TAŞINABİLİR YÜRÜTÜLEBİLİR DOSYALARDA YİNELENEN SİNİR AĞLARINI KULLANARAK STATİK KÖTÜ AMAÇLI YAZILIM ALGILAMA

YÜKSEK LİSANS TEZİ

MUSA GÜL
ORCID ID: 0000-0002-1802-2537

MERSİN ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

BİLGİSAYAR MÜHENDİSLİĞİ
ANABİLİM DALI

MERSİN
MAYIS- 2021

TAŞINABİLİR YÜRÜTÜLEBİLİR DOSYALARDA YİNELENEN SİNİR AĞLARINI KULLANARAK STATİK KÖTÜ AMAÇLI YAZILIM ALGILAMA

YÜKSEK LİSANS TEZİ

MUSA GÜL
ORCID ID: 0000-0002-1802-2537

MERSİN ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

BİLGİSAYAR MÜHENDİSLİĞİ
ANABİLİM DALI

Danışman
Doç. Dr. Erdinç AVAROĞLU
ORCID ID: 0000-0003-1976-2526

MERSİN
MAYIS - 2021

ÖZET

Taşınabilir Yürütülebilir Dosyalarda Yinelenen Sinir Ağlarını Kullanarak Statik Kötü Amaçlı Yazılım Algılama

Teknolojideki son gelişmeler ile kötü amaçlı yazılımdan koruma yazılımının ortaya çıkmasından bu yana, bu yazılım ya da yazılımları atlatmaya yönelik özel olarak tasarlanmış karmaşık kötü amaçlı yazılımlarda bir artış görülmüştür. Bu da daha gelişmiş algılama tekniklerine yönelik araştırmalara öncülük etmiştir. Bu çalışmanın temel amacı, taşınabilir yürütülebilir dosyaları statik olarak kötü niyetli veya zararsız olarak sınıflandırmak için derin bir sinir ağı tasarlamak ve değerlendirmektir.

Bu amaçla, bilinen kötü niyetli ve zararsız dosyaların taşınabilir yürütülebilir dosyalarından çıkarılan verileri içeren Microsoft'un sunduğu Microsoft Malware Classification Challenge (BIG 2015) ekinliği için hazırlanan veri seti kullanılmıştır. Python programlama dili kullanılarak taşınabilir yürütülebilir dosya örnekleri özellik çıkarımına uygun hale gelecek şekilde parçalara ayrılmıştır. Tüm dosyalar sadece işlem kodları kalacak şekilde ayıklanmıştır. Kod sekansları içerisinden tekrar eden ve gereksiz olan işlem kodları silinmiş, her bir dosyadan gelen kod sekansının büyüklüğü belirli bir boyut ile sınırlandırılmıştır. Bu boyuttan büyük olan dosyalar için geri kalan kodlar alınmazken, küçük olanlar içinse eksik kalan kısımlar sıfır kullanılarak doldurulmuştur. Oluşturulan sözlük, popüler gözetimsiz ve tahmin temelli doğal dil işleme modellerinden Word2Vec kullanılarak vektörel hale getirilmiştir. Word2Vec kullanılırken çalışmaya uygunluğu göz önünde bulundurularak Sürekli Kelime Torbası (CBOW) mimarisi kullanılmıştır. CBOW modeli uygun görüldükten sonra en iyi sonuçların pencere boyutunun 15 olduğu çalışmada elde edildiği görülmüştür, bu nedenle pencere boyutu 15 olarak belirlenmiştir. Çalışma uzun sekanslar içerdiğinden RNN modelinde hız düşüşü öngörülerek RNN'nin farklı bir versiyonu olan LSTM kullanılmıştır.

LSTM modelinin oldukça az sayıda eğitim turu yapılsa dahi öğrenmeye gayet yüksek doğruluk oranları ile başladığını, ardından eğitim turu arttıkça da doğruluk oranının dramatik bir şekilde yükseldiği görülmektedir. Beklenildiği gibi 10 eğitim turu sonrasında ise artık model doygunluğa eriştiği için performansındaki gelişmeler çok sınırlı kalmıştır. 50 eğitim turu sonrası elde edilen en iyi doğruluk değeri ise %95,8 olarak elde edilmiştir. Bulgular, yeni üretilen ya da az bilinen kötü amaçlı yazılımların kolaylıkla tespit edilmesi konusunda oldukça önem arz etmekte ve virüs imza veri tabanı temelli koruma yazılımları yerine makine öğreniminin modellendiği daha gelişmiş kötü amaçlı yazılımdan korunma tekniklerinin tasarımında yol gösterici olacaktır.

Anahtar Kelimeler: LSTM, Kötü amaçlı yazılım, RNN, Word2Vec, Virüs.

Danışman: Doç. Dr. Erdinç AVAROĞLU, Mersin Üniversitesi, Bilgisayar Mühendisliği Anabilim Dalı, Mersin.

ABSTRACT

Static Malware Detection Using Recurrent Neural Networks in Portable Executables

Since the advent of anti-malware software with recent advances in technology, there has been an increase in sophisticated malware specifically designed to circumvent them. This led to research into more advanced sensing techniques. The main purpose of this study is to design and evaluate a deep neural network to statically classify portable executables as malicious or harmless.

For this purpose, the data set prepared for the Microsoft Malware Classification Challenge (BIG 2015) event presented by Microsoft, which includes the data extracted from the portable executable files of known malicious and harmless files, was used. Examples of portable executables using the Python programming language are segmented to be suitable for feature extraction. All files have been extracted so that only opcodes remain. Repetitive and unnecessary opcodes were deleted from the code sequences, and the size of the code sequence from each file was limited to a certain size. For the files larger than this size, the remaining codes are not taken, while for the smaller ones, the missing parts are filled with zeros. The created dictionary was vectorized using Word2Vec, one of the popular unattended and predictive-based natural language processing models. When using Word2Vec, the Continuous Bag of Words (CBOW) architecture was used considering its suitability to work. After the CBOW model was approved, it was seen that the best results were obtained in the study with a window size of 15, therefore the window size was determined as 15. Since the study includes long sequences, a different version of RNN, LSTM, was used by predicting a speed decrease in the RNN model.

It is seen that the LSTM model starts learning with very high accuracy rates even if a very small number of training tours are performed, and then the accuracy rate increases dramatically as the training tour increases. As expected, after 10 training rounds, the performance improvements were very limited as the model had reached saturation. The best accuracy value obtained after 50 training rounds was obtained as 95.8%. The findings are very important in easily detecting newly produced or lesser-known malware and will guide the design of more advanced anti-malware techniques modeled by machine learning rather than virus signature database-based protection software.

