KSÜ Mühendislik Bilimleri Dergisi, 28(4), 2025
Araştırma Makalesi

KSU J Eng Sci, 28(4), 2025
Research Article

Kahramanmaras Sutcu Imam University
Journal of Engineering Sciences

Geliş Tarihi :
Kabul Tarihi :

Received Date :
Accepted Date :

DERİN PEKİŞTİRMELİ ÖĞRENME İLE SATRANÇ: BİTBOARD VE DQN UYGULAMASI

DERİN PEKİŞTİRMELİ ÖĞRENME İLE SATRANÇ: BİTBOARD VE DQN UYGULAMASI

Zafer AVCI (ORCID: 0000-****-****-****)

[1] Kahramanmaraş Sütçü İmam Üniversitesi, Bilgisayar Mühendisliği Bölümü, Kahramanmaraş, Türkiye

*Sorumlu Yazar / Corresponding Author: Zafer AVCI, zaferavci89@gmail.com

ÖZET

Bu makale, Derin Q-Network (DQN) kullanarak bir satranç ajanı geliştirmektedir. Ajan, satranç tahtasını her biri 8x8 boyutunda olan 16 farklı bitboard ile temsil eder; bu bitboard'lar beyaz ve siyah taşların konumlarını, boş kareleri, rok haklarını, geçerken almayı ve oyuncu sırasını içerir. Aksiyon uzayı, 64 başlangıç karesinden 64 hedef kareye olası tüm 4096 hamleyi kapsar. Geçersiz hamleler, sinir ağının çıktısına uygulanan bir maskeleme katmanı ile filtrelenir.

DQN mimarisi, her bir aksiyon için Q-değerlerini tahmin etmek üzere üç evrişimli katman ve ardından iki tam bağlantılı katman kullanır. Öğrenme, öncelikli deneyim tekrarı (prioritized experience replay) ile gerçekleştirilir. Ödül fonksiyonu, Stockfish motorunun tahta değerlendirme skorlarındaki değişimlere dayanır; her hamle için küçük bir ceza ve oyun sonu durumları (kazanma/kaybetme/beraberlik) için özel ödüller içerir. Keşif için epsilon-greedy stratejisi kullanılır ve bu strateji keşif sırasında Stockfish'in en iyi hamle önerilerinden yararlanabilir. Uygulama, oyun mekanikleri için python-chess ve DRL modeli için PyTorch kütüphanelerini kullanır. Ajan, rastgele hamle yapan bir rakibe karşı eğitilir ve tam oyunlardaki performansı ile basit mat bulmacalarını çözme yeteneği üzerinden değerlendirilir.

Anahtar Kelimeler: derin pekiştirmeli öğrenme, derin q-network, bitboard, hareket maskeleme

ABSTRACT

This journal develops a chess-playing agent using a Deep Q-Network (DQN). The agent represents the chessboard using 16 distinct 8x8 bitboards, which include the positions of white and black pieces, empty squares, castling rights, en passant, and player turn. The action space encompasses all 4096 possible moves from any of the 64 starting squares to any of the 64 target squares. Invalid moves are filtered by a masking layer applied to the neural network's output.

The DQN architecture employs three convolutional layers followed by two fully-connected layers to estimate Q-values for each action. Learning is facilitated through prioritized experience replay. The reward function is based on changes in Stockfish's board evaluation scores, incorporating a small penalty per move and specific rewards for game outcomes (win/loss/draw). An epsilon-greedy strategy is used for exploration, which can leverage Stockfish's best move suggestions during the exploration phase. The implementation utilizes the python-chess library for game mechanics and PyTorch for the DRL model. The agent is trained against a random-move opponent and evaluated on its performance in full games and its ability to solve simple mate puzzles.

Keywords: Deep reinforcement learning, deep q-network, bitboard, action masking

ToCite: AVCI, Z., (2025). MAKALE BAŞLIĞI. *Kahramanmaraş Sütçü İmam Üniversitesi Mühendislik Bilimleri Dergisi*, 28(3), 1-4.

