T.C
DOKUZ EYLÜL ÜNİVERSİTESİ
FEN FAKÜLTESİ
BİLGİSAYAR BİLİMLERİ BÖLÜMÜ

BİR BİLGİSAYAR OYUNU İÇİN ÇEŞİTLİ YAPAY ZEKA ALGORİTMALARININ UYARLANMASI VE ANALİZİ

İpek SOYDEMİR

Aslı FERİKLİOĞLU

Aslıhan AKBIYIK

Emrecan BALGÜN

Danışman: Doç. Dr. METE EMİNAĞAOĞLU

Mayıs, 2022

İZMİR

İpek SOYDEMİR, Aslıhan AKBIYIK, Aslı FERİKLİOĞLU, Emrecan BALGÜN tarafından **Doç. Dr. METE EMİNAĞAOĞLU** yönetiminde hazırlanan **BİR BİLGİSAYAR OYUNU İÇİN ÇEŞİTLİ YAPAY ZEKA ALGORİTMALARININ UYARLANMASI VE ANALİZİ** başlıklı rapor tarafımızca okunmuş, kapsamı ve niteliği açısından bir Bitirme Projesi olarak kabul edilmiştir.

Doç. Dr.
METE EMİNAĞAOĞLU

Öncelikle bu projenin her aşamasında bizden hiçbir yardımını esirgemeden yol gösteren değerli danışman hocamız Associate Professor Dr. METE EMİNAĞAOĞLU ’na bize kattığı değerler için şükranlarımızı sunarız. Her zaman olduğu gibi projemizde de maddi ve manevi desteğini bizlerden esirgemeyen sevgili ailelerimize teşekkür ederiz.

ÖZET

Çok geniş bir kullanım alanına sahip olan Yapay Zeka, günlük yaşantımıza girdiğinden beri oyun sektöründe de yaygınlaşmaktadır. Agar.io oyunu ise, stratejik karar vermede sezgisel oyun tasarımını kullanması nedeniyle internette popüler hale gelmiştir.

Bu projenin amacı Agar.io oyunu için CNN, DQN, SARSA algoritmalarını 6 farklı hiperparametre ile eğiterek, her algoritma için eğitimler sırasında ortalamada en yüksek skor almış olan eğitim modelini seçmek ve bu seçilmiş modellerin kendi aralarındaki başarısını betimleyici istatistiklerden yararlanarak performans analizini ölçmektir.

Bu projenin sonucunda 3 algoritma için de skora etki eden en kritik parametrenin 'Step' değeri olduğu gözlemlenmektedir. Ortalamada en yüksek skor alan 34.26 puan ile CNN, ardından 28.32 puan alan DQN ve 25.74 puan alan SARSA gelmektedir. Her ne kadar en yüksek skora ulaşmış olan algoritma CNN olsa da Q ve SARSA ile aralarında süre bakımından uçurum vardır. Eşit episode ve step değerinde CNN için eğitim 5 saat sürerken Q ve SARSA algoritmalarında bu süre 15 dakika civarındadır.

CNN için en iyi parametrelerin; 50 episode, 1500 step, 4 Tau, 0.8 Gamma, 0.001 Learning Rate, 32 Batch Size olduğu, DQN ve SARSA için 50 episode, 1500 step, 4 Tau, 0.8 Gamma, 0.001 Learning Rate, 32 Batch Size olduğu hesaplanmıştır.

Anahtar kelimeler: Zeka, Pekiştirmeli Öğrenme, Derin Öğrenme, Evrişimsel Sinir Ağı, Deep Q Network (Derin Q Ağı),Q Öğrenme, SARSA (Durum–Eylem–Ödül–Durum–Eylem)

ABSTRACT

Since its introduction into our daily lives, artificial intelligence, which has a wide range of applications, has also grown popular in the gaming industry. Agar.io, on the other hand, has grown in popularity owing to its use of intuitive game design for strategic decision-making.

The goal of this project is to train CNN, DQN, and SARSA algorithms with 6 different hyperparameters in order to select the training model with the highest average score during training for each algorithm and measure performance analysis using statistics that compare the success of these selected models.

As a consequence of this experiment, it has been discovered that the 'Step' value is the most important parameter impacting the score for all three methods. CNN has the highest average score of 34.26, followed by DQN with a score of 28.32, and SARSA with a score of 25.74. Although CNN gets the best score, there is a huge time difference between Q and SARSA. CNN takes 5 hours to train with equal episode and step values, but the Q and SARSA algorithms take roughly 15 minutes.

The best parameters for CNN were calculated as 50 episodes, 1500 steps, 4 Tau, 0.8 Gamma, 0.001 Learning Rate, 32 Batch Size for CNN, 50 episodes, 1500 steps, 4 Tau, 0.8 Gamma, 0.001 Learning Rate, 32 Batch Size for DQN and SARSA.

Keywords: Artificial Intelligence, Reinforcement Learning, Deep Learning, Convolutional Neural Network, Deep Q Network, Q Learning, SARSA (State-Action-Reward-State-Action)

İÇİNDEKİLER

ÇİZELGELER DİZİNİ

ŞEKİLLER DİZİNİ

1. GİRİŞ

Sosyal bilimlerde 1870'lerde ortaya çıkan deneyimsel öğrenme kuramı, öğrenmede deneyimi temele alan Dewey, öğrenme sürecinde bireylerin etkin olmasının önemini vurgulayan Lewin ve zekayı sadece doğuştan gelen bir özellik olarak görmeyip kişiler ve çevre arasındaki etkileşimin bir sonucu biçiminde nitelendiren Piaget'in çalışmalarına dayanmaktadır (Yoon, 2000:36; Kolb, 1984:20).

Makine öğrenmesi yaklaşımı olan Pekiştirmeli Öğrenmede ise, öğrenen makinemizin (ajan), çevreyle olan etkileşimlerini algılamasıyla (sensation), amaca yönelik (goal), ne yapılması gerektiğiyle ilgili bir tepki verir (action) ve bunun karşılığında sayısal bir ödül sinyali alır. Ajanımız aldığı bu ödül puanını maksimuma ulaştırmak için çalışır.

Projemize başlarken öncelikle hangi makine öğrenimi algoritmalarını kullanacağımıza karar vererek ve kullanacağımız algoritmalar hakkında bilgi edinerek başladık. Ardından bu algoritmaları kurallarını kendimizin koyduğu bir oyunda gerçeklemektense hali hazırda kuralları oturmuş bir oyunda denemenin daha doğru olacağına karar verdik. Bu sürecin sonunda bizim için daha kullanışlı olduğu için Agario oyununu seçtik.

2. OYUN

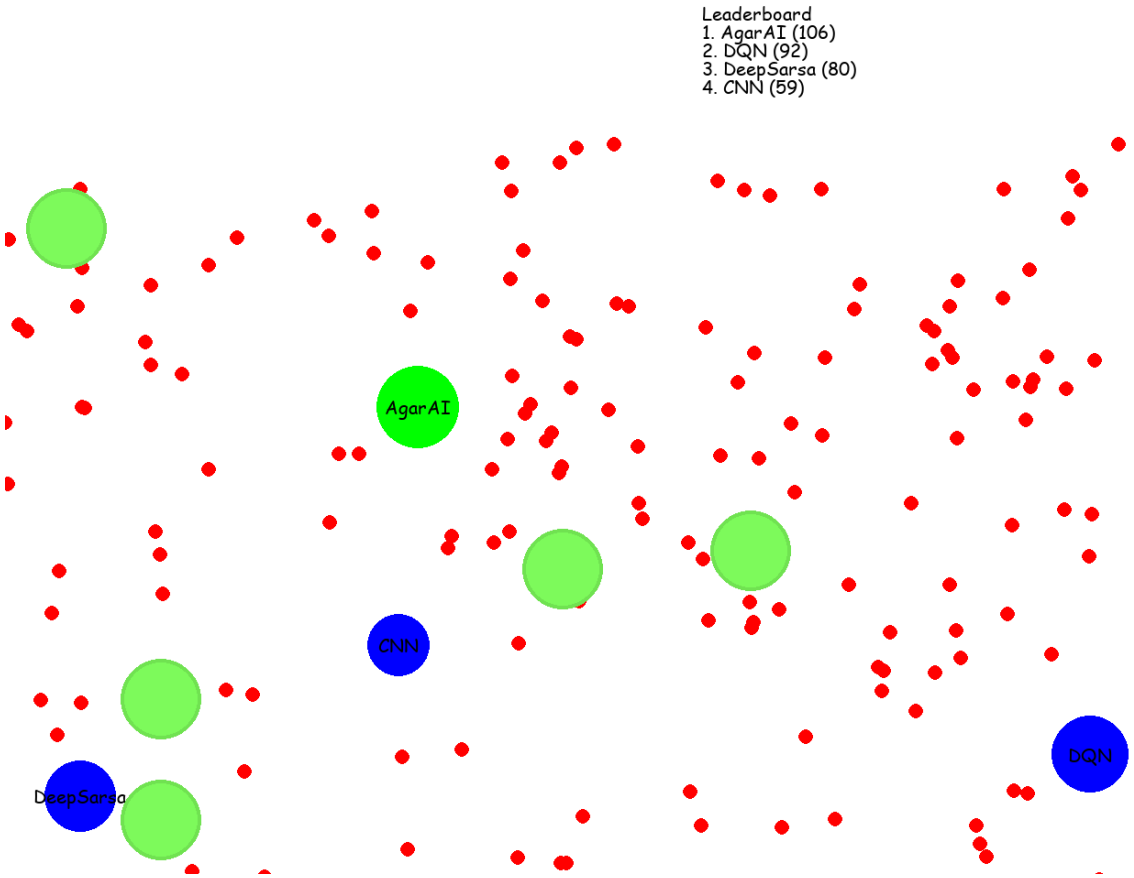
Çok oyunculu bir strateji ve aksiyon oyunu olan Agar.io'nun amacı, bir Petri(kültür) kabında oyuncu tarafından kontrol edilen bir hücrenin kütlesini artıran rastgele oluşturulmuş peletleri(agar(su yosunlarından elde edilmiş jelatin)) ve daha da büyük hücreler tarafından yenilmeden daha küçük hücreleri yutarak büyütülmesidir.

Oyunun amacı, en büyük hücreyi elde etmektir. Oyuncunun tüm hücreleri daha büyük oyuncular tarafından yenilirse, oyun küçük bir hücre ile yeniden başlamaktadır.

Yeşil virüslere çarpmak eşit olmayan parçalara bölünmenizi sağlayacağından oyunda lider tablosunda geçirilen süreyi uzatır.