Keywords: LSTM, Malware, RNN, Word2Vec, Virus.

Advisor: Assoc. Dr. Erdinç AVAROĞLU, Mersin University, Department of Computer Engineering, Mersin.

TEŞEKKÜR

Bu çalışmada makine öğrenimi ve yapay sinir ağları konusundaki bilgileri ile bana destek olan Kutluhan KİBRİT ve Özkan KIRIK büyüklerime ve tez hazırlık sürecimde baştan sona bana destek olup emeğini esirgemeyen değerli hocam Doç. Dr. Erdinç AVAROĞLU hocama ayrıca tez sunumuma katılan değerli hocalarım ve jüri üyelerim Prof. Dr. Hamza EROL ve Doç. Dr. Taner TUNCER hocalarıma ve son olarak destekleri ile motivasyonumu yüksek tutan ailem ve eşime sonsuz teşekkürler.

İÇİNDEKİLER

TABLOLAR DİZİNİ

ŞEKİLLER DİZİNİ

KISALTMALAR ve SİMGELER

Kısaltma/Simge	Tanım
URL	Uniform Resource Locator
HTTP	Hyper Text Transfer Protocol
COFF	Common Object File Format
API	Application Programming Interface
PE	Portable Executable
RNN	Recurrent Neural Network
LSTM	Long Short-Term Memory

1. GİRİŞ

Kötü amaçlı yazılım algılama kavramı, esas olarak kötü niyetli amaç oluşturmak için yürütülebilir dosyaların analiz edilmesiyle ilgili bir konu olarak bilinmektedir. Kötü amaçlı yazılımdan koruma yazılımının ortaya çıkmasından bu yana, bu yazılımı atlatmak için özel olarak tasarlanmış karmaşık kötü amaçlı yazılımlarda bir artış görülmüştür. Bu da daha gelişmiş algılama tekniklerine yönelik araştırmalara öncülük etmiştir. Kötü amaçlı yazılım analizi veya kötü amaçlı yazılım tespiti iki şekilde gerçekleştirilebilir: statik veya dinamik olarak.

Statik Kötü Amaçlı Yazılım Algılama: Statik kötü amaçlı yazılım algılama, bir ikili dosyayı yürütmeden analiz etme sürecidir. Bu, dosyanın tamamen yayılmasını ve her bileşenin incelenmesini, tersine mühendislik yapmak için bir sökücü kullanılmasını veya akışını incelemek için montaj koduna dönüştürülmesini içerebilir [1]. Varsa yazılımın orijinal kaynak koduna da genişletilebilir [2]. Bu, genellikle tüm kötü amaçlı yazılımdan koruma yazılımları tarafından kullanılan kötü amaçlı yazılımlara karşı ilk savunma hattıdır.

Statik analiz genellikle bilinmeyen bir dosyayla uğraşırken ilk olarak gerçekleştirilir. İlk adım, ana bilgisayarda [3] yüklü antivirüs programı ile dosyayı manuel olarak taramaktır. Dosya zaten biliniyorsa, kendi başınıza çözmeye çalışmak için saatler harcamanın bir anlamı yoktur. (Öğrenme deneyimi hariç.) Sistem antivirüs programına ek olarak, dosya, 43 farklı antivirüs programı kullanarak dosyayı tarayan VirusTotal gibi bir site üzerinden çalıştırılabilir. Aynı dosyayla başka birinin karşılaşıp karşılaşmadığını görmek için dosyanın karmasını hesaplamak ve çevrimiçi olarak aramak da yararlı olabilir.

Dize analizi, dosya hakkında ipuçları almanın basit bir yoludur. Komut satırı seçenekleri, kullanıcı diyaloğu, şifreler, URL'ler e-posta adresleri, kitaplıklar ve işlev çağrıları gibi dosya bilgilerindeki tüm dizeleri listeleyerek bulunabilir [3].

Demontaj, statik analizin hayati bir parçasıdır. Bir ikiliden derleme talimatlarını alarak, programın ne yaptığını anlamak için kaynak kodu araştırılabilir. Yine de anlamak için, (bizim durumumuzda) x86 ve x86-64 mimarisi ve Windows iç bileşenleri hakkında derin bilgiye sahip olmak gerekir. Kodun anlaşılmasını kolaylaştırmak için, kod çözülebilir. Bu şekilde kod, daha yüksek seviyeli bir dilde temsil edilir. Kod orijinali gibi olmayacak olsa da, kodu inceleyen kişinin işini kolaylaştıracaktır.

Dinamik Kötü Amaçlı Yazılım Algılama: Dinamik kötü amaçlı yazılım algılama, kötü amaçlı bir yazılımı belirlemek için kötü amaçlı yazılım çalışırken davranış analizini kullanır. Genellikle bu, yürütülebilir dosyanın hedef sisteme herhangi bir zarar vermemesini sağlamak için bir SandBox ortamında yapılır. Bu analiz biçimi genellikle yoğun kaynak gerektirir ve çeşitli şekillerde atlatılabilir. Hata ayıklayıcılar kara kutu testi kullanılarak tespit edilemeyen sistem çağrılarını veya diğer davranış kalıplarını analiz etmek için de kullanılabilir [4].

Bu tezin kapsamı için sadece statik kötü amaçlı yazılım tespitine odaklanılmıştır.

Makine öğrenimi, matematiksel işlevler kullanılarak kolayca belirlenemeyen karmaşık özelliklere sahip verileri sınıflandırmak için uzun süredir kullanılmaktadır. Günümüzde derin sinir ağları, veri sınıflandırması, veri tahmini, görüntü tanıma, doğal dil işleme vb. dahil (ancak bunlarla sınırlı olmamak üzere) çeşitli farklı uygulamalar için kullanılmaktadır. Sinir ağlarının bu çok yönlülüğü, büyük miktarda verinin mevcut olduğu büyük veri gibi bir şey için mükemmeldir, ancak belirli bir sonuç elde etmek için işlemek hesaplama açısından pahalıdır.

