başarılı olduğunu göstermektedir. Oluşturulan yol bazen ihmal edilebilir sapmaları da içerebilmektedir. Ancak bu görünüşte küçük kusurları öğrenmek için öğrenme süresini iki veya üç katına çıkarmak gerekmektedir.

Öte yandan, öğrenme sürecini aşırı uzatmak, robotların önceden bulunan kısa yolları unutmasına neden olabiliyor. Bu sorun, çok ajanlı bir ortamda daha da artar. Aşırı keşif, diğer ajanları bozabilir ve keşfedici ajanın öğrenme görevini daha zor hale getirebilir (Buşoniu ve diğerleri, 2010). Fakat tüm sapmaların bu türden olmadığını da bilmemiz gerekir, örneğin şekil 4.1c'deki yolun başlangıcında robot 3'ün dönüşü farklılık ihtiva etmektedir. Aslında bu dönüş manevrası çarpışmaları önlemek içindir.

4.4 NDUD ve SDUD Öğrenme Grafikleri



(a)



(b)

Şekil 4.2 Öğrenme grafiği

(a) senaryo 2 için NDUD öğrenme grafiği; (b) senaryo 2 için SDUD öğrenme grafiği

Şekil 4.2a ve b, şekil 3.9b'deki senaryo için sırasıyla NDUD ve SDUD için üç farklı deneyin öğrenme grafiklerini göstermektedir. Grafiğin dikey eksenini ilgili sezonda kazanılan ödül ve yatay eksenini ise sezon numarasını göstermektedir. Grafikler denklem (4.1)'i kullanarak çizilir. Her 200 sezondan¹ sonra bu grafikleri çizmek için algoritma, şimdiye kadar eğitilmiş modelin kalitesini ölçmek için yalnızca 50 sezon boyunca sömürü modunda (exploitation mode) çalışacak şekilde ayarlanmıştır. Bu 50 sezondaki toplam ödülün ortalaması grafikteki bir noktayı temsil ediyor. Hem NDUD hem de SDUD'nin öğrenme grafiklerinin çok benzer olduğu görülebilir. Bu grafikleri analiz ederken, bu yöntemlerin bir dereceye kadar stokastik olduğunu, yani aynı yöntemle aynı problemin bir çalıştırmada diğerine kıyasla daha hızlı çözüm bulabileceği anlamına geldiğini unutmamalıyız.

Öte yandan, kabul edilebilir bir model oluşturmak için, durumların çoğunun araştırılması gerekmektedir ve bu doğal olarak zaman alacak. Dolayısıyla kabul edilebilir sonucu çok hızlı bulmalarını beklemiyoruz. Öğrenmenin başlangıcında keşif yüzdesi yüksekken, sonlara doğru öğrenme ilerledikçe adım adım azalır. Ayrıca, başlangıçta verilen Q-Tablosundaki varsayılan sıfır değeri, denetleyiciyi durumların çoğunu keşfetmeye teşvik eder.

Bu iki yöntemi karşılaştırırken dikkat etmemiz gereken konu şu ki aslında SDUD kullandığımızda bile gerçek durum-uzayı değişmez. Tek fark, bellek kullanımındadır. Söylenen sebepler, en iyi cevabın her zaman erken bir aşamada veya her zaman aynı aşamada bulunamamasının nedenleridir. Sonuçlar incelendiğinde, beklendiği gibi, her iki yöntem de yaklaşık 400.000 sezon sonra makul bir çözüme yaklaştı.