Agent eats other if:

1. it has mass greater by at least CONSUME_MASS_FACTOR, and 2. the agent's circle overlaps with the center of other

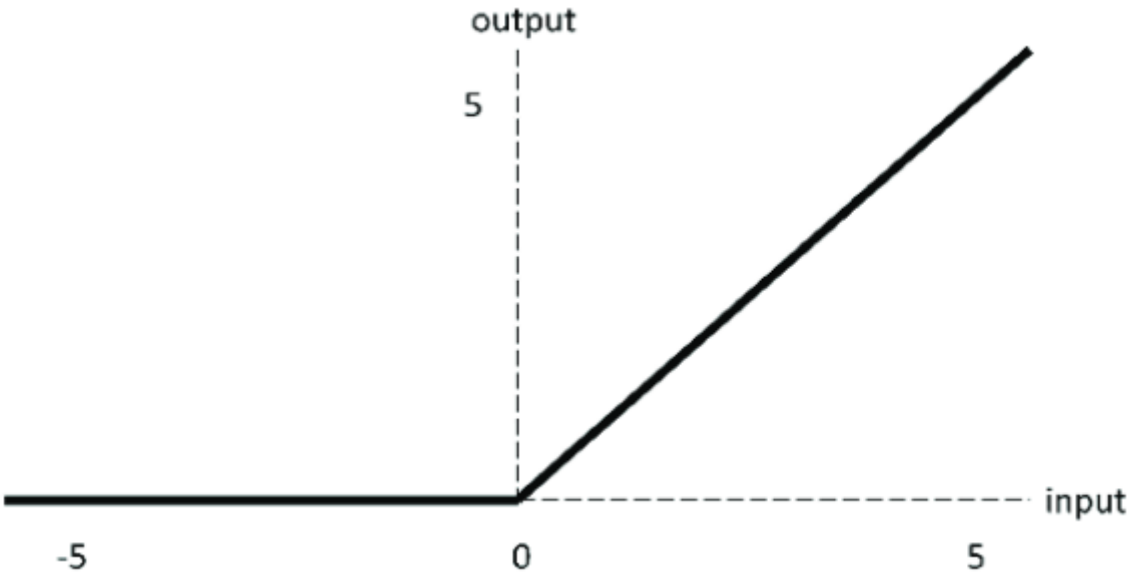


Şekil 2.1

3. YAPAY ZEKA ALGORİTMALARI

Projemizde Reinforcement Learning(Ödüllü Öğrenme) algoritmalarını kullandık.

En genel haliyle, bir pekiştirmeli öğrenme algoritmasının üç bileşeni vardır: oyundaki farklı eylemleri denemek için bir keşif stratejisi, her bir eylemin ne kadar iyi olduğuna dair geri bildirim veren bir pekiştirme işlevi ve ikisini birbirine bağlayan bir öğrenme kuralı. Bütün algoritmalarımızda, en yaygın kullanılan fonksiyon olan ReLu'yu kullandık.



Çizelge 2.1

3.1.CNN

CNN'nin mimarisi canlıların görsel algısından ilham alınarak oluşturulmuştur.Geleneksel bir evrişimli sinir ağı, tek veya birden çok evrişim ve havuzlama katmanı bloğundan, ardından bir veya birden çok tam bağlı (FC) katmandan ve bir çıktı katmanından oluşur. Evrişim katmanı, bir CNN'nin temel yapı taşıdır. Bu katman, girdinin özellik temsillerini öğrenmeyi amaçlar. Evrişim katmanı, farklı özellik haritalarını hesaplamak için kullanılan birkaç öğrenilebilir evrişim çekirdeği veya filtreden oluşur. Her birim özellik haritası, önceki katmandaki bir alıcı alana bağlanır.

```
for ( b = 0; b < B; b++ ) { // B: number of images in a batch        Outer loops
  for ( h = 0; h < H; h += T_h ) { // H: height of ofms              (off-chip data access)
    for ( w = 0; w < W; w += T_w ) { // W: width of ofms
      for ( j = 0; j < J; j += T_j ) { // J: depth of ofms
        for ( i = 0; i < I; I += T_i ) { // I: depth of ifms & wghs
          // load ifms, wghs, and ofms
          for ( p = 0; p < P; p++ ) { // P: height of wghs          Inner loops
            for ( q = 0; q < Q; q++ ) { // Q: width of wghs         (on-chip processing)
              for ( h_x = h; h_x < min(h_x+T_h, H); h_x++ ) {
                for ( w_x = w; w_x < min(w_x+T_w, W); w_x++ ) {
                  for ( j_x = j; j_x < min(j_x+T_j, J); j_x++ ) {
                    for ( i_x = i; i_x < min(i_x+T_i, I); i_x++ ) {
                      ofms [b][h_x][w_x][j_x] += wghs [p][q][i_x][j_x] *
                                                 ifms [b][str*h_x+p][str*w_x+q][i_x]
          }}}}}}
          // store ofms
}}}}}
```

Şekil 3.1.1

```python
def preprocess_state(self, state):
    # convert RGB to grayscale via relative luminance
    gray_state = np.dot(state[...,:3], [0.299, 0.587, 0.114])
    # size down the image to speed up training
    resized_state = transform.resize(gray_state, self.downsample_size, mode='constant')
    return resized_state
```

Şekil 3.1.2

3.2.Deep Q Network

Deep Q Network, Q-Learning ile Neural Network algoritmasının birleşiminden oluşmaktadır.

3.2.1 Q Learning *(Ai for games kitap, sayfa 658)
Q-Learning, bir Q-fonksiyonu kavramına dayanmaktadır.
Q-Learning, her olası durum ve eylem hakkında tuttuğu kalite bilgisi (Q-değerleri) kümesi için adlandırılır. Algoritma, denediği her durum ve eylem için bir değer tutar. Q değeri, o durumdayken eylemin ne kadar iyi olduğunu düşündüğünü temsil eder.

$$Q(s,a) = (1-\alpha)\,Q(s,a) + \alpha\left(r + \gamma \max\left(Q(s',a')\right)\right)$$

Şekil 3.2.1.1

Q-öğrenme kuralı, doğrusal karışımı kontrol etmek için öğrenme oranı parametresini kullanarak iki bileşeni bir araya getirir. Karışımı kontrol etmek için kullanılan learning rate parametresi [0, 1] aralığındadır.

İlk bileşen Q(s,a), durum ve eylem için mevcut Q değeridir. Mevcut değerin bir kısmını bu şekilde tutmak, daha önce keşfettiğimiz bilgileri asla çöpe atmamamız anlamına gelir.

Genel bir Q-öğrenme sistemi aşağıdaki yapıya sahiptir:

```
# Holds the store for Q-values, we use this to make
# decisions based on the learning
store = new QValueStore()

# Updates the store by investigating the problem
def QLearning(problem, iterations, alpha, gamma, rho, nu):

  # Get a starting state
  state = problem.getRandomState()

  # Repeat a number of times
  for i in 0..iterations:

    # Pick a new state every once in a while
    if random() < nu: state = problem.getRandomState()

    # Get the list of available actions
    actions = problem.getAvailableActions(state)

    # Should we use a random action this time?
    if random() < rho:
      action = oneOf(actions)

    # Otherwise pick the best action
    else:
      action = store.getBestAction(state)

    # Carry out the action and retrieve the reward and
    # new state
    reward, newState = problem.takeAction(state, action)

    # Get the current q from the store
    Q = store.getQValue(state, action)

    # Get the q of the best action from the new state
    maxQ = store.getQValue(newState,
                           store.getBestAction(newState))

    # Perform the q learning
    Q = (1 - alpha) * Q + alpha * (reward + gamma * maxQ)

    # Store the new Q-value
    store.storeQValue(state, action, Q)

```

Şekil 3.2.1.2 (Ai-for-games.pdf sayfa 660)

3.2.2 Deep Q Network

Deep Q-Learning, klasik Q-Learning algoritmasının temel katkısı olan bir çeşididir.
(1) Q-fonksiyonu yaklaşımı için derin bir evrişimli sinir ağı mimarisi;
(2) son deneyime ilişkin tek adımlı güncellemeler yerine rastgele eğitim verilerinin mini batch'leri kullanmak;
(3) sonraki durumun Q-değerlerini tahmin etmek için daha eski ağ parametrelerinin kullanılması.

DQN için kaba kod, Algoritma 1'de gösterilmektedir.

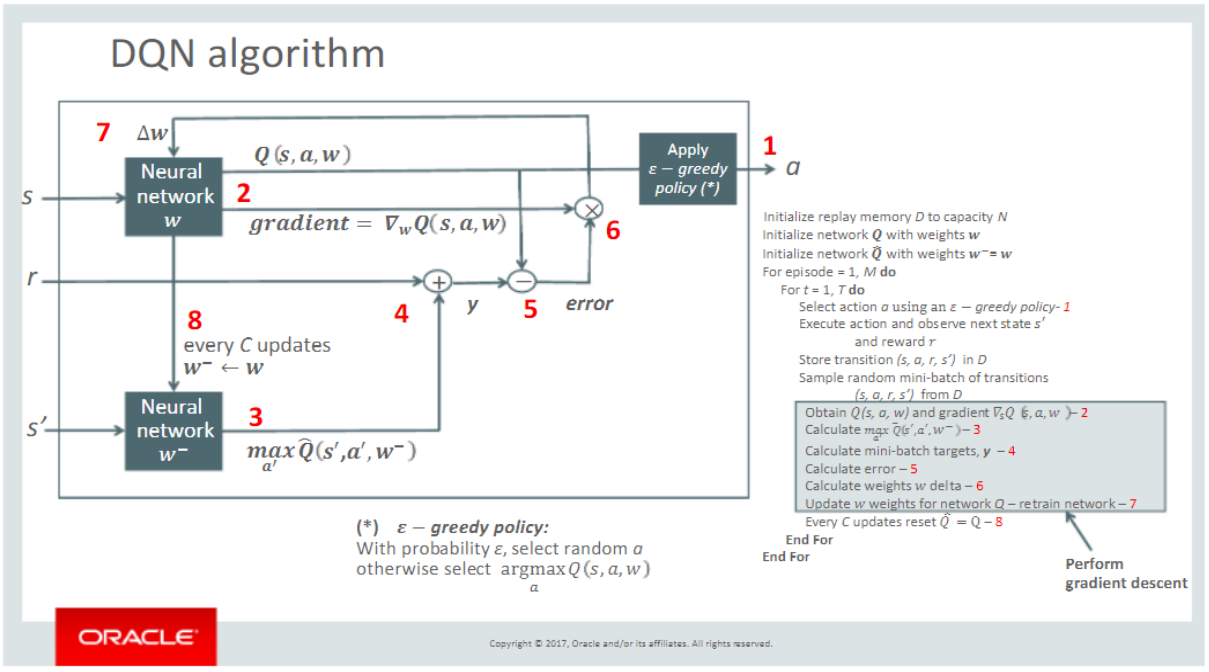
Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
  end for
end for

Şekil 3.2.2.1

```
# do Q computation
currQ = self.model(states).gather(
    1, actions.unsqueeze(1))
nextQ = self.target_net(next_states)
max_nextQ = torch.max(nextQ, 1)[0].detach()
mask = 1 - dones
expectedQ = rewards + mask * self.gamma * max_nextQ
```

Şekil 3.2.2.2



Şekil 3.2.2.3

3.3.SARSA

SARSA, Q-Learning gibi Policy-tabanlı bir destekleyici öğrenme metodudur. Baş harfleri State-Action-Reward-State-Action olmak üzere destekleyici öğrenme sürecinde eğitilmek için kullanılan ajanların mevcut perception durumunu kontrol eden ve çevresel algıların işlenmesine bağlı anlık bir durumu ifade eden "State/Durum", hareket ve eylemleri belirten "Action/Hareket", ve ajana gerçekleştirdiği anlık davranışlardan dolayı verilen ödülü temsil eden "Reward/Ödül" parametrelerinden meydana gelmektedir. SARSA'nın en temel amacı seçilen politikaya dayalı olarak bir sonraki adımda meydana gelecek State ve Action'ları hesaplayabilmektir.

Q-Learning ve SARSA birbirine benzer algoritmalar gibi gözükse de aralarında belli başlı farklılıklar bulunmaktadır. Bunlardan en önemlisi SARSA'da Q-Learning'in aksine Q değerlerini güncellemek için hesaplanan maksimum bir ödül gereksinimi kullanılmamasıdır. SARSA'da temel eylemi belirleyen sabit bir politika kullanılarak yeni bir eylem ve ödül seçimi gerçekleştirilir.