Yakın zamana kadar, denetimli öğrenim için etiketli veri kümelerinin bulunmaması, kötü amaçlı yazılım tespiti için makine öğrenimi veya derin öğrenmenin kullanımındaki ilerlemeyi yavaşlatmıştı. Igor Santos vd. bilinmeyen kötü amaçlı yazılımları tespit etmekte makine öğrenimini kullanmak için statik-dinamik yaklaşım olarak OPEM'i önermiştir [5]. Yürütülebilir dosyaların demonte edilmesinden elde edilen operasyonel kodları analiz etmeyi ve kötü niyetli niyetleri belirlemek için yürütme izlerini analiz etmeyi önermişlerdir. Benzer şekilde, Android için DroidDolphin adlı dinamik bir kötü amaçlı yazılım algılama çerçevesi, dinamik kötü amaçlı yazılım analizi kullanarak % 86.1 doğruluk elde etmeyi başarmıştır [6]. Her iki yöntem de genellikle hesaplama açısından pahalıdır ve etiketli verilerin sınırlı kullanılabilirliğinden muzdariptir.

Dinamik analizi gerçekleştirmenin en basit yolu, numuneyi çalıştırmak ve ne olduğunu izlemektir. Yalnızca korumalı alan veya çevrimdışı bir laboratuvar gibi izole bir ortamda çalıştırmak önemlidir. Dinamik yaklaşım her zaman statik analiz yapıldıktan sonra uygulanmalıdır. Bir kötü amaçlı yazılım örneği çalıştırılırken, izlenmesi gereken birkaç husus vardır [3]:

• Dosya etkinliği

Kötü amaçlı yazılım, bilgi toplamak, diğer programları başlatmak veya DLL'leri yüklemek için dosyaları okuyabilir. Diğer programları değiştirmek için dosyalar yazılabilir veya değiştirilebilir. Dosya sistemindeki tüm aktiviteyi kaydetmek için iyi bir araç Diskmon'dur [7].

• Süreçler

İşlemleri kaydetmek için Process Explorer [8] kullanılabilir. Bu araçla, işlemin yüklediği tüm dosyalar, kayıt defteri anahtarları ve DLL'ler günlüğe kaydedilir. Ayrıca, süreçler bir ağaç yapısında düzenlenir, bu nedenle sürecin herhangi bir yeni işlem doğurup doğurmadığını görmek kolaydır.

• Ağ etkinliği

Çok sayıda kötü amaçlı yazılım, komut almak ve / veya bilgi göndermek için ağ bağlantısını kullandığından, ağ etkinliği izlenmelidir. TCPView [9], hangi bağlantı noktalarının

gelen trafiği dinlediğini araştırmak için bir araçtır. Ağ üzerinden gönderilen ve alınan tüm bilgileri toplamak için Wireshark [10] kullanılabilir.

• Kayıt erişimi

Windows'taki kayıt defteri, işletim sistemi ve yüklü programların birçoğu için yapılandırma anahtarlarını içeren bir veritabanıdır. Bir kayıt defteri anahtarının değiştirilmesi, sistemin güvenliği üzerinde büyük bir etkiye sahip olabilir. Yine, İşlem Monitörü kayıt değişikliklerini izlemek için kullanılabilir.

Hata ayıklama, dinamik analiz gerçekleştirmenin başka bir yoludur.

1.1. Amaç

Bu tezin temel amacı, taşınabilir yürütülebilir dosyaları statik olarak kötü niyetli veya zararsız olarak sınıflandırmak için derin bir sinir ağı tasarlamak ve değerlendirmektir. Bu amaçla, bilinen kötü niyetli ve zararsız dosyaların taşınabilir yürütülebilir dosyalarından çıkarılan verileri içeren Microsoft'un sunduğu Microsoft Malware Classification Challenge (BIG 2015) ekinliği için hazırlanan veri seti kullanılmıştır. Model oluşturulurken kötü amaçlı yazılımın statik analizini ele almak için literatürde daha önce önerilen benzer modeller incelenmiştir.

1.2. Taslak

- Bölüm 1, bu tezde kapsanan kavramları tanıtır.
- Bölüm 2, statik kötü amaçlı yazılım analizi alanlarında yapılan önceki çalışmalardan ve kötü amaçlı yazılım tespitinde kullanılan makine öğrenimi yaklaşımlarını inceler.
- Bölüm 3, önerilen modeli anlamadan önce üzerinde çalışılması gereken taşınabilir yürütülebilir dosyaların çeşitli yönlerini açıklar. Modelimiz için kullanılan veri setini ve taşınabilir yürütülebilir dosya formatını nasıl ilişkilendirdiğini kısaca kapsar.
- Bölüm 4, modelimizin uygulanmasında yer alan adımları ve süreçleri, nihai modelin tüm yapısıyla birlikte ayrıntılı olarak açıklamaktadır.
- Bölüm 5'te modelimiz üzerinde yapılan deneyleri ve bunların gerçek dünyadaki sonuçlarını tartışıyoruz. Model için kaynak koduna erişim kaynakları ve yapılan tüm deneyler bu bölümde yer almaktadır.
- Bölüm 6, tezin içeriğini, modelini ve bu tezde kapsanmayan araştırma alanlarını özetlemektedir. Ayrıca bu tezdeki olası boşluklardan ve bu boşlukları kapatmak için yapılabilecek araştırmalardan da bahsediyor.

2. KAYNAK ARAŞTIRMALARI

Bu bölümde, kötü amaçlı yazılım tespiti için makine öğrenimini kullanma konusunda yayınlanan çalışmalar incelenmiştir. Bazı uygulamalar bu tezde ele alınanlara benzer, ancak kullanılan veri setinin mevcut olmaması veya sonuç elde etmek için özel çerçevelerin kullanılması nedeniyle tekrarlanamaz. Ayrıca, dosyaların statik ve dinamik analizini kullanarak diğer platformlarda kötü amaçlı yazılım tespiti ile ilgilenen bu alandaki bazı ilgili çalışmalar da incelenmiştir.

Klasik imza tabanlı yöntemlerin üstesinden gelemediği bilinmeyen kötü amaçlı yazılımlarla başa çıkmak için iki farklı yaklaşım geliştirilmiştir: anormallik algılayıcıları ve veri madenciliği tabanlı algılayıcılar [11]. Anormallik algılayıcıları, zararsız yazılıma dayalı bir profil oluşturur ve bir dosya profilden saptığında şüpheli olarak işaretlenir. Veri madenciliği tabanlı, her iki veri kümesindeki özelliklere bakar ve bir dosyayı bu özelliklere göre sınıflandırır.