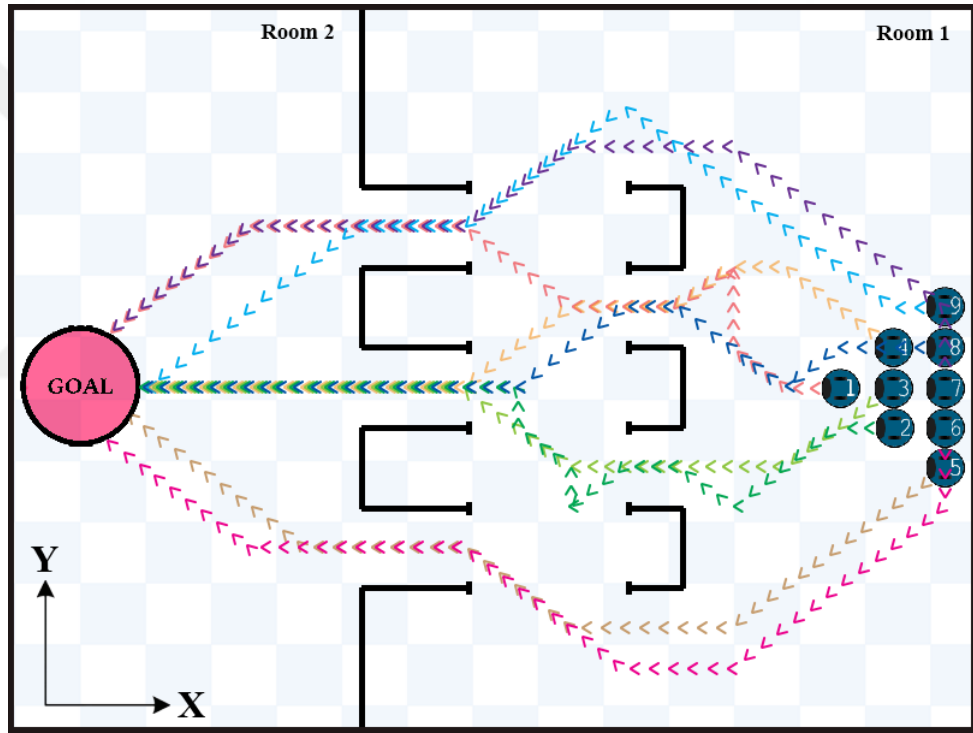
4.5 Genelleştirilmiş SDUD

Çok ajanlı takviyeli öğrenmede birden çok ajan bulunması nedeniyle, ek sorunlar ortaya çıkmaktadır. Ajanlar, sadece çevre hakkında değil, aynı zamanda diğer robotların

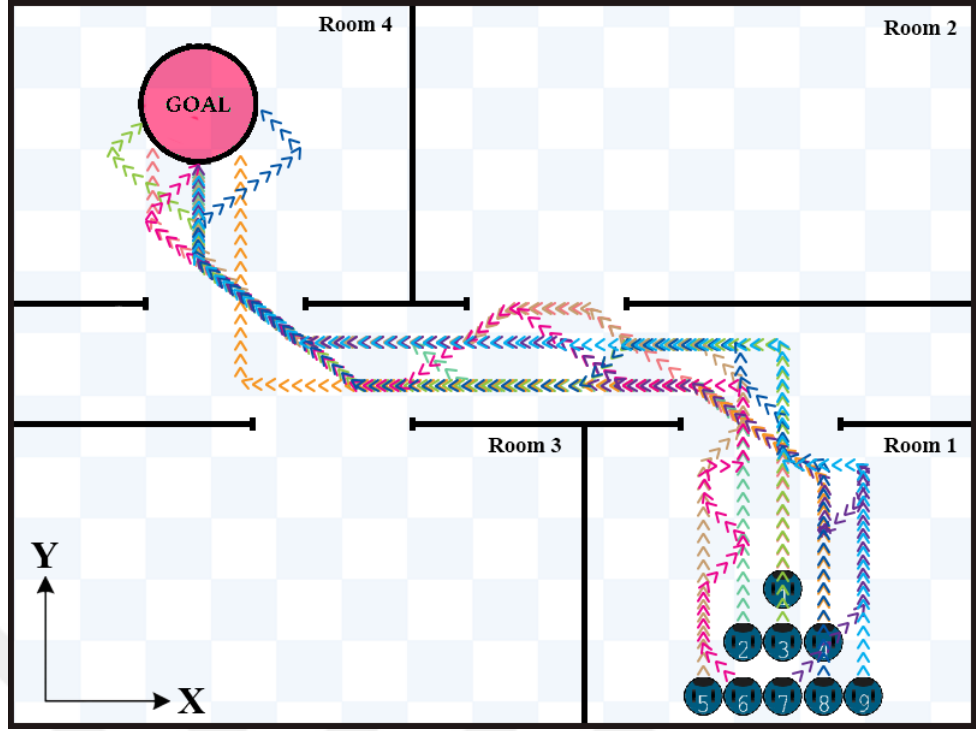
¹ “Sezon”, ajanın tanımlı olduğu ortamdaki başlangıç durumundan final duruma kadar atılan adım-zaman setine verilen addır. Bu problemde final durum hedefe varmak, engele çarpmak veya atılan adımın sayısının 1000'i geçmesi olabilir.

eylemlerine uyum sağlayabilmek ve reaksiyon geliřtirmek için de bilgi edinmelidir. Bununla birlikte, fazla uzun süre keřif yapmak diğerk ajanların optimum yollarını bozabilir ve bu da keřif ajanının öğrenme görevini daha zor hale getirebilir.