Aşağıda SARSA algoritmasının matematiksel olarak ifade edilmiş biçimini görebiliriz:

$$Q_t(s,a) = (1-\alpha)Q_t(s,a) + \alpha(R_{t+1}(s,a) + \gamma Q_{t+1}(s,a)$$

Şekil 3.3.1

Burada da gözlenebileceği gibi SARSA'nın temel matematiksel formülasyonunda Q-Learning'in aksine maksimum ödülün önceliklendirmesini kontrol edecek herhangi bir parametre bulunmamaktadır. Bu sebeple Q-Learning'e kıyasla daha farklı ve belirli durumlarda Q-Learning ile kıyaslandığında daha avantajlı performans sağlayabilecek use-case'ler barındırmaktadır. Fakat belirli bazı başka durumlarda Q-Learning'in SARSA'ya kıyasla daha yüksek performans ve verimlilik sağlaması mümkündür.

Yapı ve Matematiksel Formülasyon:
Qt (s, a): (t) anındaki Q değerini ifade etmektedir.
(1 - α), α: Learning Rate, yani öğrenme oranı olarak tanımlanmaktadır. Bir step'te mevcut state ile yeni geliştirilen state arasındaki farkı tanımlamak için kullanılmaktadır. Yüksek learning rate, t ve t+1, yani takipli iki state arasındaki daha yüksek farklılıkları ifade etmektedir.
Rt+1(s, a): Bir sonraki gerçekleştirilen eylem için hesaplanan ödül değeridir.
γ: Discount Factor, yani indirim faktörü olarak ifade edilmektedir. 0 ile 1 arasında bir değer alır. 0, gelecek steplerdeki ödüllerin önemsizliğini, 1 ise gelecek steplerdeki potansiyel ödüllerin t anındaki ödüller kadar önemli olduğunu belirtmektedir.
Qt+1(s, a): Bir sonraki state için Q değerini ifade etmektedir.

Q-Learning ve SARSA Farklılıkları:
- Q-Learning: Önce durumları ve ödülleri kontrol et, sonra davranışta bulun.

- SARSA: Önce davran. Sonra durumu kontrol et ve buna göre görüşlerini yeniden ayarla.
- Q-Learning: Bir sonraki durum için diğer durumları kontrol et ve maksimum ödüle yönelik bir eğilimde bulun.
- SARSA: Bir eylemi gerçekleştirmek için kararda bulun, bir önceki durumu gerçek değerden faydalanarak güncelle.
- Q-Learning: t+1 anındaki potansiyel ödül en iyi mümkün eylemdir.
- SARSA: t anındaki adımın t+1 anındaki ödülü, eylem için gerçeklenen esas ödüldür.
- Q-Learning: Off-Policy reinforcement learning yöntemidir. Davranış üzerine meta-cognition olmadan optimal Q değerleri geliştirilmeye çalışılır.
- SARSA: On-Policy reinforcement learning yöntemidir. Q değerleri uygun aksiyonların sağlanması adına geliştirilir.
- Q-Learning: Bu metodolojide ajan önce keşifte bulunur, sonra karar verir.

Kullanım Alanları:

SARSA, öğrenme sürecinde ajanın performansı önemseniyorsa uygun bir seçim olabilir. Hataların bedeli yüksek ise bu yöntem tercih edilebilir. Ancak, eğitim sürecinde performans çok önemli bir yere sahip değilse, Q-Learning algoritmasını tercih etmek daha mantıklı olabilmektedir.

Aşağıda SARSA algoritmasının psödo-koduna ilişkin bir figür paylaşılmıştır:

```
Initialize Q(s, a) arbitrarily
Repeat (for each episode):
    Initialize s
    Choose a from s using policy derived from Q
          (e.g., ε-greedy)
    Repeat (for each step of episode):
       Take action a, observe r, s'
       Choose a' from s' using policy derived from Q
          (e.g., ε-greedy)
       Q(s, a) <-- Q(s, a) + α[r + γQ(s', a')- Q(s, a)]
       s <-- s'; a <-- a';
    until s is terminal
```

Şekil 3.3.2

Hangi destekleyici öğrenme yönteminin seçileceğinin esasında birçok faktöre dayalı ve önemli bir karar olduğu unutulmamalıdır. Her durum için eşdeğer düzeyde başarılı tek bir yöntemden söz etmek mümkün değildir ve her yöntem kendisine göre avantaj sağlayacağı belirli kullanım alanları bulundurabilmektedir. Bu sebeple proje dahilinde de hem Q-Learning hem de SARSA kullanılarak çeşitli modeller oluşturulmuş ve bu modellerin performansları ayrı ayrı test edilerek kıyaslanmıştır. Bu kıyaslanmalar sayesinde proje dahilinde işlenen uygulama alanında Q-Learning ve SARSA'nın birbirine göre ne derece farklı sonuçlar ürettiği daha etkin incelenebilecektir.

```
# do Q computation
currQ = self.model(states).gather(
    1, actions.unsqueeze(1))
nextQ = self.target_net(next_states)
mean_nextQ = torch.mean(nextQ, 1)[0].detach()
mask = 1 - dones
expectedQ = rewards + mask * self.gamma * mean_nextQ
```

Şekil 3.3.3

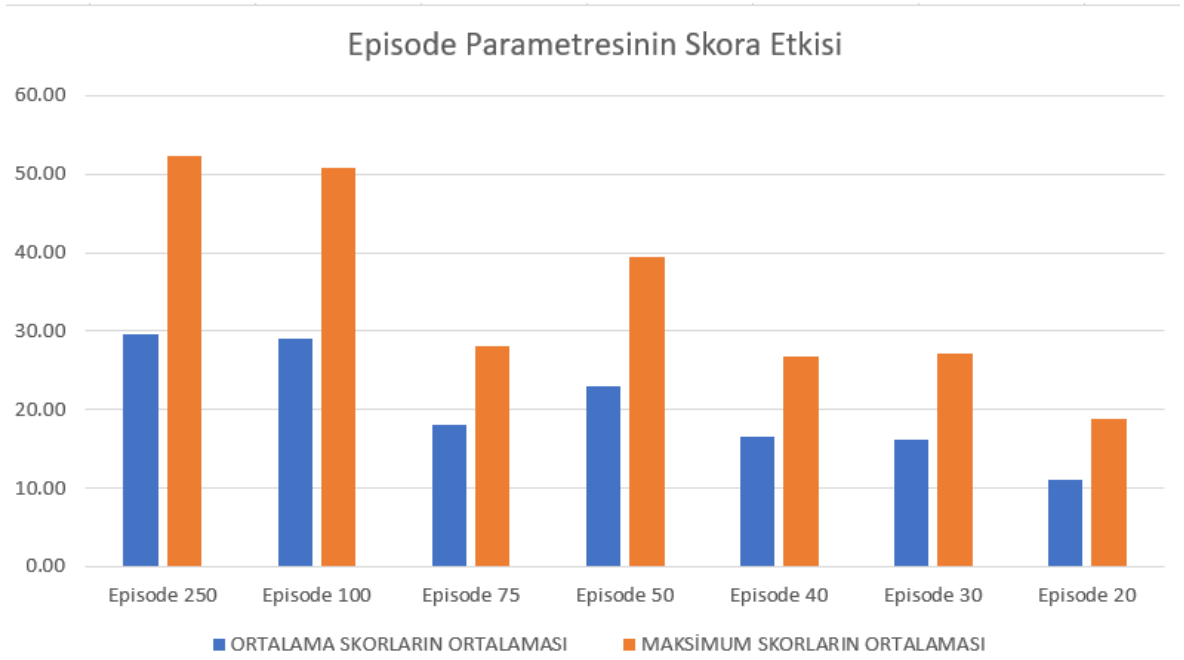
ALGORİTMALARIN EĞİTİMİ

Her algoritma için 6 parametre ve her parametre için 7 farklı değerde ele alacak şekilde toplam 126 eğitim gerçekleştirdik.

3.4. CNN

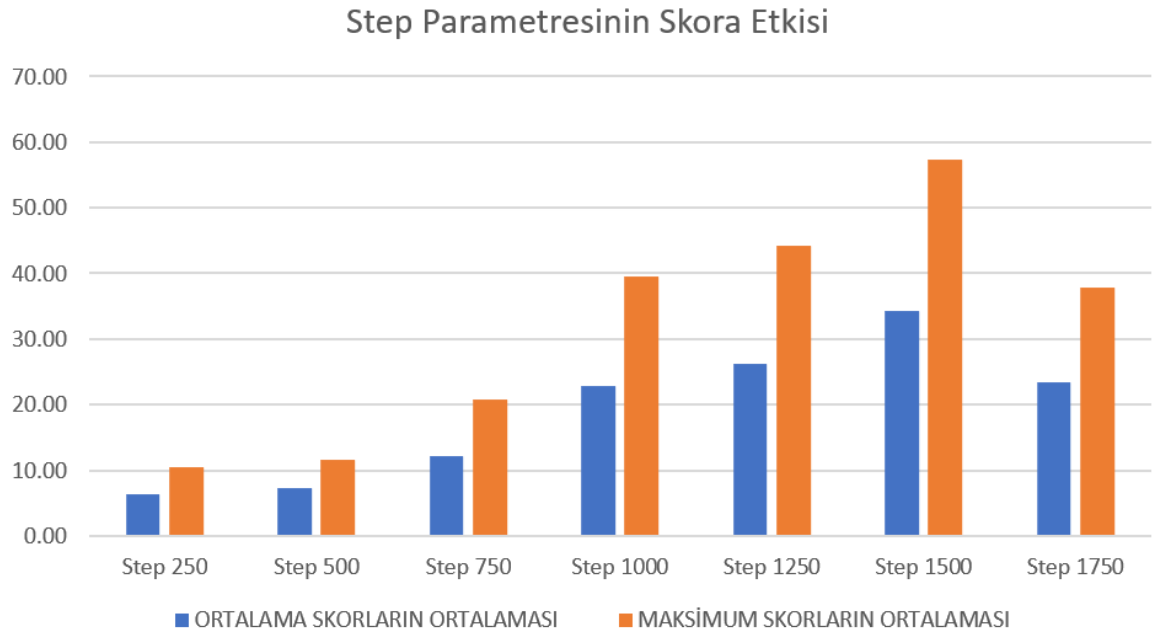
CNN	**Kullanılan Değerler**						
Episode	20	30	40	50	75	100	250
Step	250	500	750	1000	1250	1500	1750
Tau	1	2	3	4	5	6	7
Gamma	0.5	0.6	0.7	0.8	0.9	1.0	1.1
Learning Rate	0.0001	0.001	0.002	0.003	0.004	0.005	0.010
Batch Size	29	30	31	32	33	34	35

Epoch(Episode) Parametresi
Bu parametre belirlenen Step parametresinin kaç kez tekrar edeceğinin bilgisidir. Her tekrarda skor sıfırlanarak geçmişte öğrendiğinin üzerine koyarak ilerler. Epoch (Episode) Parametresi aynı zamanda bizim durdurma koşulumuzdur (stopping criteria). 50 ile 75 arasında overfitting gözlemlenmektedir. Zaman açısından yüksek episode değerinin çok verimsiz olması ve Epoch sayısının artması skora yeterli etkiyi sağlayamadığı için 50 Episode bitirme koşulumuz olarak seçildi.