GİRİŞ

Satranç, karmaşık stratejiler ve derinlemesine hesaplama gerektiren yapısıyla yapay zeka araştırmaları için önemli bir alan olmuştur. Son yıllarda, Derin Pekiştirmeli Öğrenme (DPÖ) teknikleri, satranç gibi oyunlarda insanüstü performans gösterebilen ajanların geliştirilmesine olanak tanımıştır. Bu tez çalışması, "main.py" adlı Python dosyasında sunulan yaklaşımlar doğrultusunda, satranç oynamak üzere bir yapay zeka ajanının geliştirilmesini ve analizini konu almaktadır. Bu giriş bölümü, tezin ilerleyen kısımlarında detaylandırılacak olan materyal ve yöntemleri, elde edilen bulguları, bu bulguların tartışılmasını ve varılacak sonuçları ana hatlarıyla özetleyecektir.

Bu tez kapsamında ele alınacak **Materyal ve Yöntem** bölümünde, öncelikle Derin Q-Network (DQN) tabanlı DPÖ yaklaşımı ayrıntılı olarak incelenecektir. Satranç tahtası durumlarının temsili için, beyaz ve siyah taşların konumları, boş kareler, rok hakları, geçerken alma durumu ve oyuncu sırası gibi bilgileri içeren 16 kanallı 8x8 bitboard yapısı kullanılmıştır. Ajanın aksiyon uzayı, olası tüm 4096 hamleyi (64x64) kapsayacak şekilde tasarlanmış ve yasal olmayan hamleler, sinir ağı çıktısında bir maskeleme katmanı aracılığıyla filtrelenmiştir. Kullanılan DQN mimarisi, üç evrişimli sinir ağı (CNN) katmanı ve ardından gelen iki tam bağlantılı katmandan oluşmaktadır. Ajanın öğrenme süreci, öncelikli deneyim tekrarı (prioritized experience replay) yöntemiyle optimize edilmiş, ödül fonksiyonu ise Stockfish satranç motorunun değerlendirme skorlarındaki değişimler temel alınarak ve her hamle için küçük bir ceza eklenerek tasarlanmıştır. Keşif ve sömürü dengesi için epsilon-greedy stratejisi benimsenmiş olup, tüm bu sistem python-chess ve PyTorch kütüphaneleri kullanılarak hayata geçirilmiştir.

MATERYAL VE YÖNTEM

Bu bölümde geliştirilen Derin Pekiştirmeli Öğrenme tabanlı satranç ajanının oluşturulmasında kullanılan materyaller ve izlenen yöntemler ayrıntılı olarak açıklanmaktadır.

Genel Yaklaşım

Çalışmada, satranç oynamak üzere Derin Q-Network (DQN) algoritması temel alınarak bir DPÖ ajanı geliştirilmiştir. Satranç oyunu ortamı ve kuralları için python-chess kütüphanesi, sinir ağı modellemesi ve eğitimi için ise PyTorch kütüphanesi kullanılmıştır.

Satranç Ortamı ve Araçlar

- python-chess Kütüphanesi: Satranç tahtası temsili, hamle üretimi, yasal hamle kontrolü ve oyun kurallarının yönetimi için kullanılmıştır.
- PyTorch: Dinamik sinir ağı oluşturma, otomatik türev alma (autograd) ve GPU hızlandırma yetenekleri sayesinde DQN modelinin implementasyonu ve eğitimi için tercih edilmiştir.
- NumPy: Sayısal işlemler, özellikle bitboard'ların ve maskelerin matris formatında işlenmesi için kullanılmıştır.
- Stockfish Satranç Motoru: Ajanın öğrenme sürecinde ödül fonksiyonunun hesaplanması ve keşif stratejisinde en iyi hamle ipuçlarının sağlanması amacıyla harici bir satranç motoru olarak kullanılmıştır (belirli bir sürüm ve derinlikte, örn. derinlik 5).
- Diğer Kütüphaneler: random (rastgele seçimler için), os (dosya işlemleri için), time (zaman takibi için), pandas (sonuçların analizi için) ve matplotlib (sonuçların görselleştirilmesi için) kütüphanelerinden yararlanılmıştır.