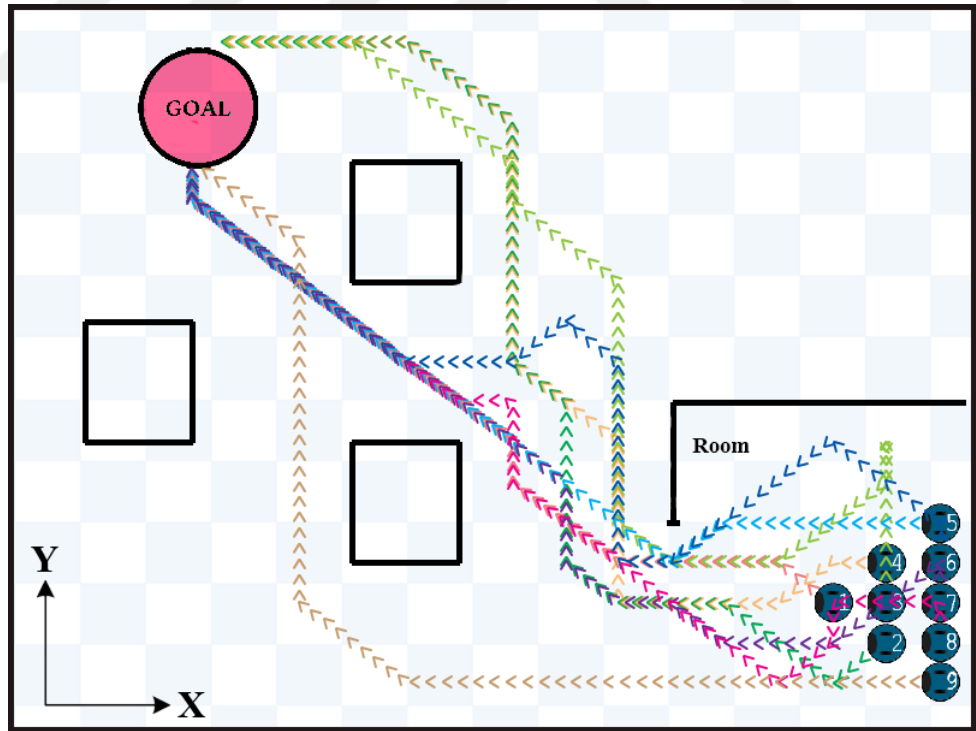
Sonraki deney setinde, sürü gezinim algoritması ve engel önceliklendirme algoritmasını (GSDUD) kullanarak daha fazla robotla deneyler yaptık. Bunun için önceki deneydeki aynı üç senaryoyu kullandık. Bu deney setinde dokuz robottan oluşan bir sürü seçtik. On sekiz robotla yaptığımız deney sonuçlarını resimle göstermek mümkün değildi, o yüzden onun videolarını paylaşıyoruz. Şekil 4.3 örnek sonuçları göstermektedir:



(a)



(b)



(c)

Şekil 4.3 GSDUD kullanarak dokuz robot gezinimi sonuçları

Bu senaryolardaki üç robot deneyinden farklı olarak, hedef noktasının etrafındaki tüm robotlar için yeterli alan yoktu, bu yüzden hedefe ulaşan robotları park noktalarına kaldırdık. Bu nedenle 9 robot deneyinde aynı noktadan birden fazla robot çarpışma yaşanmadan hedefe ulaşabilir.

Çizelge 4.2 Farklı sayıda robotla yapılan deneylerin performans sonuçları

Senaryo No.	Robot Sayısı	Zaman(saniye)	Ortalama açısal hız (Derece°)	Ortalama kat edilen mesafe(cm)	Min. kat edilen mesafe(cm)	Maks. Kat edilen mesafe (cm)
1	3	12	24	531	508	554
	9	17	18	566	500	645
	18	20	25	605	481	545
2	3	17	20	793	612	1025
	9	21	21	697	579	787
	18	25	23	743	558	730
3	3	14	21	650	508	783
	9	25	19	673	525	783
	18	24	22	716	466	866