2005 yılında Li ve arkadaşları [12], dosya türünü tanımlamak için bir dosyanın normalleştirilmiş bayt değeri dağılımının 1 gramlık bir temsilini kullanmayı önerdi. Bunun hem exe, gif, jpg, pdf ve doc dosyalarında K-Means algoritması kullanılırken ortalama %98,9 doğrulukla son derece doğru olduğu kanıtlandı. Yu ve arkadaşları benzer deneyler yaptı [13].
Bilar'dan [14], opcode dağıtımının kötü amaçlı ve zararsız yazılımlarda farklı olduğunu biliyoruz. 67 kötü niyetli ve 20 iyi huylu örnekten oluşan bir veri kümesinde, işlem kodlarının yaklaşık üçte biri aynı sıklığa, üçte bir oranında daha yüksek ve kötü amaçlı yazılımlara karşı iyi huyluya karşı üçte bir oranında daha düşük seviyeye sahipti. Ayrıca, kötü amaçlı yazılım daha yüksek oranda nadir işlem kodları içerir.

Moskovitch ve arkadaşları [15], 30.000'den fazla dosya içeren bir veri kümesi üzerinde bayt dizisi n-gram kullanarak bir deney yaptı. Dengesizlik sorununu hesaba kattılar: bir sınıfın diğerine kıyasla önemli ölçüde daha fazla örneği olduğu. Veri kümesindeki yalnızca %15 kötü amaçlı dosyalar ile %99 doğruluk elde ettiler. Yapay sinir ağlarının, karar ağaçlarının ve saf Bayes'in Weka uygulamaları kullanıldı. N-gramların karmaşıklığını azaltmak için, yalnızca en üstteki 1.000 baytlık kodları seçmek için terim frekansı kullanıldı. Bunu yaparak, n = 6'ya kadar n-gram kullanılabilir. İlginç bir şekilde n = 2, muhtemelen zararsız dosyalara kıyasla kullanılan az sayıda kötü amaçlı yazılım nedeniyle en iyi sonuçları verdi.

Shankarapani ve arkadaşları [16], derleme ve Uygulama Programlama Arayüzü (API) çağrı dizileri aracılığıyla kötü amaçlı yazılım algılamayı karşılaştırarak, işlem kodlarının daha yüksek doğruluğa sahip olduğunu, ancak hesaplama açısından daha pahalı olduğunu keşfettiler. Ayrıca, paketleyicilerin her iki dosya sınıfı tarafından kullanıldığını, şifrelemenin ise yalnızca kötü amaçlı yazılım tarafından kullanıldığını buldular.

Santos ve arkadaşları [11] aşağıdaki yöntemi kullandı: Dosyaları sökmek için NewBasic Assembler kullanıldı. Ardından bir işlem kodu profili oluşturuldu. Bu, kötü niyetli ve zararsız veri kümelerinde farklı işlem kodlarının kaç kez kullanıldığının bir listesiydi. Ayrıca, işlem kodu alaka düzeyi hesaplandı. Bu, değişkenler arasındaki istatistiksel bağımlılığı ölçmek için karşılıklı bilgi kullanılarak yapıldı. İşlem kodları n-gram uzunlukta n = 1 ve n = 2 olarak gruplandırıldı. Kullanılan sınıflandırma algoritmaları karar ağaçları, destek vektör makineleri, k-en yakın komşular ve Bayes ağlarıydı. Normalleştirilmiş polinom çekirdeği ve n-gram uzunluğu n = 2 olan destek vektör makineleri en iyi sonucu verdi (%95,9).

Bulduğumuz en son çalışma Zolothukin ve arkadaşları [17] tarafından yapılmıştır. Kötü amaçlı yazılımları tanımlamak için yinelemeli destek vektör makinelerine dayalı bir kümeleme algoritması kullandılar. N-gram uzunluk n = 1 ve n = 2 kullanılmıştır. Boyut indirgeme yöntemi olarak n = 2 ve ReliefF ile %97 doğruluk elde edilmiştir.

2.1. Statik Kötü Amaçlı Yazılım Tespiti

Statik kötü amaçlı yazılım analiziyle ilgili çeşitli zorluklar vardır. Bu sorunların çoğu, çalışma zamanı sırasında dosya bozulması, kod gizleme veya şifrelenmiş ikili çalıştırılabilir dosyalar gibi dinamik kötü amaçlı yazılım analizi kullanılarak çözülebilir. Aşağıda, bu problemlerden bazılarını ve anlambilimsel analizcilerin bunları çözmedeki eksiklikleri ortaya çıkarılmıştır.

Symantec'in son tehdit raporuna göre, 2014 yılında önceki yıllara göre çok daha fazla kötü amaçlı yazılım tespit edildi [18]. Geçen yıl 317 milyondan fazla yeni kötü amaçlı yazılım parçası oluşturuldu, yani her gün yaklaşık bir milyon yeni tehdit ortaya çıktı.

2.1.1. Farklı Kötü Amaçlı Yazılım Türleri

Kötü amaçlı yazılım için çeşitli tanımlar mevcuttur, örneğin Skoudis ve diğerleri [19]:

Kötü amaçlı yazılım, bilgisayarınızda çalışan ve sisteminizin bir saldırganın yapmasını istediği bir şeyi yapmasını sağlayan bir dizi talimattır.

Bu tanıma göre, bu tezde kullandığımız gibi çalıştırılabilir olması gerekmez. Donanıma uygulanabileceği için yazılım olması bile gerekmez. Tanımın ikinci bölümü çok çeşitli senaryolara atıfta bulunabilir. Bir saldırgan, örneğin sistemdeki çok sayıda değerli dosyayı silmek gibi, yalnızca zarar vermek isteyebilir. Veya amaç para olabilir, bu nedenle dosyalar şifrelenir ve kurbandan şifre çözme anahtarı için ödeme yapması istenir. Ayrıca, bir saldırının nedeni casusluk veya kredi kartı numaraları gibi bilgilerin çalınması olabilir.

Benzer ve daha yeni bir tanım Srakew ve diğerleri tarafından sağlanmıştır:

Kötü amaçlı yazılım, birçok yönden sisteme karşı savunmasız kalabilen kötü amaçlı kod veya yazılımdır.

Şaka veya vandalizm için oluşturulan en eski kötü amaçlı yazılım türlerinin aksine, günümüzün kötü amaçlı yazılımları çok farklı. Artık kötü amaçlı yazılım, büyük bir yeraltı ekonomisinin bir parçası ve yeraltı kuruluşları tarafından para kazanmak için ve hükümetler tarafından casusluk ve saldırılar için kullanılan bir araçtır [20].