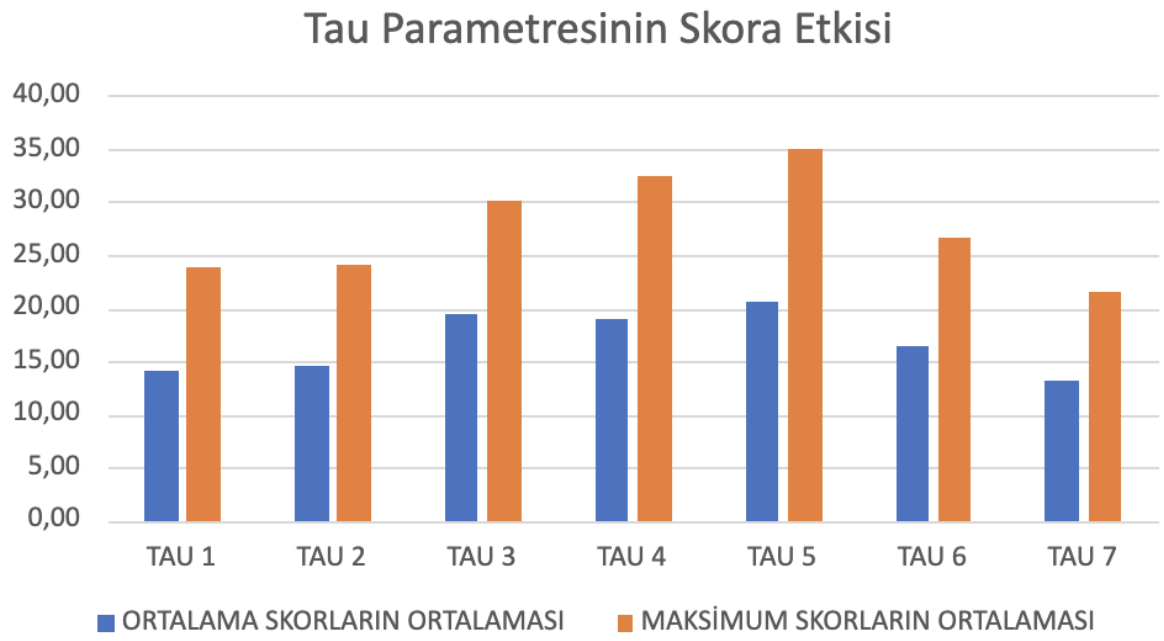
Step Parametresi

CNN algoritmasının kaç iterasyonda işlenerek 1 Epoch gerçekleştireceğinin bilgisidir. Her tekrarda skorun üzerine koyarak kümülatif ilerler. Step 1500 e kadar, step arttıkça skor arttığı görülmüştür fakat 1750 değerinde over fitting gözlemlenmiştir. 34,26 puan alarak ortalama skorların ortalamaları arasında en yüksek skora sahip değer 1500 Steptir, 6,31 puan ile ortalama skorların ortalamaları arasında en düşük skora sahip olan 250 Step'tir.



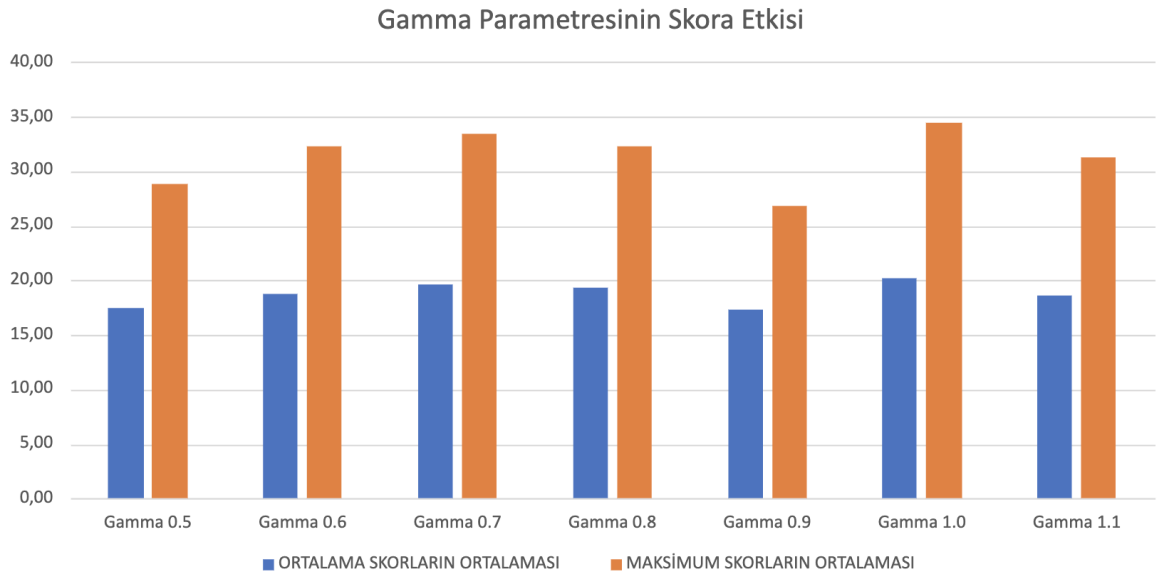
Tau Parametresi

Değişkenler arasında ilişkinin varlığını araştırmak ilişkinin gücünü ölçmek amacıyla korelasyon katsayılarından yararlanılmaktadır. Tau da bu korelasyon katsayılarından biridir. Tau katsayılarını artırırken 5'e kadar istikrarlı bir artış gözlense de 5'ten sonraki değerlerde overfitting görülmeye başlanmıştır. Maksimum skoru elde eden Tau değeri 5'tir.



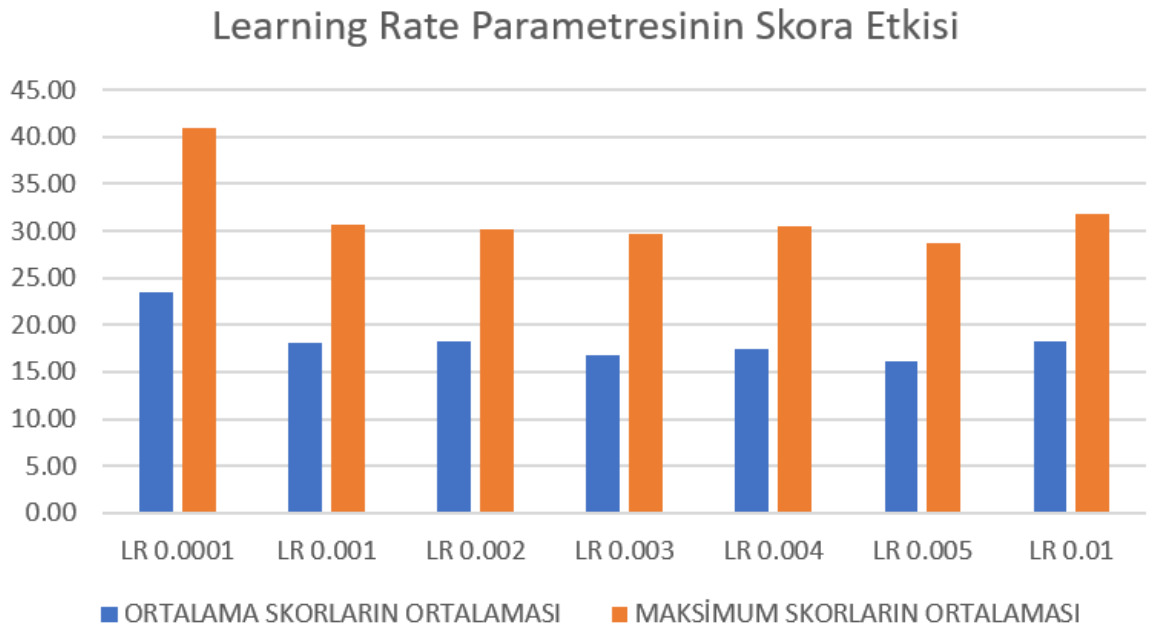
Gamma Parametresi

Başlangıç değeri olan 0.5 değerinden 0.7 değerine kadar istikrarlı bir artış gözlenmiş olsa da, bu değerden sonra azalma artmalar istikrarsız olarak gözlenmiştir.



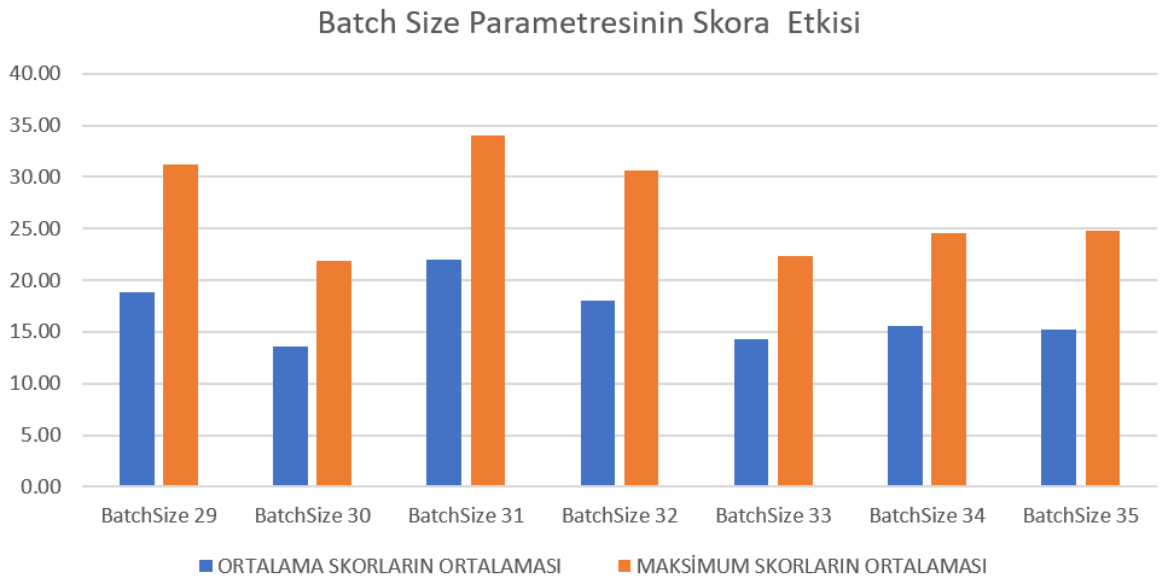
Learning Rate Parametresi

Learning Rate (öğrenme hızı), model ağırlıkları her güncellendiğinde tahmini hataya yanıt olarak modelin ne kadar değiştirileceğini denetleyen bir hiper parametredir. Değerler yakın olduğu için genel olarak birbirine benzer sonuçlar elde edilmiştir, fakat learning rate 0.0001 iken yani daha düşük bir öğrenme hızında daha iyi sonuç vermiştir. Bu yüzden en verimli learning rate değerimiz 0.0001 olmuştur.



Batch Size Parametresi

Batch size(Toplu iş boyutu), ağ üzerinden yayılacak örnek sayısını tanımlar. Genel olarak batch size ne kadar büyük olursa, modelin eğitim sırasında her epoch (episode) daha hızlı tamamlanacaktır. Bunun nedeni, hesaplama kaynaklarına bağlı olarak makinenin aynı anda birden fazla örneği işleyebilmesidir. 29 ve 30 değerleri arasında overfitting gözlenmektedir. Maksimum skora sahip değerimiz ise 31'dir.