Çizelge 4.2’den de görüldüğü üzere tüm senaryolar için üretilen çözümler makul ve optimuma yakın aralıktadırlar. Çoğu durumda, daha fazla robotla yapılan testlerin tamamlanması daha uzun sürdü; bu kısmen dinamik engellerin aktif robotun kısa yoldan sapmasına neden olmasından kaynaklanmaktadır. Başka bir deyişle, daha fazla robotun varlığı, ortamların belirsizliğini artır ve bu nedenle, özellikle ön hattın arkasındaki robotlar için geleceğe dönük durum tahmini daha zor hale gelir; ancak sonuçlardaki ortalama değerlere bakıldığında ihmal edilemez bir sorun olduğu söylenemez. Bunun başlıca nedeni, oluşturulan optimal ve optimal olmayan yolların sayısının neredeyse eşit olması ve ortalamanın üç robot durumunda olduğu gibi aynı aralıkta olmasıdır. Dahası, kat edilen minimum ve maksimum mesafeyi analiz edince, daha fazla robot eklemek, modelin daha iyi bir çözüm öğrenmesine yardımcı olduğunu ve ayrıca çoğu durumda ürettiği en kötü yolun da üç robotlu senaryodan daha iyi olduğunu söyleyebiliriz. Bunun başlıca nedeni, robot sayısı arttıkça öğrenme modeline sağlanan örnek verinin de artmasıdır.

5. SONUÇ VE TARTIŞMA

Q-Öğrenme, model tabanlı algoritmalara kıyasla çevreyi modellemeden veya ortamın modeline ihtiyaç duymadan politikaları (hareket tarzı) çıkaran, daha esnek, basit ve ayrıca uyarlanabilir ve modelsiz bir takviyeli öğrenme algoritmasıdır. Ayrıca Q-Öğrenme adaptasyon gerektirmeden rassal geçişlerde ödüllerle ilgili sorunları çözebilir. Q-öğrenme, mevcut durumdan başlayarak, birbirini takip eden tüm adımlar boyunca toplam ödülün beklenen değerini maksimize etme anlamında optimal bir politika bulmakta başarılıdır. Ancak, önemli miktarda bellek gerektirdiğinden, büyük görevler için kullanılması olanaksız hale gelir. Bu tez çalışmasında özellikle aynı Q-Öğrenme sistemi konfigürasyonlarını kullanan robotlardan oluşan çoklu-ajan gezinimi gibi karmaşık sorunları çözmeyi mümkün kılmak için durum uzayı boyutunu sıkıştıran yeni bir yöntem (SDUD) tanıtılmıştır. Bu yöntem tekrarlanan özellikleri teke düşürerek ve ayrıca örtüşmeyen özellikleri yığarak hem hafıza tüketimini önemli ölçüde azaltmıştır hem de az olsa da eğitim sonucu oluşan modelin genelleme özelliğini artırmıştır. Deneylerin sonucuna bakarsak bu yöntemle oluşturulan sistem, üç robottan oluşan çoklu ajan gezinmede başarılıdır, fakat ortama kullanılan hafıza miktarından dolayı üç robottan daha fazla ajan eklemek mümkün değildi. Bunun için, tezde bir nesne önceliklendirme algoritması geliştirilerek duyuşsal verileri azaltmak için başka bir yöntem (GSDUD) tanıtıldı ve kullanıldı; İlk yöntemin üzerine entegre edilen bu yöntemin kullanılması, Q-Öğrenme kullanan birçok robottan oluşan sürüler için gezinmeyi mümkün kılmıştır. Bu yöntemin deneylerde 18 robotluk bir çoklu robot gezinme için başarılı sonuçlar oluşturduğu görülmüştür. Öğrenme grafikleri çizmek için önerilen formülümüzü kullanmak, öğrenmeyi bitirmek ve öğrenme aşaması başarısını belirlemek için uygun zamanın belirlenmesinde faydalı olmuştur. Deneyler ve yürütülen testler, farklı senaryolar ve robot sayıları için bellek azaltma oranını, gezinme sonuçlarını ve performans ölçümlerini sunmaktadırlar. Sonuçlar, SDUD, GSDUD yöntemlerini kullanan çok ajanlı görevler için önemli bellek kullanımı azalmasını ortaya koymaktadır. Ayrıca yapılan deneyler, bu tekniklerin optimale yakın çözümleri belirlemede ve gezinim problemini çözmede muvaffak olduğunu ortaya koymuştur. GSDUD ile daha fazla robot eklenmesinin ortalama performansı fazla etkilemediği sonuçlar ile ifade edilebilmektedir.