Çeşitli kötü amaçlı yazılım türleri mevcuttur. Bilgi paylaşımını kolaylaştırmak için kötü amaçlı yazılımlar kategorize edilmelidir. Bu aynı zamanda, örneğin bir şirkette bir güvenlik ihlalinden sonra "temizlemeyi" kolaylaştırır. Bulunan kötü amaçlı yazılım bir rootkit ise, solucan olduğundan farklı prosedürler izlenmelidir. Ne yazık ki, gerçek bir endüstri standardı mevcut değildir [21]. Bilgisayar Antivirüs Araştırmacısı Kuruluşu (CARO), kötü amaçlı yazılımlar için bir adlandırma standardı geliştirdi, ancak bu yalnızca genel bir kılavuz görevi görüyor. Satıcıların, virüs, damlalık, truva atı, PWS (Parola çalan) ve arka kapı alt kategorilerini içeren standardı izlemesi gerekmez.

Microsoft'un daha uzun ve daha ayrıntılı olan kendi listesi vardır. Aşağıdakilerden [22] oluşur: Adware, Backdoor, Behavior, BrowserModifier, Constructor, DDoS, Dialer, DoS, Exploit, HackTool, Joke, Misleading, MonitoringTool, Program, PWS, Ransom, RemoteAccess, Rogue, SettingsModifier, SoftwareBundler , Spammer, Spoofer, Spyware, Tool, Trojan, TrojanClicker, TrojanDownloader, TrojanDropper, TrojanNotifier, TrojanProxy, TrojanSpy, VirTool, Virüs ve Solucan.

Daha yüksek bir ayrım olarak, kötü amaçlı yazılım iki ana kategoriye ayrılabilir: Virüs ve arka kapılar gibi bir ana bilgisayar programına ihtiyaç duyan parazitik kötü amaçlı yazılımlar ve solucanlar ve botlar gibi bağımsız olarak çalışabilen kendi kendine yeten programlar [23] [s. 216]. Ardından, en çok kullanılan kötü amaçlı yazılım türlerinden bazılarının açıklamasını izler.

2.1.1.1. Virüs

Bilgisayarlarla ilgili olarak virüs terimi ilk olarak 1987'de Cohen tarafından tanıtıldı [24]. Virüs, diğer programları değiştirerek [23] [s. 220]. Kendilerini diğer programlara bağlarlar ve ana bilgisayar programı yapması gerekeni yaparken arka planda çalışırlar. Üç bölümden oluşur: Bulaşma mekanizması, tetikleyici ve yük. Birincisi, virüsün "üreme" veya yayılma şeklidir. İkincisi, virüsün yükünü etkinleştirdiği veya teslim ettiği durumdur. Sonuncusu, gerçekleştirdiği kötü niyetli faaliyettir.

Birkaç tür virüs vardır. Ek olarak, onu önyükleme sektörü, dosya ve makro gibi hedeflere göre sınıflandırmak için, virüsün nasıl gizlemeye çalıştığını ayırt edebiliriz [23] [s. 224]:

• Şifrelenmiş virüs: Bu tür bir virüsle, virüs kodunun geri kalanını rastgele bir anahtarla şifreler. Kopyaladığında, farklı bir anahtar kullanır, böylece araştırmacılar tarafından sabit bir model gözlemlenemez.

• Gizli virüs: Ana anahtar, tespit edilmekten saklanmaya çalışmasıdır. Örneğin iyi huylu bir programla aynı uzunlukta olabilir. G / Ç rutinlerini kesintiye uğratarak, birisinin diskin kendi başına kullandığı kısmını okuduğunu algılayabilir ve ardından kendisini orijinal, enfekte olmamış program olarak sunabilir.

• Polimorfik virüs: Bit modellerini değiştirerek, virüs her sürüm için farklı imzalar oluşturacaktır. Gizli virüs gibi, amaç da tespit edilmekten kaçınmaktır.

• Metamorfik virüs: Polimorfik ile aynıdır, ancak hem davranış hem de görünüm değişir. Bu, yeni sürümü tespit etmeyi daha da zorlaştırır.

2.1.1.2. Worm(Solucan)

Stallings ve diğerleri, "bir solucanın kendini kopyalayabilen ve ağ bağlantıları üzerinden bilgisayardan bilgisayara kopyalar gönderebilen bir program olduğunu belirtir. Varışta solucan çoğalmak ve yeniden yayılmak için etkinleştirilebilir" [23] [s. 231]. Bu genellikle iki yoldan biriyle yapılır: Bir ağ hizmetindeki güvenlik açıklarından yararlanarak veya e-posta yoluyla [20]. Herhangi bir kaynak olmasa da ifadeyi desteklemek için, farklı sosyal medya sitelerinin artık üçüncü bir seçenek olarak hizmet ettiğini varsaymak güvenli olacaktır.

Bir virüs gibi, solucan üç aşamadan oluşur [23] [s. 231]:

1. Bulaşacak diğer sistemleri arayın.
2. Uzak sistemle bir bağlantı kurun.
3. Kendini uzaktaki sisteme kopyalayın ve kopyanın yeni sistemde çalışmasını sağlayın.

Ek olarak, yeni sisteme bulaşmadan önce yeni sisteme bulaşıp bulaşmadığını anlamaya çalışabilir.

2.1.1.3. Bot

Bot, bir bilgisayarı gizlice kontrol eden bir programdır. Robotun kısaltmasıdır ve başlangıçta uzaktan erişim truva atı olarak adlandırılmıştır [25]. Enfekte olan bir bilgisayara bot veya zombi de denir [26]. Saldırganlar genellikle aynı anda yüzlerce veya binlerce bilgisayara virüs bulaştırmayı hedefler. Bu şekilde, tüm virüs bulaşmış bilgisayarlar kontrol edilebilir ve koordineli bir şekilde kullanılabilir. Buna botnet [23] [s. 240].

Botnet'ler, Komut ve Kontrol sunucuları (C&C) kullanılarak kontrol edilir. İletişim farklı protokoller üzerinden geçebilir. Bazıları hem kanallar hem de özel mesajlar yoluyla IRC kullanır,

bazıları HTTP kullanır ve yanıt mesajlarını komut olarak yorumlarken, bazıları eşler arası (p2p) tabanlı iletişim kullanır [26].

Bir kötü amaçlı yazılım yazarının botları kullanmasının birkaç nedeni vardır. En yaygın olanları, dağıtılmış hizmet reddi (DDoS) saldırıları, spam gönderme, trafik koklama, keylogging, yeni kötü amaçlı yazılım yayma ve reklam yazılımı yükleme [23] [s. 240-241]. Son yıllarda botnet'ler bitcoin madenciliği için de kullanıldı [27].