Durum Temsili (State Representation)

Ajanın çevreyi algılaması için satranç tahtası durumu, 16 kanallı 8x8 boyutunda bir bitboard matrisi ile temsil edilmiştir (convert_state metodu). Bu 16 kanal şunları içerir:

- Beyaz taşlar için 6 bitboard (Piyon, At, Fil, Kale, Vezir, Şah).
- Siyah taşlar için 6 bitboard (p, n, b, r, q, k).
- Boş kareleri gösteren 1 bitboard (board.occupied ^ 2 ** 64 - 1 formülü ile elde edilir).
- Oyuncu sırasını belirten 1 bitboard (sıra beyazda ise tüm bitler 1, siyahta ise 0).
- Rok (castling) haklarını gösteren 1 bitboard (board.castling_rights kullanılarak).
- Geçerken alma (en passant) karesini gösteren 1 bitboard (eğer varsa, ilgili kare biti 1 yapılır). Bu temsil, evrişimli sinir ağları (CNN) için uygun bir girdi formatı sunar.

Aksiyon Uzayı ve Seçimi (Action Space and Selection)

- Aksiyon Uzayı: Ajanın yapabileceği tüm olası hamleler, 64 başlangıç karesinden 64 hedef kareye doğru yapılan tüm potansiyel hareketleri içeren 4096 boyutlu bir aksiyon uzayı ile tanımlanmıştır. Her bir aksiyona index = 64 * (move.from_square) + (move.to_square) formülü ile benzersiz bir indeks atanmıştır.
- Hareket Maskeleme: DQN modelinin çıktısı olan 4096 Q-değerine bir maskeleme katmanı (MaskLayer) uygulanmıştır. Bu maske (mask_and_valid_moves metodu ile oluşturulur), mevcut durumda yasal olmayan hamlelere karşılık gelen Q-değerlerini etkisiz hale getirerek (genellikle sıfırlayarak) ajanın sadece geçerli hamleler arasından seçim yapmasını sağlar.

Sinir Ağı Mimarisi (DQN Architecture)

Kullanılan DQN modeli (DQN sınıfı) aşağıdaki katmanlardan oluşmaktadır:

- Girdi: 8x8x16 boyutunda bitboard temsili.
- Evrişimli Katmanlar:

    1. conv1: 16 giriş kanalından 32 çıkış kanalına, 3x3 kernel, padding=1, ardından BatchNorm2d ve ReLU aktivasyonu.
    2. conv2: 32 giriş kanalından 64 çıkış kanalına, 3x3 kernel, padding=1, ardından BatchNorm2d ve ReLU aktivasyonu.
    3. conv3: 64 giriş kanalından 128 çıkış kanalına, 3x3 kernel, padding=1, ardından BatchNorm2d ve ReLU aktivasyonu.

- Düzleştirme Katmanı (Flatten): Evrişimli katmanların 8x8x128 boyutundaki çıktısını 8192 boyutlu tek bir vektöre dönüştürür.
- Tam Bağlantılı Katmanlar:
    1. fc1: 8192 girişten 8192 çıkışa, ardından ReLU aktivasyonu.
    2. fc2 (Çıkış Katmanı): 8192 girişten 4096 çıkışa (her bir aksiyon için Q-değeri), negatif Q-değerlerine izin vermek için aktivasyon fonksiyonu kullanılmamıştır.
- Maskeleme Katmanı (MaskLayer): fc2 katmanının çıkışına, yasal hamle maskesini uygular.

Öğrenme Algoritması ve Parametreleri

- DQN Algoritması: Q-öğrenme algoritmasının sinir ağları ile birleştirilmiş halidir.
- Deneyim Tekrarı ve Önceliklendirme (remember, learn_experience_replay metotları):
    1. Ajanın deneyimleri (durum, aksiyon, ödül, sonraki_durum, bitti_mi, geçerli_hamleler, sonraki_geçerli_hamleler) bir hafızada (MEMORY_SIZE = 512) saklanır.
    2. Öncelikli deneyim tekrarı kullanılarak, daha yüksek TD-hatasına (zamansal fark hatası) sahip deneyimlerin eğitim için seçilme olasılığı artırılır. Yeni eklenen oyun deneyimlerine başlangıçta yüksek bir öncelik (MAX_PRIORITY = 1e+06) atanır ve bu öncelik, öğrenme adımı sonrasında hesaplanan hataya göre güncellenir.
- Hedef Q-Değeri:
    1. $Q\_hedef = \text{Ödül} + \gamma \times max_a^1(sonraki\_durum,\ a^1)$ (1)
    formülü ile hesaplanır (eğer durum son değilse). İndirgeme faktörü γ=0.5 olarak belirlenmiştir.