5.1 Gelecek Çalışmaları

Bu tezde sunulan SDUD ve GSDUD yöntemleri Q-Öğrenmenin hafıza tüketimini önemli ölçüde azaltmaktadır. Tezde sunulan örnek problemde bu yöntemlerin başarılı olduğu ayrıntılı olarak araştırıldı. SDUD yöntemi birçok toplu robot kullanan problemde etkili olacaktır, bunun sebebi ise bu yöntemlerin genellikle hedefe varma ve çarpışma özelliklerine ihtiyaç duymalarıdır. Fakat örtüşmeyen özelliklerin hangi problemlerde ve ne oranda bulunduğu başka bir araştırma kapsamı olabilir. Ayrıca, öğrenme grafiğine bakıldığında SDUD yönteminin NDUD yöntemine göre daha hızlı yakınsadığını görülmekte. Büyük olasılıkla bunun sebebi SDUD yönteminin tek bir çarpışma ve hedefe varma durumuna sahip olmasıdır. Bu yüzden, diğer problemlerde de tekrarlanan durumları bulmak ve onları teke indirerek öğrenme hızı ve genelleme özelliği nasıl etkilendiğini incelemek başka bir araştırma konusu olabilir.

Bir başka çalışma konusu da yapılan deneyleri gerçek robotlar üzerinde uygulamaktır. Bu deneylerde eğitim süresini düşürmek için eğitim simülasyon ortamlarında gerçekleştirilebilir. Tezde bahsedildiği gibi bu deneylerde iyi sonuçlar elde edebilmek için eğitim süresine dikkat edilmeli. Yetersiz eğitim gerçek robotların çarpışmasına veya en uygun yolu bulamamalarına sebep olurken fazla eğitim de buldukları optimum yolu unutmalarına sebep olabilir. Buna ek olarak robotların yanlış dönmelerini engellemek için bir ara denetleyici tasarlanabilir. Q-Öğrenme modelini kullanan denetleyici zaman zaman aynı durum için kaç defa zıt komutlar verebiliyor, örneğin sağa döndürdükten sonra tekrar sola döndürerek robotu aynı duruma geri getirebiliyor. Önerilen ara denetleyici bu durumları engellemek için kullanılabilir.

Birkaç çalışma ve geliştirme konusu aşağıdaki gibidir:

- SDUD yönteminin örtüşen durumları yığma ve durum sayısını düşürme özelliklerinin derin q-öğrenme üzerinde etkisi,
- GSDUD yönteminin üstesinden geldiği maksimum robot sayısını bulmak,

- SDUD yönteminin etkisini daha çeşitli ve karmaşık tasklar üzerinde incelemek. Örneğin, çekim gücü kullanarak akın etme sürü gezinmesi yapmak veya sürü için formasyon tanımlayarak gezinme.



KAYNAKLAR

- Abdoos, M., Mozayani, N. ve Bazzan, A.L.C. (2011). Traffic light control in non-stationary environments based on multi agent Q-learning. 2011 *14th International IEEE conference on intelligent transportation systems (ITSC)* içinde (ss. 1580–1585). IEEE.
- Ahmadabadi, M.N. ve Asadpour, M. (2002). Expertness based cooperative Q-learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 32(1), 66–76.
- Al., M.D. et al. (2014). Swarm robotics. *Scholarpedia*. doi:10.4249
- Aleliunas, R., Karp, R. M., Lipton, R.J., Lovász, L. ve Rackoff, C. (1979). Random walks, universal traversal sequences, and the complexity of maze problems. *20th Annual Symposium on Foundations of Computer Science (sfcs 1979)* içinde (ss. 218–223). IEEE Computer Society.
- Amado, L. ve Meneguzzi, F. (2018). Q-Table compression for reinforcement learning. *The Knowledge Engineering Review*, 33, 1–21. doi:10.1017/ s02698 889 18 00 0280
- Azar, M.G., Munos, R., Ghavamzadeh, M. ve Kappen, H.J. (2011). Speedy Q-learning.
- Bahgeçi, E. ve Sahin, E. (2005). Evolving aggregation behaviors for swarm robotic systems: A systematic case study. *Proceedings 2005 IEEE Swarm Intelligence Symposium, 2005. SIS 2005.* içinde (ss. 333–340). IEEE.
- Berenji, H.R. (1996). Fuzzy Q-learning for generalization of reinforcement learning. *Proceedings of IEEE 5th International Fuzzy Systems* içinde (C. 3, ss. 2208–2214). IEEE.
- Bertsekas, D.P. (2005). Dynamic Programming and Optimal Control, volume 1 of Optimization and Computation Series. *Athena Scientific, Belmont, MA, USA, 3rd edition. Cited on, 2.*
- Bode, N.W.F., Wood, A.J. ve Franks, D.W. (2011). Social networks and models for collective motion in animals. *Behavioral ecology and sociobiology*, 65(2), 117–130.
- Borkar, V.S. ve Meyn, S.P. (2000). The ODE method for convergence of stochastic approximation and reinforcement learning. *SIAM Journal on Control and Optimization*, 38(2), 447–469.
- Brambilla, M., Ferrante, E., Birattari, M. ve Dorigo, M. (2013). Swarm robotics: a review from the swarm engineering perspective. *Swarm Intelligence*, 7(1), 1–41. doi:10.1007/s11721-012-0075-2
- Buşoniu, L., Babuška, R. ve De Schutter, B. (2010). Multi-agent reinforcement learning: An overview. *Innovations in multi-agent systems and applications-1*, 183–221.
- Camazine, S., Deneubourg, J.-L., Franks, N.R., Sneyd, J., Bonabeau, E. ve Theraula, G. (2003). *Self-organization in biological systems*. Princeton university press.