2.1.1.4. Rootkit

Rootkit, saldırgan yöneticinin sisteme erişmesini sağlayan ve aynı zamanda varlığını gizleyen bir dizi programdır. Ad, orijinal olarak Unix / Linux'taki yönetici hesabı kökünden ve bu erişim düzeyini sağlayan bir dizi araçtan gelmektedir. Bunlara ps, netstat, ls ve passwd [28] dahildir.

Yönetici ayrıcalıkları nedeniyle, rootkit'lerin algılanması çok zor olabilir. API'lere yapılan çağrıları yakalayabilir ve yanıtları değiştirebilirler. Bu şekilde işlem monitörü, dosya listeleri ve kayıtlar yanlış bilgileri görüntüleyebilir [23] [s. 242].

2.1.1.5. Backdoor (Arka Kapı)

Arka kapı, normal güvenlik prosedürlerini atlayan bir programa giden gizli bir yoldur. Bu şekilde sisteme yetkisiz erişime izin verir. Özel bir giriş dizisi tarafından tetiklenebilir veya özel bir kullanıcı kimliği ile çalıştırılabilir [23] [s. 216].

2.1.1.6. Truva atı

Truva atı, yararlı veya zararsız görünen, ancak sandığından daha fazlasını yapan bir şeydir. Tipik örnekler, kötü amaçlı yazılım taraması yapıyormuş gibi yapan, ancak bunun yerine arka planda başka bir şey yapan sahte antivirüs programlarıdır [20]. Adını Yunan mitolojisindeki Truva atından almıştır.

Bu tür kötü amaçlı yazılımların genellikle bir arada olduğu belirtilmelidir. Örneğin, saldırıya uğrayan sisteme bir rootkit yüklemek için bir truva atı kullanılabilir.

2.1.2. İmzadan Kaçınma

Tipik olarak, anti-virüs yazılımı, kötü amaçlı yazılımları tespit etmek için imza tabanlı bir yöntem kullanır. Kötü amaçlı yazılım yürütülebilir dosyasında bulunan talimatlar, kötü amaçlı

yazılımı tanımlayan benzersiz bir imza elde etmek için ayrıştırılır ve bu daha sonra bilinen kötü amaçlı yazılım imzalarının büyük bir veritabanıyla karşılaştırılır [29,30]. Bonfante vd. bu problemle mücadele etmek için bir kontrol akış grafiği yöntemi önerdi [31]. Yaygın olarak kullanılan tüm montaj talimatları için düğümleri olan bir grafik kullandılar ve ardından kötü amaçlı yazılımları sınıflandırmak için bu grafiğin küçültülmüş bir sürümünü imza olarak kullandılar. Testlerine göre, bu algılama biçimi, grafikler daha büyük olduğunda (daha büyük yürütülebilir dosyalar için) daha iyi genel algılama doğruluğu ile sonuçlandı.

2.1.3. Kod Gizleme

Statik kötü amaçlı yazılım analizi, temel olarak anlamsal analiz ve sınıflandırma için kaynak kodu analizi açısından incelenmiştir. Moser vd. kodun anlamsal analizden gizlenmesi için basitçe opak sabitler kullanarak program kontrol akışını gizlemek için bir yöntem önerdi [32]. Bu, günümüzde mevcut olan statik kötü amaçlı yazılım analizi tekniklerindeki önemli bir kusuru vurgulamaktadır; burada semantik analiz, sabitleri gerçek zamanlı olarak hesaplamak için rastgele bir yaklaşım getirilerek yenilebilir. Bahsedilen bu tür bir yöntem, değişkenlerin depolandığı adresleri oluşturmak için rastgele bir tohum kullanmak veya işlemi papatya dizimi yapmak ve değişkenleri diğer adreslerde bulunan adreslerde depolamaktır. Koddaki belirli sabitlerin değerini belirlemek için NP-hard algoritmanın tanıtımı da bu makalede tartışılmıştır. Örneğin, kodda bir 3SAT problemi uygulamak, bu bölüm koduna giriş değişkenleri her zaman statik bir değer döndürür (0 diyelim). Bu, programın çalışma süresi sırasında 3SAT algoritmasına herhangi bir değişken atandığında her zaman 0 değeri üreteceği anlamına gelir. Bunu, kodu okuyan bir insan tarafından belirlemek kolay olsa da, anlam bilincine sahip bir analizörün bu algoritmanın tüm olası çıktılarını belirlemesi ve sonunda bunun çıktısının her zaman 0 olduğunu belirlemesi çok zordur, çünkü algoritma polinom zamanı. İkili dosyalarda birden çok kez şifreleme kullanarak kod gizleme ve ardından şifre çözme için bir aracın paketlenmesi Christodorescu ve Jha tarafından tartışılmıştır [33]. Bu tür bir gizleme biçiminin, bellekteki şifresi çözülmüş dosyayı analiz ederek çalışma süresi sırasında yakalanması kolaydır, ancak dosyanın şifresini çözmeden ve dinamik olarak analiz etmeden dosyanın şifreleme düzeyini belirlemek zordur.

Preda ve diğerleri tarafından orijinal kötü amaçlı yazılım kodu ile karmaşık hale getirilmiş kötü amaçlı yazılım kodu arasındaki benzerliği ölçmek için bir ölçüt öneren anlambilim tabanlı bir yaklaşım önerildi [34]. Ayrıca, kötü amaçlı yazılım koduna, NOP yerleştirmeye, komut ikamesine ve değişken yeniden adlandırmaya sürekli gizlemenin dahil edilmesini (NP-sabit hesaplama veya benzer yöntemler ekleyerek) saptama yöntemlerini tartıştı. Ancak, bu yaklaşımın pratik uygulaması tam olarak gerçekleştirilmemiştir. Arama grafiği analizi ve

tetikleyicilere dayalı davranış tanımlama dahil olmak üzere birçok dinamik kötü amaçlı yazılım algılama yöntemi vardır [35]. Ancak, bu yöntemler hesaplama açısından pahalıdır ve kötü amaçlı yazılımların güvenli bir şekilde yürütülebileceği ve analiz edilebileceği bir sanal alan altyapısı gerektirir.