- Kayıp Fonksiyonu: Tahmin edilen Q-değeri ile hedef Q-değeri arasındaki Ortalama Karesel Hata (MSELoss) kullanılır.
- Optimizasyon: Adam optimizasyon algoritması, $1x10^{-3}$ öğrenme oranı (learning_rate) ile kullanılmıştır.
- Mini-batch Boyutu: Her bir eğitim adımında batch_size = 16 deneyim kullanılır.

Ödül Fonksiyonu

Ödül sistemi, ajanın stratejik gelişimini yönlendirmek üzere tasarlanmıştır:

- Oyun devam ederken:

$$\text{ödül} = \left(\frac{\text{sonraki\_durum\_Stockfish\_skoru}}{100}\right) - \left(\frac{\text{önceki}_{\text{durum}_{\text{Stockfish}_{\text{skoru}}}}}{100}\right) - 0.01 \quad (2)$$

  Stockfish skorları (derinlik 5) centipawn cinsinden alınıp piyon birimine çevrilir ve her hamle için -0.01'lik bir ceza uygulanarak ajanın daha hızlı kazanması hedeflenir.
- Oyun sonu durumları için özel ödüller:
- Ajan (Beyaz) kazanırsa: +1000.
- Ajan (Beyaz) kaybederse: -1000.
- Beraberlik veya izin verilen maksimum hamle sayısına (max_game_moves) ulaşılırsa: -10.

Keşif Stratejisi

- Epsilon-Greedy Stratejisi: Keşif (exploration) ve sömürü (exploitation) dengesini sağlamak için kullanılır.
  - epsilon değeri başlangıçta 1.0 olarak ayarlanır, her adımda epsilon_decay = 0.99 ile çarpılarak azaltılır ve minimum epsilon_min = 0.01 değerine kadar düşer.
  - ϵ olasılıkla keşif yapılır: Bu durumda %10 olasılıkla Stockfish tarafından önerilen en iyi hamle, %90 olasılıkla ise yasal hamleler arasından rastgele bir hamle seçilir.
  - $1-\epsilon$ olasılıkla sömürü yapılır: Ajan, DQN modelinden elde ettiği Q-değerlerine göre en yüksek değerli yasal hamleyi seçer. Eğer seçilen hamle geçersizse veya tüm yasal hamlelerin Q-değeri 0 veya negatifse, rastgele bir yasal hamle yapılır.

Antrenman Prosedürü

Antrenman süreci, ajanın (ChessAgent nesnesi) belirli sayıda oyun (games_to_play) boyunca, siyah taşlarla rastgele hamleler yapan bir rakibe karşı oynamasıyla gerçekleştirilir. Her bir beyaz hamlesi ve ardından gelen siyah hamlesinden sonra, elde edilen deneyim hafızaya eklenir ve learn_experience_replay metodu ile sinir ağı güncellenir. Epsilon değeri de her adımda adaptiveEGreedy metodu ile ayarlanır. Antrenman öncesinde hafıza, generate_random_sample fonksiyonu ile rastgele oluşturulmuş örneklerle doldurulabilir.

Değerlendirme Metrikleri

Ajanın performansı aşağıdaki metriklerle değerlendirilir:

- Antrenman süresince ortalama kayıp (loss) değerinin değişimi.
- Antrenman süresince oyun sonu elde edilen skorların (ve hareketli ortalamalarının) değişimi.
- Belirli sayıda test oyunu sonucunda, rastgele hamle yapan bir rakibe karşı elde edilen kazanma, kaybetme ve beraberlik yüzdeleri (test fonksiyonu).
- Basit mat-1 ve mat-2 bulmacalarını çözme başarısı.

EĞİTİM PROSEDÜRÜ VE METOTLARI

Bu bölümde, satranç ajanının eğitilmesi için chess_agent_training.py dosyasında implemente edilen ana fonksiyonlar ve izlenen prosedür detaylandırılmaktadır. Eğitim süreci, train() fonksiyonu tarafından yönetilmekte ve asıl öğrenme mantığı Q_learning() fonksiyonu içerisinde gerçekleştirilmektedir.