- Celikkanat, H. ve Şahin, E. (2010). Steering self-organized robot flocks through externally guided individuals. *Neural Computing and Applications*, 19(6), 849–865.
- Chamanbaz, M., Mateo, D., Zoss, B.M., Tokić, G., Wilhelm, E., Bouffanais, R. ve Yue, D.K.P. (2017). Swarm-enabling technology for multi-robot systems. *Frontiers in Robotics and AI*, 4, 12.
- Cheng, T.M. ve Savkin, A.V. (2011). Decentralized control for mobile robotic sensor network self-deployment: Barrier and sweep coverage problems. *Robotica*, 29(2), 283.
- Cherkassky, V. ve Mulier, F.M. (2007). *Learning from data: concepts, theory, and methods*. John Wiley & Sons.
- Costa, V., Duarte, M., Rodrigues, T., Oliveira, S.M. ve Christensen, A.L. (2016). Design and development of an inexpensive aquatic swarm robotics system. *Oceans 2016-Shanghai* içinde (ss. 1–7). IEEE.
- Dearden, R., Friedman, N. ve Russell, S. (1998). Bayesian Q-learning. *Aaai/iaai* içinde (ss. 761–768).
- Deisenroth, M. ve Rasmussen, C.E. (2011). PILCO: A model-based and data-efficient approach to policy search. *Proceedings of the 28th International Conference on machine learning (ICML-11)* içinde (ss. 465–472).
- Devraj, A.M. ve Meyn, S.P. (2017a). Zap Q-learning. *Proceedings of the 31st International Conference on Neural Information Processing Systems* içinde (ss. 2232–2241).
- Devraj, A.M. ve Meyn, S.P. (2017b). Fastest convergence for Q-learning. *arXiv preprint arXiv:1707.03770*.
- Er, M.J. ve Deng, C. (2004). Online tuning of fuzzy inference systems using dynamic fuzzy Q-learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 34(3), 1478–1489.
- Ferrante, E., Turgut, A.E., Huepe, C., Stranieri, A., Pinciroli, C. ve Dorigo, M. (2012). Self-organized flocking with a mobile robot swarm: a novel motion control method. *Adaptive Behavior*, 20(6), 460–477.
- Flierl, G., Grünbaum, D., Levins, S. ve Olson, D. (1999). From individuals to aggregations: the interplay between behavior and physics. *Journal of Theoretical biology*, 196(4), 397–454.
- Gehring, C., Kaelbling, L.P. ve Lozano-Perez, T. (2018). Adaptable replanning with compressed linear action models for learning from demonstrations. *Proceedings of The 2nd Conference on Robot Learning*, 87(CoRL), 432–442.
- Giannoccaro, I. ve Pontrandolfo, P. (2002). Inventory management in supply chains: a reinforcement learning approach. *International Journal of Production Economics*, 78(2), 153–161.
- Glorennec, P.Y. ve Jouffe, L. (1997). Fuzzy Q-learning. *Proceedings of 6th international fuzzy systems conference* içinde (C. 2, ss. 659–662). IEEE.
- Goodfellow, I., Bengio, Y. ve Courville, A. (2016). *Deep learning*. MIT press.