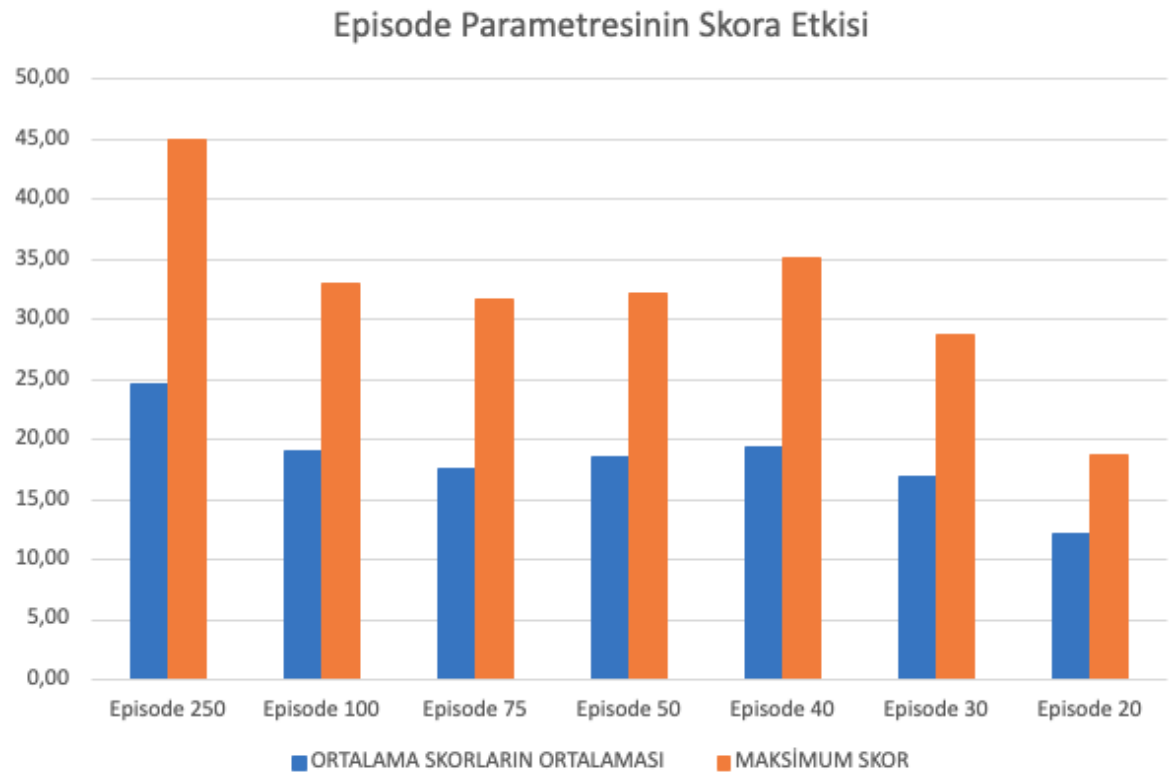


3.5.DEEP Q NETWORK

DQN	**Kullanılan Değerler**						
Episode	20	30	40	50	75	100	250
Step	250	500	750	1000	1250	1500	1750
Gamma	0.5	0.6	0.7	0.8	0.9	1.0	1.1
Learning Rate	0.0001	0.001	0.002	0.003	0.004	0.005	0.010
Batch Size	29	30	31	32	33	34	35
Epsilon	0.25	0.50	0.75	1.00	1.25	1.50	1.75

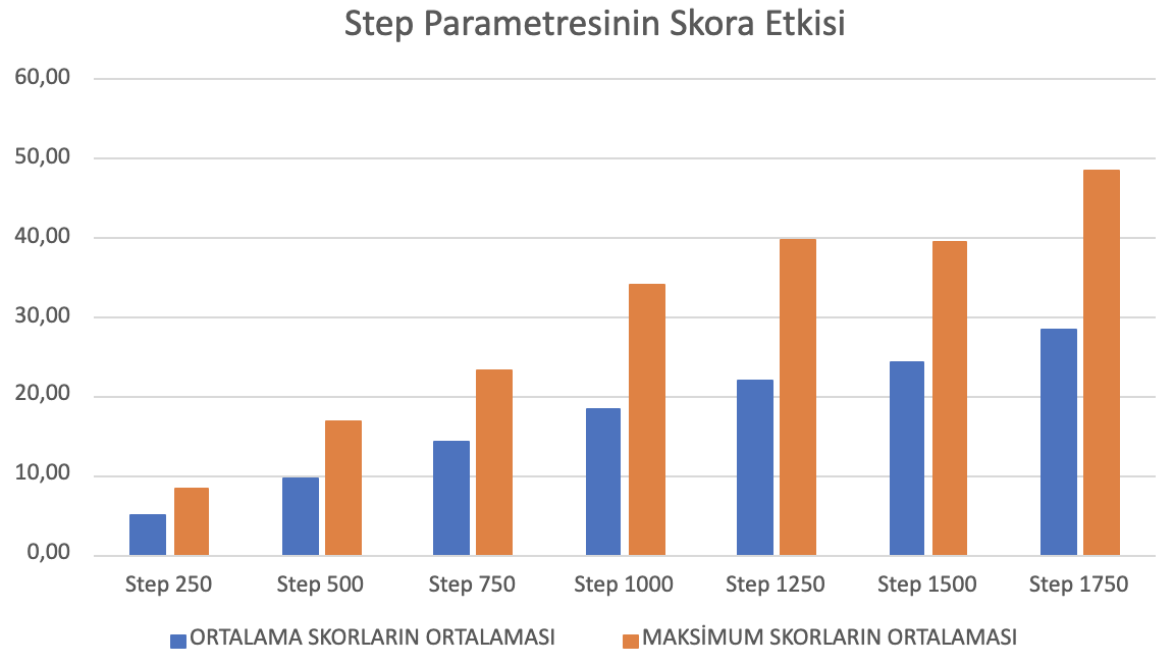
Epoch(Episode) Parametresi

Bu parametre belirlenen Step parametresinin kaç kez tekrar edeceğinin bilgisidir. Her tekrarda skor sıfırlanarak geçmişte öğrendiğinin üzerine koyarak ilerler. Epoch parametresi aynı zamanda bizim durdurma koşulumuzdur.(stopping criteria). 50 ile 250 arasında overfitting gözlenmektedir. Zaman açısından yüksek episode değerinin çok verimsiz olmasına rağmen en yüksek skoru 250 episode vermiştir.



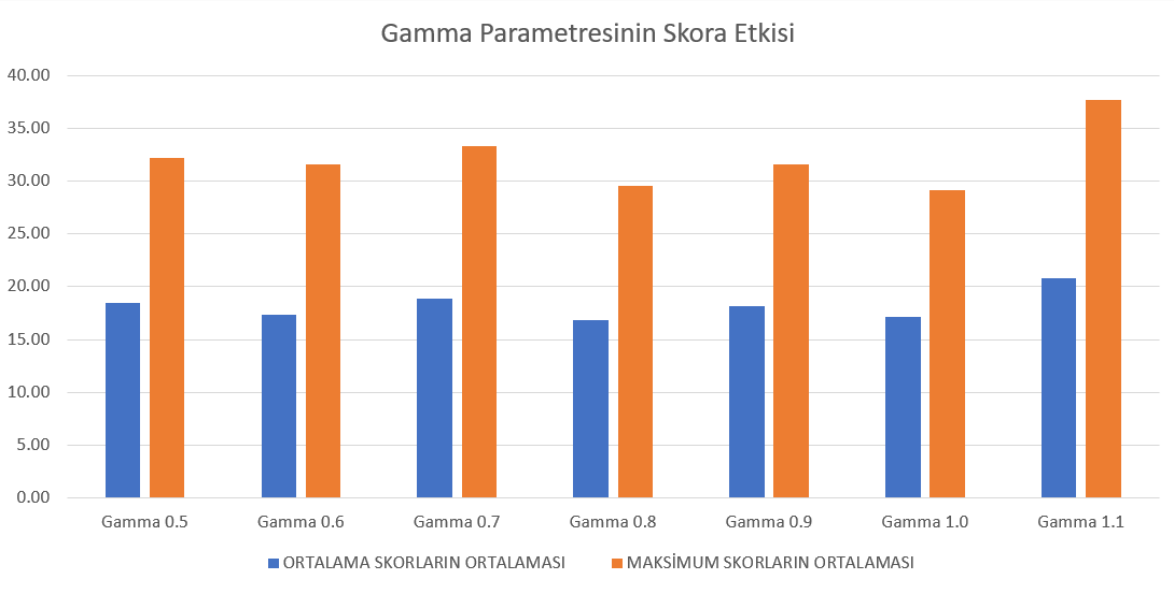
Step Parametresi

CNN algoritmasının kaç iterasyonda işlenerek 1 Epoch gerçekleştireceğinin bilgisidir. Her tekrarda skorun üzerine koyarak kümülatif ilerler. Step 1500'e kadar, step arttıkça skor arttığı görülmüştür fakat 1750 değerinde overfitting gözlemlenmiştir.



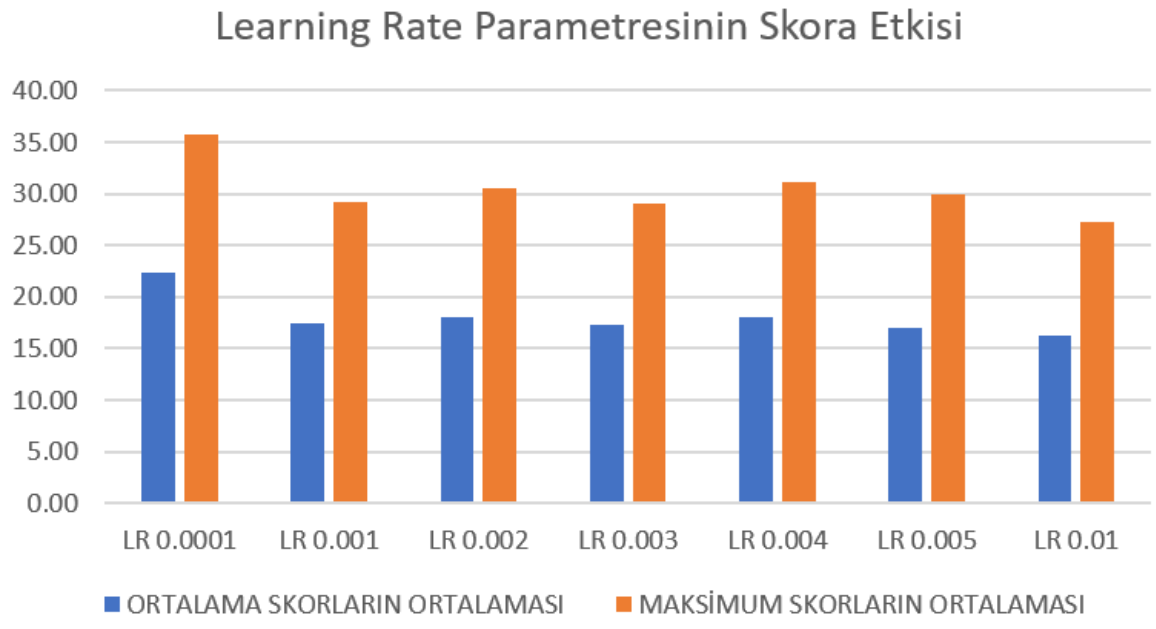
Gamma Parametresi

Gamma 0.5 de 18.45 maksiumum skora shipken 0.6 da 17.39 a düşmüştür fakat 0.7 de ise yükselme eğilimi gösterip 18.91 skor değerine ulaşmıştır. Bu parametrede bir yükselen bir düşen bir yapıya sahiptir. Gamma 1.1 değerinde maksimum skora ve ortalamaya sahiptir.