Eğitim Sürecini Başlatma

Train fonksiyonu, tüm eğitim sürecini başlatan ve yöneten ana fonksiyondur. Bu fonksiyonun temel görevleri şunlardır:

1. Ajanın Oluşturulması: İlk olarak, ChessAgent sınıfından bir nesne (ajan) oluşturulur. Bu nesne, öğrenme için gerekli olan sinir ağı modelini (DQN) ve ilgili tüm metotları (eylem seçimi, hafızaya kaydetme, öğrenme vb.) içerisinde barındırır.
2. Parametrelerin Belirlenmesi: Eğitim için gerekli olan temel parametreler bu fonksiyonda tanımlanır. Bunlar arasında Stockfish satranç motorunun sistemdeki dosya yolu, toplam oynanacak oyun sayısı (games_to_play=50) ve bir oyundaki maksimum hamle sayısı (max_game_moves=75) bulunur.
3. Q-Learning'in Çağrılması: Belirlenen parametreler ile Q_learning() fonksiyonu çağrılarak ajanın aktif öğrenme süreci başlatılır.
4. Modelin Kaydedilmesi: Eğitim tamamlandıktan sonra, ajanın öğrendiği sinir ağı ağırlıkları (policy_net'in state_dict'i), ileride tekrar kullanılabilmesi veya test edilebilmesi için .pth uzantılı bir dosyaya kaydedilir (agent.save_model).

Modelin Eğitilmesi

Bu fonksiyon, ajanın Derin Q-Öğrenme (DQN) algoritması ile eğitildiği temel döngüyü içerir. İzlenen adımlar, bir Pekiştirmeli Öğrenme (PÖ) sürecinin standart bileşenlerini yansıtmaktadır.

Başlatma ve Hazırlık

- Stockfish Motorunun Başlatılması: Fonksiyon, belirtilen yolda Stockfish motorunu bir alt işlem (subprocess) olarak başlatır. Eğer motor bulunamazsa, kullanıcıya bir hata mesajı gösterilir. Stockfish'in çalışması sırasında bir sorun yaşanması ihtimaline karşı hata yakalama (try-except) blokları ile motorun yeniden başlatılması sağlanmıştır.
- Veri Kayıt Değişkenleri: Eğitim süresince her adımdaki kayıp (loss) değerlerini ve her oyun sonundaki skoru kaydetmek için loss ve final_score adında listeler oluşturulur.

Oyun Döngüsü

Ana döngü, games_to_play parametresi ile belirlenen sayıda oyun oynanana kadar devam eder. Her bir oyunun başlangıcında:

- Satranç tahtası (chess.Board) sıfırlanır. Opsiyonel olarak, board_config parametresi ile belirli bir başlangıç pozisyonundan (örneğin, bir bulmaca durumu) oyun başlatılabilir.
- Oyunun bitip bitmediğini kontrol eden done bayrağı ve hamle sayacı sıfırlanır.
- Başlangıç pozisyonu Stockfish tarafından analiz edilir ve en iyi hamle (best_move), ajanın keşif (exploration) stratejisinde kullanılmak üzere saklanır.

Hamle Döngüsü (PÖ Adımları)

Her bir oyun içindeki bu döngü, oyun bitene veya max_game_moves sınırına ulaşılana kadar devam eder. Her bir adımda aşağıdaki PÖ süreci işler:

- Eylem Seçimi: Ajan, mevcut tahta durumuna göre select_action metodunu kullanarak bir eylem (satranç hamlesi) seçer. Bu seçim, epsilon-greedy stratejisine göre ya keşif (rastgele veya Stockfish ipucuna dayalı) ya da sömürü (DQN modelinin tahminine dayalı) odaklı olur.
- Ödül Hesaplama:
    - Ajan hamle yapmadan hemen önce, mevcut durumun Stockfish skoru (board_score_before) kaydedilir.
    - Ajan (beyaz) hamlesini yapar.
    - Eğer oyun biterse (mat, pat, beraberlik veya hamle sınırı), terminal bir ödül atanır: kazanma için +1000, kaybetme için -1000, beraberlik/sınır için -10. Bu deneyim, sonraki_durum (next_state) None olacak şekilde hafızaya (agent.remember) eklenir.
    - Eğer oyun bitmezse, rakip (siyah) yasal hamleler arasından rastgele bir hamle yapar. Rakibin bu hamlesinden sonra oluşan yeni durumun Stockfish skoru (board_score_after) alınır. Anlık ödül, bu iki skor arasındaki farktan ve her hamle için uygulanan küçük bir cezadan oluşur: ödül = board_score_after - board_score_before - 0.01. Bu tam deneyim (mevcut durum, eylem, ödül, sonraki durum) hafızaya eklenir.
- Öğrenme: Her hamle döngüsünün sonunda agent.learn_experience_replay() metodu çağrılır. Bu metot, hafızadan bir grup (mini-batch) deneyim seçer ve bu deneyimleri kullanarak sinir ağının ağırlıklarını günceller.
- Epsilon Güncellemesi: agent.adaptiveEGreedy() metodu ile keşif oranı (epsilon) bir miktar azaltılarak ajanın zamanla daha fazla sömürü yapması sağlanır.

Sonuçların Görselleştirilmesi

Tüm oyunlar tamamlandıktan sonra, pandas ve matplotlib kütüphaneleri kullanılarak iki adet grafik çizdirilir:

- Oyun Sonu Skoru Grafiği: Her bir oyunun sonunda elde edilen skoru ve bu skorların hareketli ortalamasını (ma) gösterir. Bu grafik, ajanın genel performans trendini anlamak için kullanılır.
- Adım Başına Kayıp (Loss) Grafiği: Her bir eğitim adımında hesaplanan kayıp değerini ve hareketli ortalamasını gösterir. Bu grafiğin zamanla azalması, sinir ağının başarılı bir şekilde öğrendiğine işaret eder.

Yapay Zeka Katkı Beyanı

Bu makalenin hazırlanması sürecinde yapay zeka araçlarından yararlanılmıştır. Özellikle, "main.py " adlı Python dosyasında sunulan teknik çalışmanın akademik bir metne dönüştürülmesi amacıyla, Google tarafından geliştirilen Gemini dil modeli kullanılmıştır. Yapay zeka aracının katkıları; tez başlığı önerileri sunma, notebook içeriğine dayalı bir özet ve anahtar kelimeler oluşturma, giriş ve materyal-yöntem gibi bölümlerin ilk taslaklarını yazma ve oluşturulan bazı metinlerin İngilizceye çevrilmesi aşamalarını kapsamaktadır.

Makalede sunulan temel algoritmaların geliştirilmesi, kodların yazılması, deneylerin yapılması ve bulguların analiz edilmesi gibi tüm özgün bilimsel çalışma tamamen yazarlara aittir. Yapay zeka aracı, bu çalışmanın metinsel olarak ifade edilmesi ve yapılandırılmasında bir asistan olarak görev yapmıştır. Makalenin nihai içeriği yazarlar tarafından düzenlenmiş, doğrulanmış ve onaylanmış olup tüm sorumluluk yazarlara aittir.

Artificial Intelligence Contribution Statement

Artificial intelligence (AI) tools were utilized in the preparation of this manuscript. Specifically, the Gemini language model, developed by Google, was used to assist in transforming the technical work presented in the Python file "main.py" into an academic text. The contributions of the AI tool include: suggesting thesis titles, generating a summary and keywords based on the notebook's content, writing the initial drafts of sections such as the introduction and materials and methods, and translating some of the generated text into English.

KAYNAKLAR

1. Fiekas, N. (2024). *python-chess* (Sürüm 1.8.0) [Yazılım]. Alınan yer: https://github.com/niklasf/python-chess
2. Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., ... & Oliphant, T. E. (2020). Array programming with NumPy. *Nature, 585*(7825), 357-362.
3. Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.
4. Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., ... & Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature, 518*(7540), 529-533.
5. Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., ... & Chintala, S. (2019). PyTorch: An imperative style, high-performance deep learning library. In *Advances in neural information processing systems* (s. 8026-8037).
6. Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., ... & Hassabis, D. (2017). Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*.
7. Stockfish Team, The. (2024). *Stockfish* (Sürüm 16.1) [Yazılım]. Alınan yer: https://stockfishchess.org
8. Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.