- Guo, M., Liu, Y. ve Malec, J. (2004). A new Q-learning algorithm based on the metropolis criterion. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 34(5), 2140–2143.
- Güzel, M.S. ve Kayakökü, H. (2017). A collective behaviour framework for multi-agent systems. *Mechatronics and robotics engineering for advanced and intelligent manufacturing* içinde (ss. 61–71). Springer.
- Güzel, M.S., Ajabshir, V. B., Nattharith, P., Gezer, E.C. ve Can, S. (2018). A Novel Framework for Multi-Agent Systems Using a Decentralized Strategy. *Robotica*, 1–17. doi:10.1017/S0263574718001261
- Güzel, M.S., Ajabshir, V.B., Nattharith, P., Gezer, E.C. ve Can, S. (2019). A novel framework for multi-agent systems using a decentralized strategy. *Robotica*, 37(4), 691–707.
- Hasselt, H. (2010). Double Q-learning. *Advances in neural information processing systems*, 23, 2613–2621.
- Hinton, G.E. ve Sejnowski, T. J. (1999). *Unsupervised learning: foundations of neural computation*. MIT press.
- Hu, J. ve Wellman, M.P. (2003). Nash Q-learning for general-sum stochastic games. *Journal of machine learning research*, 4(Nov), 1039–1069.
- Jaakkola, T., Jordan, M. I. ve Singh, S. P. (1994). Convergence of stochastic iterative dynamic programming algorithms. *Advances in neural information processing systems* içinde (ss. 703–710).
- Jaidee, U. ve Muñoz-Avila, H. (2012). Classq-l: A q-learning algorithm for adversarial real-time strategy games. *Eighth Artificial Intelligence and Interactive Digital Entertainment Conference* içinde .
- Jin, C., Allen-Zhu, Z., Bubeck, S. ve Jordan, M.I. (2018). Is Q-learning provably efficient? *Advances in Neural Information Processing Systems* içinde (ss. 4863–4873).
- Kaelbling, L.P., Littman, M.L. ve Moore, A.W. (1996). Reinforcement Learning: A Survey. *Journal of Artificial Intelligence Research*, 4, 237–285. doi:10.1613/jair.301
- Kim, C., Kang, S., Shin, D., Choi, S., Kim, Y. ve Yoo, H.-J. (2019). A 2.1 TFLOPS/W mobile deep RL accelerator with transposable PE array and experience compression. *2019 IEEE International Solid-State Circuits Conference-(ISSCC)* içinde (ss. 136–138). IEEE.
- Kober, J., Bagnell, J.A. ve Peters, J. (2013). Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32(11), 1238–1274.
- Koenig, S. ve Simmons, R.G. (1993). Complexity analysis of real-time reinforcement learning. *AAAI* içinde (ss. 99–107).
- Kumar, H., Koppel, A. ve Ribeiro, A. (2019). On the sample complexity of actor-critic method for reinforcement learning with function approximation. *arXiv preprint arXiv:1910.08412*.

- Lan, Q., Pan, Y., Fyshe, A. ve White, M. (2020). Maxmin q-learning: Controlling the estimation bias of q-learning. *arXiv preprint arXiv:2002.06487*.
- Li, Q., Gama, F., Ribeiro, A. ve Prorok, A. (2020). Graph neural networks for decentralized multi-robot path planning. *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* içinde (ss. 11785–11792). IEEE.
- Long, P., Fanl, T., Liao, X., Liu, W., Zhang, H. ve Pan, J. (2018). Towards optimally decentralized multi-robot collision avoidance via deep reinforcement learning. *Proceedings - IEEE International Conference on Robotics and Automation*, 6252–6259. doi:10.1109/ICRA.2018.8461113
- Low, E.S., Ong, P. ve Cheah, K. C. (2019). Solving the optimal path planning of a mobile robot using improved Q-learning. *Robotics and Autonomous Systems*, 115, 143–161.
- Matignon, L., Laurent, G. J. ve Le Fort-Piat, N. (2007). Hysteretic q-learning: an algorithm for decentralized reinforcement learning in cooperative multi-agent teams. *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems* içinde (ss. 64–69). IEEE.
- Melo, F. S. (2001). Convergence of Q-learning: A simple proof. *Institute Of Systems and Robotics, Tech. Rep*, 1–4.
- Millán, J. D. R., Posenato, D. ve Dedieu, E. (2002). Continuous-action Q-learning. *Machine Learning*, 49(2–3), 247–265.
- Mohan, Y. ve Ponnambalam, S. G. (2009). An extensive review of research in swarm robotics. *2009 World Congress on Nature & Biologically Inspired Computing (NaBIC)* içinde (ss. 140–145). IEEE.
- Mostaani, A., Vu, T.X., Chatzinotas, S. ve Ottersten, B. (2020). Task-Based Information Compression for Multi-Agent Communication Problems with Channel Rate Constraints, 2022, 1–15.
- Patel, P.G., Carver, N. ve Rahimi, S. (2011). Tuning computer gaming agents using q-learning. *2011 Federated Conference on Computer Science and Information Systems (FedCSIS)* içinde (ss. 581–588). IEEE.
- Peng, J. ve Williams, R.J. (1994). Incremental multi-step Q-learning. *Machine Learning Proceedings 1994* içinde (ss. 226–232). Elsevier.
- Rajalakshmi, T. ve Sathappan, S. (2020). Incorporation of Reinforcement Learning and Compression of Deeprt for Accurate and Fast Classification of Cps Data, 7(3), 1366–1375.
- Reynolds, C.W. (1987). Flocks, herds and schools: A distributed behavioral model. *Proceedings of the 14th annual conference on Computer graphics and interactive techniques* içinde (ss. 25–34).
- Savkin, A.V, Wang, C., Baranzadeh, A., Xi, Z. ve Nguyen, H. T. (2016). Distributed formation building algorithms for groups of wheeled mobile robots. *Robotics and Autonomous Systems*, 75, 463–474.