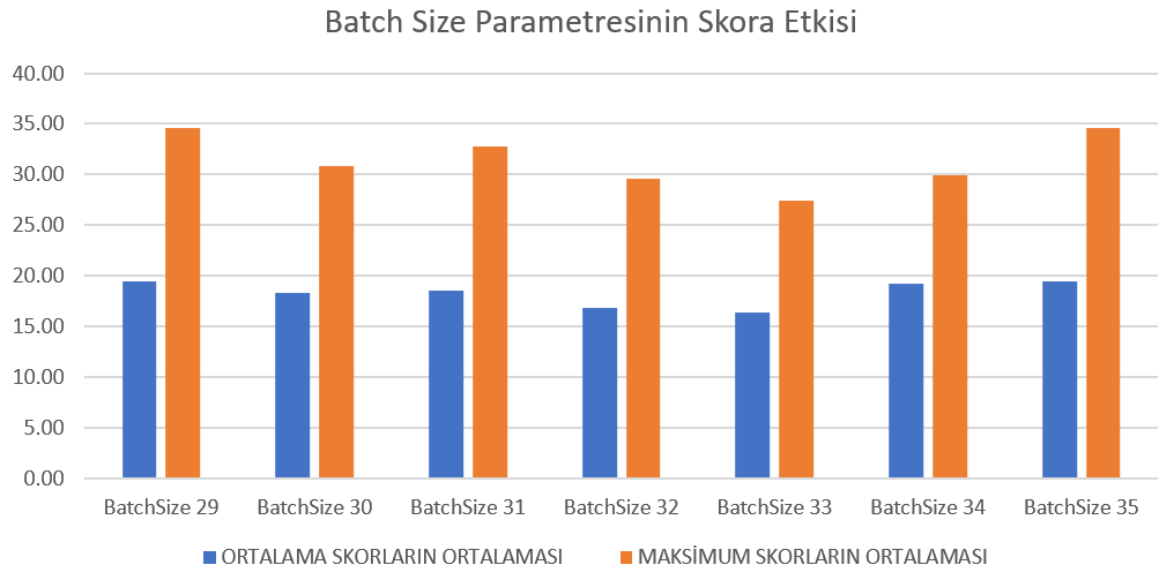
Learning Rate Parametresi

Diğer değerlere oranla gözle görülür fark yaratan 0.0001 değeri en küçük değerimiz olması sebebiyle maksimumlarda ve ortalamalarda üstün başarı sergileyerek 22,37 puan almıştır. En kötü başarıyı ise 16,22 puan alaran en büyük değerimiz 0,01 almıştır. Buradan Learning Rate değerinin küçük olması gerektiğini çıkarıyoruz.



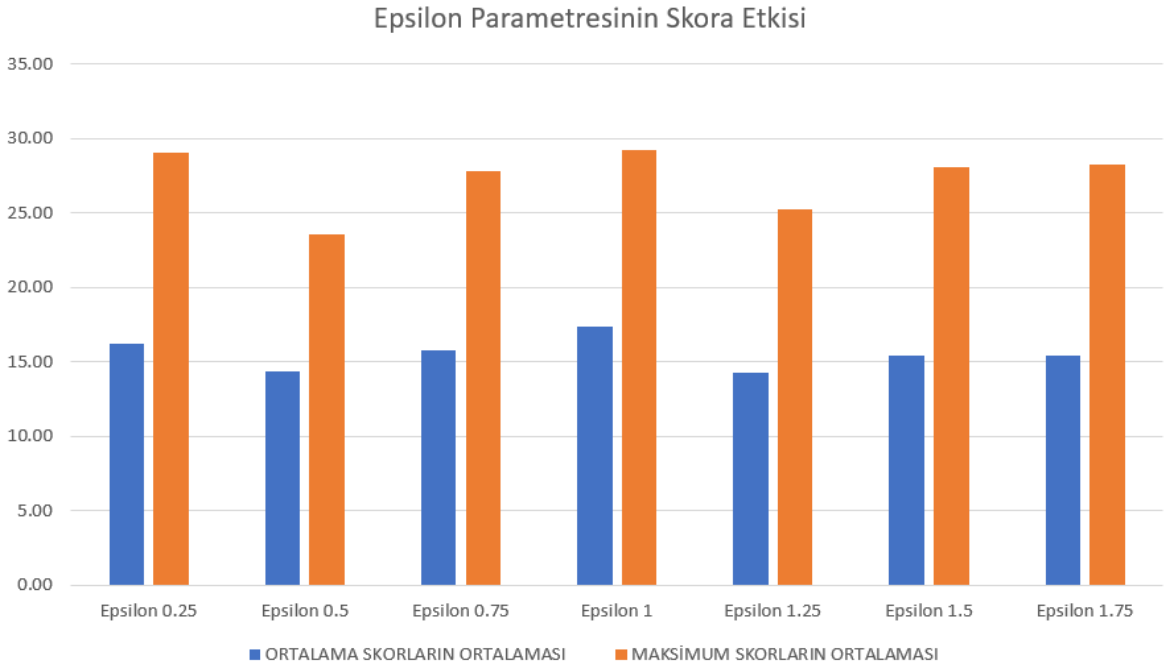
Batch Size Parametresi

Batch size 29 da maksimum skora sahipken 30'a geçtiğinde maksimum skor düşmüştür fakat 31'e geçtiğinde tekrar yükselme eğilimi göstermiştir bu nedenle genel olarak bir yükselen bir düşen bir yapıya sahiptir fakat buna rağmen birbirine yakın değerler elde edilmiştir. Batch size da 29 ve 35 değerleri aynı maksimum skora sahiptir.



Epsilon Parametresi

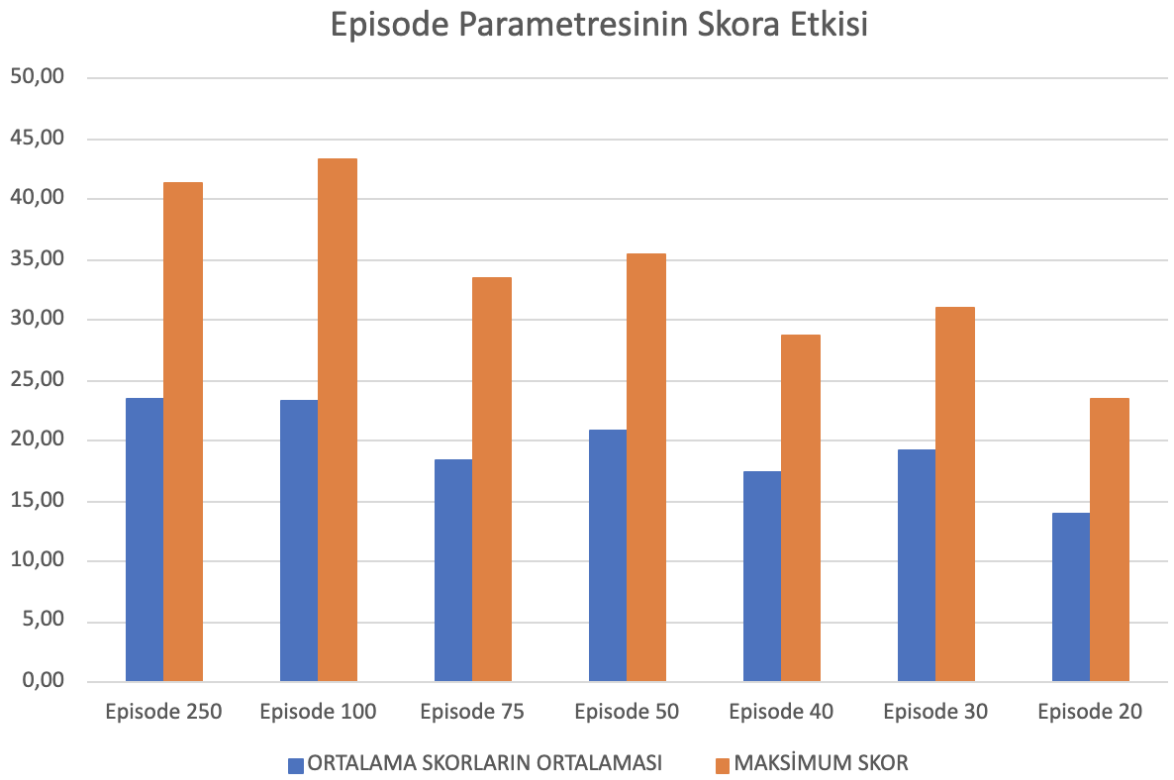
Epsilon parametresi Greedy Exploration Strategy'i kullanım miktarımızın bilgisidir. Ortalamalarda maximum seviyeye ulaşan değer 17,37 puan ile 1'dir.



3.6.DEEP SARSA

SARSA	Kullanılan Değerler						
Episode	20	30	40	50	75	100	250
Step	250	500	750	1000	1250	1500	1750
Gamma	0.5	0.6	0.7	0.8	0.9	1.0	1.1
Learning Rate	0.0001	0.001	0.002	0.003	0.004	0.005	0.010
Batch Size	29	30	31	32	33	34	35
Epsilon	0.25	0.50	0.75	1.00	1.25	1.50	1.75

Epoch(Episode) Parametresi
Bu parametre belirlenen Step parametresinin kaç kez tekrar edeceğinin bilgisidir. Her tekrarda skor
sıfırlanarak geçmişte öğrendiğinin üzerine koyarak ilerler. Epoch parametresi aynı zamanda bizim durdurma koşulumuzdur.(stopping criteria).

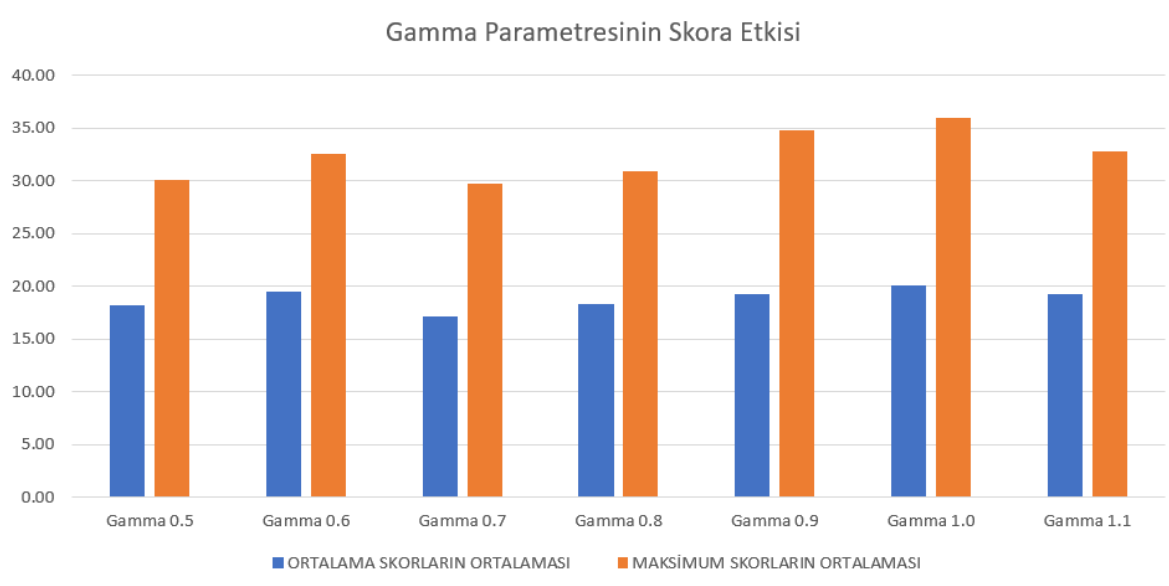


Step Parametresi

CNN algoritmasının kaç iterasyonda işlenerek 1 Epoch gerçekleştireceğinin bilgisidir. Her tekrarda skorun üzerine koyarak kümülatif ilerler. Step 1750'ye kadar step arttıkça skor arttığı gözlenmiştir.