2.1.4. Yazılım Paketleme

Dosya paketleme, büyük yazılımları küçük, kompakt bir pakette bir araya getirirken kullanılan yaygın bir tekniktir [30]. Bu tür paketleme teknikleri genellikle kötü amaçlı yazılımın kolay tanımlanmasını potansiyel olarak engelleyebilecek bir tür şifreleme içerir. PolyPack adı verilen bu tür bir araç, paketleyicilerin virüsten ve kötü amaçlı yazılımdan kaçmak için etkili bir yöntem olduğunu kanıtlamak için özel olarak tasarlanmıştır [36]. Kendilerine sağlanan verileri bağımsız olarak paketleyen 10 paketleyici sağlarlar ve ardından paketlenmiş verileri 10 iyi bilinen anti-virüs tarayıcısı ile tararlar. En iyi sonucu alan paketleyici seçilir. Çalışmaları, bunun çoğu virüsten koruma yazılımına karşı kaçınma oranlarını 2,58 kat artırdığını ortaya koydu.

2.2. Kötü Amaçlı Yazılım Algılama için Makine Öğrenimi

Makine öğreniminin daha büyük veri kümeleriyle daha iyi performans gösterdiği gerçeği iyi bilinmektedir [37]. Kötü amaçlı yazılım sınıflandırması için makine öğrenimini kullanan çeşitli çalışmalar yayınlanmıştır. Dinamik kötü amaçlı yazılım analizi için sistem çağrılarının dinamik analizi [38], kayıt defteri erişim izleme [39], gizli Markov model tabanlı analiz [40] gibi çeşitli yöntemler önerilmiştir.

Kolter ve Maloof yaklaşık 255 milyon farklı n-gram üretmek için 4 bayt dizisini birleştirerek n-gram kullanımını önerdi [41]. Makalesinde, hangi özelliklerin alakalı olduğunu belirlemek için olasılıklı bir yaklaşımın kullanılmasını önerdi ve analiz için ilk 500 n-gramı kullandı. Makale, verilerini analiz etmek için Naive Bayes, Support Vector Machine (SVM) ve J48 karar ağacının kullanılmasını önerdi. Analiz için kullanılan veriler, temel olarak Sourceforge ve VX Heavens'ten (gerçek veriler açıklanmadı), 1971 iyi huylu yürütülebilir dosya ve 1651 kötü amaçlı yürütülebilir dosya test edildi. Bu araştırmada kullanılan küçük örneklem seti ve yazarlar tarafından kullanılan kesin veri setinin mevcut olmaması gerçeği, daha büyük veri kümeleri ile kullanıldığında bu sonuçların doğruluğunu tespit etmek zordur. Benzer bir çalışma, muhtemelen büyük bir veri kümesi olan Microsoft Kötü Amaçlı Yazılım Sınıflandırması [42] ile bu yaklaşımı kullanarak Bagga tarafından yapılmıştır [43]. Ancak bu çalışma, kötü amaçlı yazılım algılama sorunu yerine kötü amaçlı yazılım sınıflandırma sorununa odaklandı.

Adobe Systems Inc. Ürün Olayı Müdahale Ekibinden Raman, taşınabilir yürütülebilir dosyalardan [44] en az ilişkili yedi özelliği çıkararak kötü amaçlı yazılımları sınıflandırmak için bir yöntem önerdi. Çıkarılan özellikler DebugSize, ImageVersion, IatRVA, ExportSize, ResourceSize, VirtualSize, NumberOfSections idi. Deneme için 100.000 kötü amaçlı yürütülebilir dosya ve 16.000 iyi huylu yürütülebilir dosya içeren bir veri kümesi kullanıldı. Bu veriler kullanılarak çeşitli modeller test edildi. Test edilen modeller arasında, J48 karar ağacı [45] en iyi sonuçları elde etti: 0,057'lik bir yanlış pozitif oranı ile 0,986'lık gerçek bir pozitif oran. Ortaya çıkan eğitimli model, kötü amaçlı yazılım sınıflandırması için ücretsiz bir araç olarak yayınlandı, ancak veri kümesi herhangi bir şekilde karşılaştırmalı araştırma yapmak için yayınlanmadı. Anderson ve Roth ayrıca bu eğitimli modeli EMBER veri seti [46] ile test ettiler ve 0,53'lük bir yanlış pozitif oranı ve 0,08'lik bir yanlış negatif oranı sergilediğini buldular.

Huang ve Stokes tarafından 2016 yılında MtNet adı verilen derin sinir ağlarını kullanan dinamik bir kötü amaçlı yazılım sınıflandırma modeli önerildi [47]. Bu çalışma için kullanılan veri kümesi, 6.5 milyon örnek dosya içeren Microsoft Corporation tarafından sağlanmıştır. Bu veri kümesinden 2,85 milyon kötü niyetli ve 3,65 milyon zararlı dosya çıkarıldı. Temel olarak iki tür veriden oluşan çalışma zamanında dosya yürütme sırasında eğitim özellikleri çıkarıldı: sistem işlevi çağrıları ve boş sonlandırılmış nesneler. Özellik seçimi, Manning ve diğerleri tarafından önerilen karşılıklı bilgiler kullanılarak gerçekleştirildi. Toplam 50.000 giriş özelliği elde etmek için [48]. Nihai hedef, kötü amaçlı yazılımları önce iyi huylu veya kötü niyetli olarak sınıflandırmak ve ardından kötü amaçlı yazılımı bilinen 100 kötü amaçlı yazılım ailesinden biri olarak sınıflandırmaktı. ReLU aktivasyon işlevi, daha iyi model performansı için eklenen çıkarma katmanları ile birlikte kullanıldı. Bu model,% 0,07'nin altında yanlış pozitif oranlarıyla etkileyici sonuçlar gösterse de, test veri setinin ve test için kullanılan model kodunun bulunmaması, bu sonuçların yeniden üretilmesini imkansız hale getirir.

Yankı durumu ağı ve yinelenen sinir ağı tabanlı kötü amaçlı yazılım sınıflandırıcıları, Pascanu ve diğerleri tarafından kötü amaçlı yazılımların dinamik analizi için test edilmiştir [49]. Araştırmaları, kötü amaçlı yazılımın dinamik analizi için sigmoid (lojistik regresyon) aktivasyon fonksiyonu ile yankı durumu ağı tabanlı tekrarlayan bir modelin kullanımını ortaya koydu. Tam giriş vektörü açıklanmadı, ancak çalışma zamanı yürütme sırasında dosyalar tarafından gerçekleştirilen API çağrılarından türetildi. Model, 0,001'lik bir yanlış pozitif oranıyla 0,983'lük gerçek bir pozitif orana ulaştı. Yazarlar, bu araştırmada kullanılan veri setinin dahili olarak sağlandığını ve halka açık olmadığını kabul ediyor. Bu araştırmanın amacı, tekrarlayan sinir ağlarının dinamik kötü amaçlı yazılım analizi için kullanılabileceğini belirlemekti. Bununla birlikte, veri setinin mevcut olmaması ve önerilen modeli yeniden üretmek için gereken adımların sağlanmaması nedeniyle, bu sonuçları doğrulamak ve buna dayalı olarak daha fazla araştırma yapmak zordur.