- Schulman, J., Levine, S., Abbeel, P., Jordan, M. ve Moritz, P. (2015). Trust region policy optimization. *International conference on machine learning* içinde (ss. 1889–1897).
- Senanayake, M., Senthoooran, I., Barca, J.C., Chung, H., Kamruzzaman, J. ve Murshed, M. (2016). Search and tracking algorithms for swarms of robots: A survey. *Robotics and Autonomous Systems*, 75, 422–434.
- Sewak, M., Sahay, S.K. ve Rathore, H. (2020). Value-approximation based deep reinforcement learning techniques: an overview. *2020 IEEE 5th International Conference on Computing Communication and Automation (ICCCA)* içinde (ss. 379–384). IEEE.
- Sewak, M., Sahay, S.K. ve Rathore, H. (2021). Deep Reinforcement Learning for Cybersecurity Threat Detection and Protection: A Review. *International Conference On Secure Knowledge Management In Artificial Intelligence Era* içinde (ss. 51–72). Springer.
- Shinners, P. (2011). Pygame. *Dostupné z: <http://pygame.org/>*[Online].
- Spano, S., Cardarilli, G.C., Di Nunzio, L., Fazzolari, R., Giardino, D., Matta, M., ... Re, M. (2019). An efficient hardware implementation of reinforcement learning: The q-learning algorithm. *IEEE Access*, 7, 186340–186351.
- Stone, P. ve Veloso, M. (2000). Multiagent systems: A survey from a machine learning perspective. *Autonomous Robots*, 8(3), 345–383.
- Sutton, Richard S ve Barto, A. G. (1998). Reinforcement learning: an introduction MIT Press. *Cambridge, MA*, 22447.
- Sutton, Richard S ve Barto, A.G. (2018). *Reinforcement learning: An introduction*. MIT press.
- Sutton, Richard S. (1988). Learning to predict by the methods of temporal differences. *Machine Learning*, 3(1), 9–44. doi:10.1007/BF00115009
- Sutton, Richard Stuart. (1984). Temporal credit assignment in reinforcement learning. University of Massachusetts Amherst.
- Szepesvári, C. ve Smart, W.D. (2004). Interpolation-based Q-learning. *Proceedings of the twenty-first international conference on Machine learning* içinde (s. 100).
- Tan, Y. ve Zheng, Z. (2013). Research advance in swarm robotics. *Defence Technology*, 9(1), 18–39.
- Tsitsiklis, J.N. (1994). Asynchronous stochastic approximation and Q-learning. *Machine learning*, 16(3), 185–202.
- Vincent, P., Larochelle, H., Bengio, Y. ve Manzagol, P.-A. (2008). Extracting and composing robust features with denoising autoencoders. *Proceedings of the 25th international conference on Machine learning* içinde (ss. 1096–1103).
- Wang, Y.-H., Li, T.-H. S. ve Lin, C.-J. (2013). Backward Q-learning: The combination of Sarsa algorithm and Q-learning. *Engineering Applications of Artificial Intelligence*, 26(9), 2184–2193.

- Watkins, Christopher J.C.H ve Dayan, P. (1992a). Q-Learning. *Machine learning*, 8(3–4), 279–292.
- Watkins, Christopher J.C.H ve Dayan, P. (1992b). Q-Learning. *Machine learning*, 8(3–4), 279–292.
- Watkins, Christopher John Cornish Hellaby. (1989). Learning from delayed rewards.
- Weiss, G. (1999). *Multiagent systems: a modern approach to distributed artificial intelligence*. MIT press.