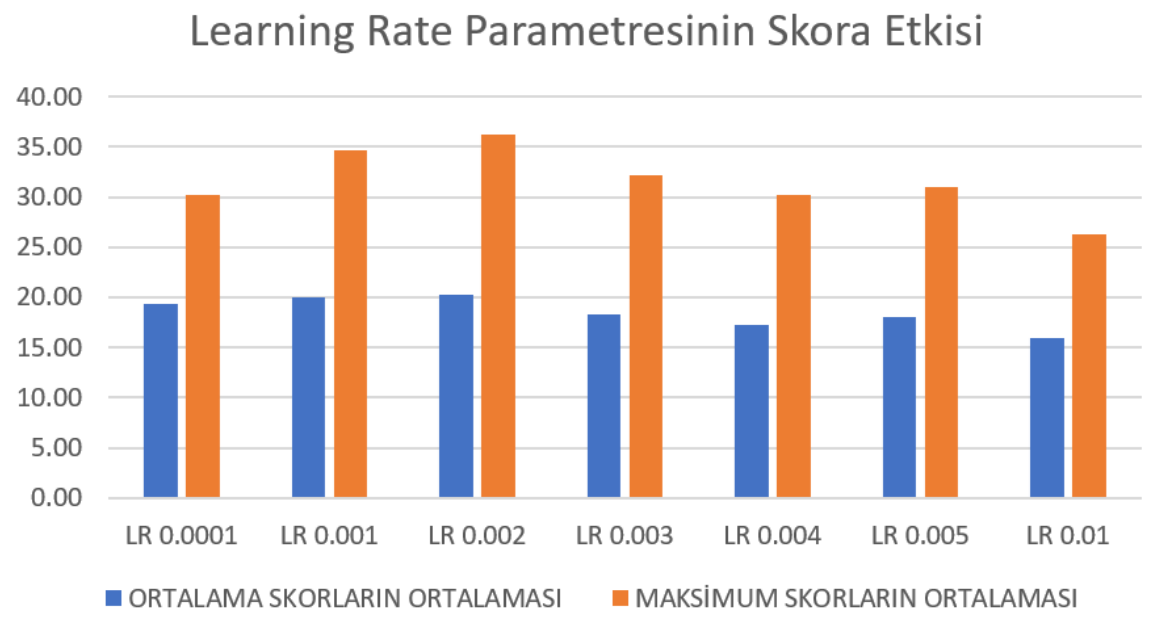
Gamma Parametresi

Gamma 0.6-0.7 ve 1.0-1.1 değerlerinde overfitting gözlemlenmektedir. 0.9 ve 1.0 değerleri birbirine çok yakın gözüksede maksimum skora 1.0 değeri sahiptir.



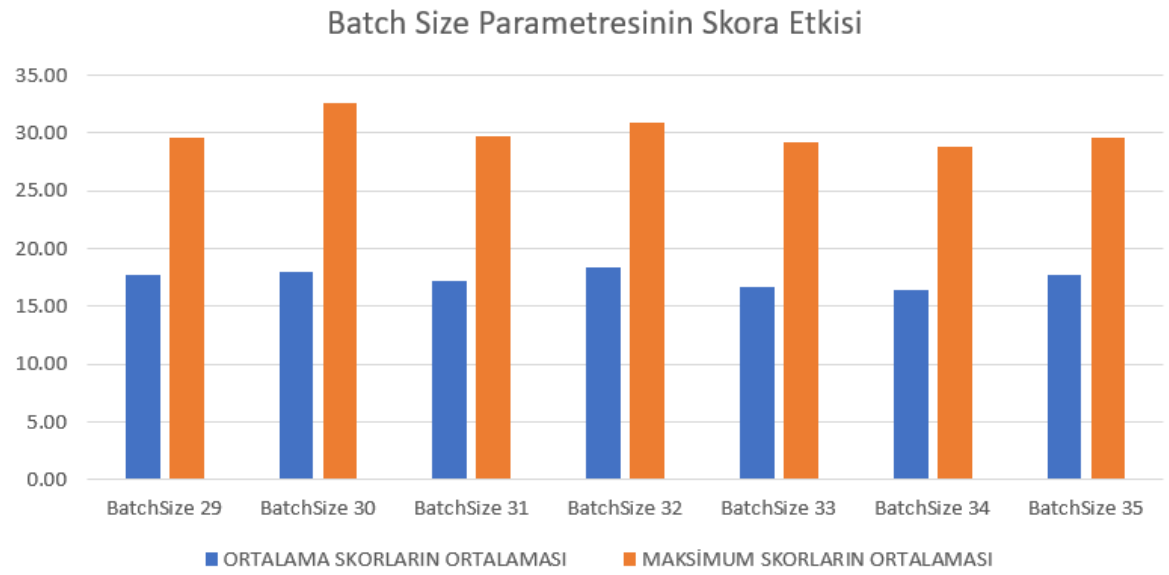
Learning Rate Parametresi

Learning rate 0.002'ye kadar skor artış göstermiş, learning rate büyüdükçe kademeli olarak skor azalmıştır.15,94 puan ile 0.01 parametresi ortalamada en düşük skora ulaşmışken 20,24 puan ile 0.002 parametresi ortalamaların ortalamasında en yüksek skora ulaşmıştır.



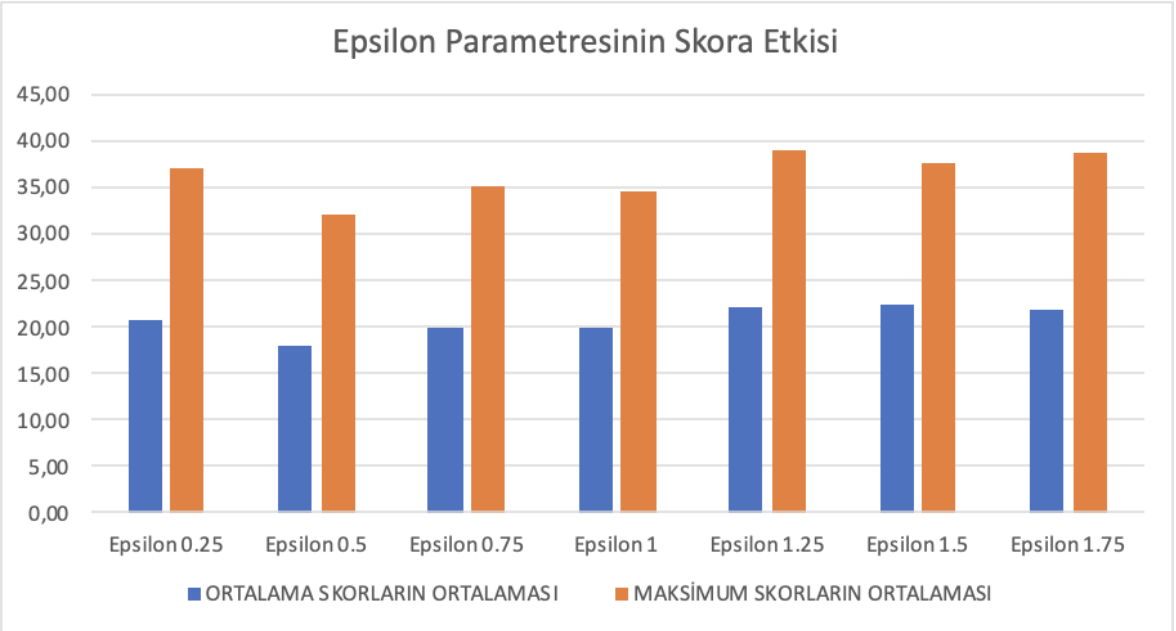
Batch Size Parametresi

Görüldüğü gibi Batch size 29-30-31-32-33 değerlerinde maksimum skor ortalaması olarak bir artan bir azalan bir yapıya sahiptir. Birbirine çok yakın değerler olsa da en iyi sonucu 32 değeri vermiştir. Maksimum skorların ortalaması olarak baktığımızda ise en iyi değeri 30 vermiştir.



Epsilon Parametresi

Bu parametre bir yükselen bir düşen bir yapıya sahiptir. Baskın bir şekilde kazanan bulunmamakla birlikte ortalama skorların ortalamasında 22,39 Skor ile diğerlerine oranla daha iyi olan Epsilon değeri 1.25'tir. 18,05 puan ile 0.05 parametresi ortalamada en düşük skora ulaşmıştır.



4. SONUÇ VE DEĞERLENDİRME

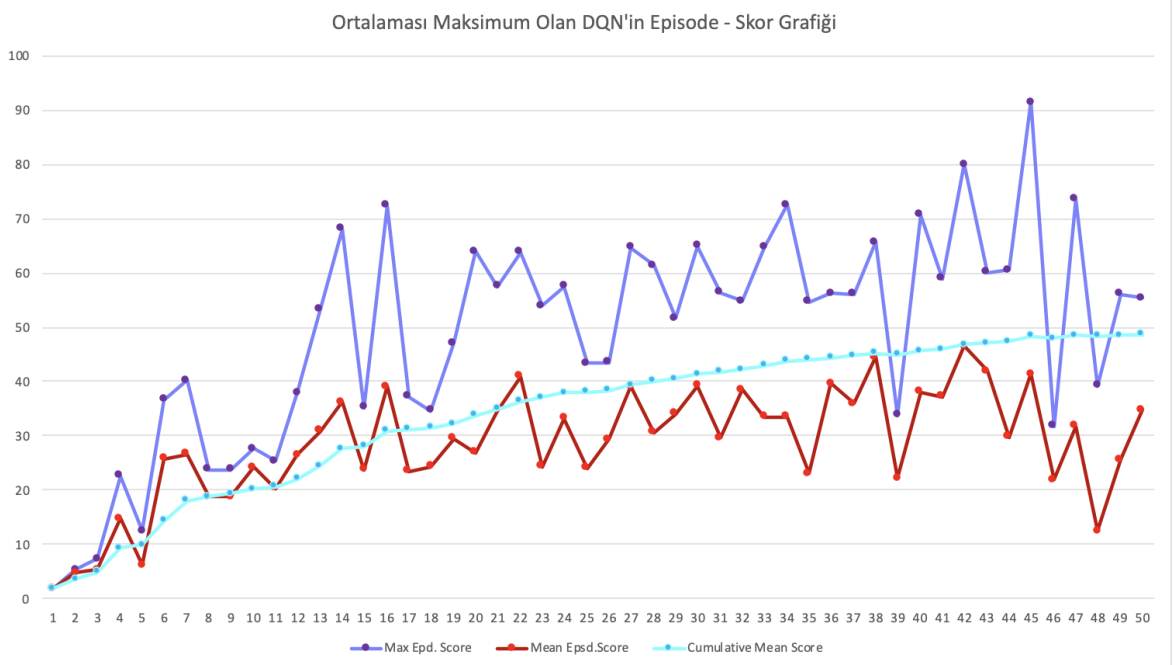
Agar.io oyununda en verimli şekilde skor toplayan algoritmayı değerlendirmek için CNN, Deep Q Network ve SARSA algoritmalarınının her birinden birer agar modeli eğittik. Sonuçlarımızda;

DQN'de,

Ortalamaların ortalamasında en yüksek değer Step 1750'de ortaya çıkmıştır. Maximumlar arasındaki en yüksek değer ise Episode 250'de ortaya çıkmıştır.

DQN

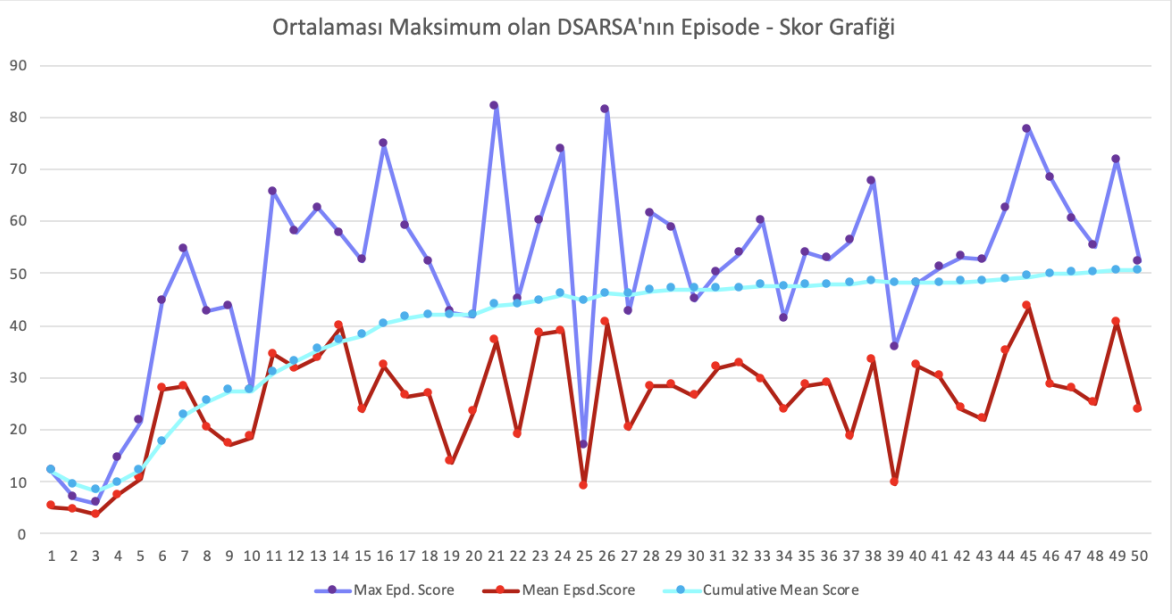
Değer	Episode 250	Episode 100	Episode 75	Episode 50	Episode 40	Episode 30	Episode 20
Ort	24.72	19.03	17.67	18.65	19.35	16.93	12.14
Max	198.32	64.89	54.01	66.53	61.14	54.55	38.21
Değer	Step 250	Step 500	Step 750	Step 1000	Step 1250	Step 1500	Step 1750
Ort	4.96	9.61	14.28	18.55	21.94	24.42	28.38
Max	21.51	35.97	51.13	62.06	70.09	67.96	91.34
Değer	BatchSize 29	BatchSize 30	BatchSize 31	BatchSize 32	BatchSize 33	BatchSize 34	BatchSize 35
Ort	19.44	18.24	18.49	16.79	16.33	19.24	19.44
Max	55.92	59.87	57.76	57.22	54.48	58.19	66.17
Değer	Gamma 0.5	Gamma 0.6	Gamma 0.7	Gamma 0.8	Gamma 0.9	Gamma 1.0	Gamma 1.1
Ort	18.45	17.39	18.91	16.79	18.16	17.10	20.77
Max	64.3	61.34	57.34	57.22	57.48	62.27	64.25
Değer	LR 0.0001	LR 0.001	LR 0.002	LR 0.003	LR 0.004	LR 0.005	LR 0.01
Ort	22.37	17.37	18.00	17.26	18.02	16.97	16.22
Max	60.18	58.94	64.14	55.22	59.61	57.05	52.9
Değer	Epsilon 0.25	Epsilon 0.5	Epsilon 0.75	Epsilon 1	Epsilon 1.25	Epsilon 1.5	Epsilon 1.75
Ort	16.24	14.34	15.78	17.37	14.31	15.46	15.44
Max	57.19	45.91	51.08	58.99	46.78	49.88	51.06



SARSA'da,
Ortalamaların ortalamasında en yüksek değer Step 17500'de ortaya çıkmıştır. Maximumlar arasındaki en yüksek değer ise Episode 100'de ortaya çıkmıştır.

DSARSA

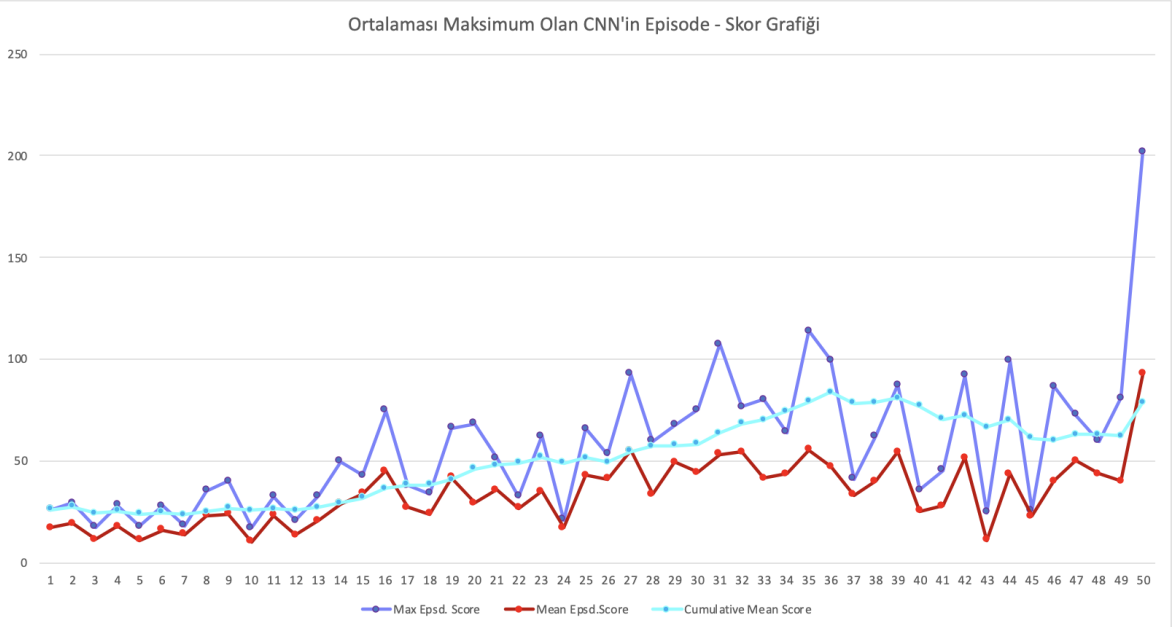
Değer	Episode 250	Episode 100	Episode 75	Episode 50	Episode 40	Episode 30	Episode 20
Ort	23,49	23,37	18,44	20,87	17,48	19,19	13,99
Max	76,04	79,23	72,25	61,8	55,71	56,54	44,13
Değer	Step 250	Step 500	Step 750	Step 1000	Step 1250	Step 1500	Step 1750
Ort	3,91	11,01	13,58	19,05	22,32	24,47	25,74
Max	17,85	43,55	43,92	63,6	71,44	78,27	82,03
Değer	BatchSize 29	BatchSize 30	BatchSize 31	BatchSize 32	BatchSize 33	BatchSize 34	BatchSize 35
Ort	17,68	18,02	17,16	18,38	16,64	16,43	17,68
Max	55,39	62,07	58,66	58,87	52,37	55,52	73,4
Değer	Gamma 0.5	Gamma 0.6	Gamma 0.7	Gamma 0.8	Gamma 0.9	Gamma 1.0	Gamma 1.1
Ort	18,19	19,51	17,19	18,38	19,29	20,08	19,26
Max	63,24	62,8	56,28	58,87	67,32	62,16	76,08
Değer	LR 0.0001	LR 0.001	LR 0.002	LR 0.003	LR 0.004	LR 0.005	LR 0.01
Ort	19,36	19,96	20,24	18,27	17,25	18,07	15,94
Max	57,56	65,15	71,84	60,74	55,54	54,71	44,63
Değer	Epsilon 0.25	Epsilon 0.5	Epsilon 0.75	Epsilon 1	Epsilon 1.25	Epsilon 1.5	Epsilon 1.75
Ort	20,84	18,05	20,05	19,95	22,17	22,39	21,9
Max	69,95	65,64	65,35	65,15	78,73	63,14	68,78



CNN'de,
Ortalamaların ortalamasında en yüksek değer Step 1500'de ortaya çıkmıştır. Maximumlar arasındaki en yüksek değer ise Episode 250'de ortaya çıkmıştır.

CNN

Değer	Episode 250	Episode 100	Episode 75	Episode 50	Episode 40	Episode 30	Episode 20
Ort	29,52	29,01	17,97	22,91	16,61	16,18	11,06
Max	222,89	206,08	67,73	83,71	77,36	63,67	31,24
Değer	**Step 250**	**Step 500**	**Step 750**	**Step 1000**	**Step 1250**	**Step 1500**	**Step 1750**
Ort	6,31	7,22	12,24	22,91	26,23	34,26	23,41
Max	21,52	23,84	49,36	83,71	87,65	201,77	198,48
Değer	**BatchSize 29**	**BatchSize 30**	**BatchSize 31**	**BatchSize 32**	**BatchSize 33**	**BatchSize 34**	**BatchSize 35**
Ort	18,89	13,59	21,94	18,07	14,34	15,52	15,20
Max	62,34	62,82	89,63	71,15	54,2	57,84	56,12
Değer	**Gamma 0.5**	**Gamma 0.6**	**Gamma 0.7**	**Gamma 0.8**	**Gamma 0.9**	**Gamma 1.0**	**Gamma 1.1**
Ort	17,48	18,71	19,62	19,36	17,31	20,19	18,67
Max	59,8	59,4	69,72	69,03	54,1	73,28	64,95
Değer	**LR 0.0001**	**LR 0.001**	**LR 0.002**	**LR 0.003**	**LR 0.004**	**LR 0.005**	**LR 0.01**
Ort	23,54	18,07	18,27	16,86	17,49	16,09	18,28
Max	82,62	71,15	61,88	72,29	55,65	62,27	54,48
Değer	**TAU 1**	**TAU 2**	**TAU 3**	**TAU 4**	**TAU 5**	**TAU 6**	**TAU 7**
Ort	13,96	14,43	19,34	18,92	20,49	16,41	13,08
Max	54,97	51,26	59,62	71,7	70,82	81,66	44,3



CNN maximumlarda ve ortalamada Deep Q Network ve Deep Sarsa'ya oranla çok üstün başarı elde etmesine rağmen süre açısından her sahnedeki her pixeli binary olarak tutmasından dolayı çok verimli değildir. Deep Q Network ile Deep SARSA kendi arasında

kıyaslandığında benzer sonuçlar elde etmelerine rağmen Deep Q Network, Deep SARSA'ya göre daha iyi bir sonuca sahiptir.

7. Kaynakça

WEB_1(2021)

WEB_2(2018)

WEB_3(2015)

WEB_4(2020)

WEB_5(2020)

WEB_6(2017)

WEB_7(2020)

WEB_8

Artificial Intelligence For Games (2nd Edition), Ian Millington & John Funge