2.2.1. Özellik Seçimi

Makine öğrenimi, eğitim için kullanılan özellik kümesine çok duyarlıdır. Çeşitli çalışmalar, makine öğrenimi tabanlı kötü amaçlı yazılım sınıflandırıcılarının etkili eğitimi için faydalı olacak bazı özellikler ortaya koymuştur. Bu amaçla farklı yaklaşımlar incelenmiştir.

Divandari vd. dosyalardan işlem kodu verisinin çıkarılmasını ve özellik kümesini [50] özetlemek için bir Markov Blanket yaklaşımının kullanılmasını önerdi. İşlem kodlarının kendileri yürütülebilir dosyaların önemli bir parçası olduğundan, kötü amaçlı yazılım tespiti için güvenilir özellikler olarak kabul edildi [51]. Önerilen model, kötü amaçlı yazılım sınıflandırması için Gizli Markov Modeli (HMM) kullanır.

Saxe ve Berlin tarafından araştırmalarında [52] önerilen bayt histogram yaklaşımı, bir dosyadan özniteliklerin çıkarılması için biçimden bağımsız bir yöntem getirmiştir. Bu yöntem, bayt bilgilerinin bir dosyadan özellikler olarak, bu baytların gerçek işlevi hakkında bilgi gerektirmeden ayıklanmasına yönelik yenilikçi bir yaklaşımdır. Dosyada kullanılan potansiyel şifreleme veya sıkıştırmanın anlaşılmasını sağlamak için ikili dosyada bulunan tüm bayt değerlerinin histogramını 2 boyutlu bir bayt-entropi histogramıyla birlikte çıkarmayı önerir. Bu yöntemi modelimizde başlık çıkarma yöntemini tamamlamak için kullanırız, böylece taşınabilir yürütülebilir dosyadaki tüm baytları vektörleştirmek için gereken yüksek genel giderler olmadan yüksek doğruluk elde ederiz.

Weinberger ve diğerleri tarafından önerilen özellik hashing hile. [53] sık sık alıntılanmış ve makine öğrenimi modelleri için kullanılmıştır. Çoğu makine öğrenimi tabanlı model için giriş vektörü statiktir ve giriş boyutuna bağlı olarak boyut olarak artırılamaz. Bu nedenle, büyük girdi özelliklerini, eğitim için daha yönetilebilir olan statik bir boyutta etkin bir şekilde özetlemek için bir yönteme ihtiyacımız var. Özellik hashing hilesi, verilerin boyutluluğunu etkili bir şekilde düşürmek için bir yöntem önerir, böylece orijinal amaçlanan verileri hala yeterince temsil eder, ancak bir modeli etkili bir şekilde eğitmek için doğrusal ayrılabilir özellikler sunar.

2.2.2. Güçlendirilmiş Karar Ağaçları ve Yapay Sinir Ağları

Karar ağacı uzun süredir keşfedilmiş ve kullanımdadır. Bununla birlikte, karar ağacı modelleri için güçlendirme yöntemindeki son gelişmelerle birlikte, performans açısından yapay sinir ağlarına benzer veya daha iyi olduklarını kanıtladılar. Çok sayıda değişkenle iyi ayarlanması ve çalışması nispeten daha kolaydır [54]. AdaBoost'un gelişiyle birlikte, karar ağacı modellerinin artırılması, ikili sınıflandırmadan çok kategorili sınıflandırmaya geçmeyi başardı [55, 56]. Bu, yapay sinir ağları için alternatif olarak güçlendirilmiş karar ağacı tabanlı modellerin kullanılmasını teşvik etti. Bu tezde önerdiğimiz model, modelimiz için kullandığımız aynı veri seti

için mevcut bir güçlendirilmiş karar ağacı modeli ile karşılaştırılmıştır. Caruana ve Niculescu-Mizil, makalelerinde [57] vektör makinelerinin, güçlendirilmiş karar ağaçlarının ve sinir ağlarının çoğu senaryoda, varyansın esas olarak hiper parametre ayarlamasıyla sınırlı olduğu karşılaştırılabilir performansa sahip olduğunu ortaya koydu.

2.3. Yinelenen Sinir Ağlarının İncelenmesi

Tekrarlayan sinir ağları, 1990'larda araştırma ve geliştirmenin önemli bir odağı olmuştur. Sıralı veya zamanla değişen kalıpları öğrenmek için tasarlanmıştır. Tekrarlayan bir ağ, geri besleme (kapalı döngü) bağlantıları olan bir sinir ağıdır [58]. Örnekler arasında BAM, Hopfield, Boltzmann makinesi ve tekrarlayan geri yayılım ağları bulunmaktadır [59].

Tekrarlayan sinir ağı teknikleri çok çeşitli problemlere uygulanmıştır. 1980'lerin sonunda, Rumelhart, Hinton ve Williams dahil olmak üzere birçok araştırmacı tarafından karakter dizilerini öğrenmek için basit, kısmen tekrarlayan sinir ağları tanıtıldı [60]. Diğer birçok uygulama, olayların zaman dizileri ile dinamik sistemleri içeren problemleri ele almıştır.

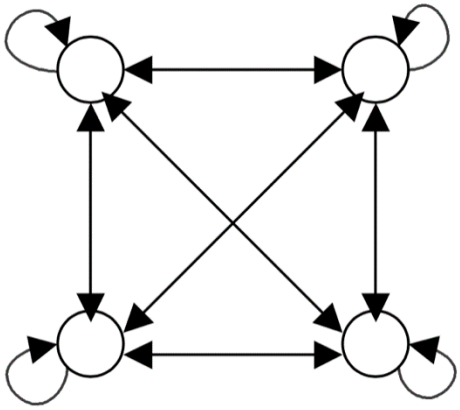
Tablo 1, yinelenen sinir ağlarının son uygulamalarının genişliği hakkında fikir vermek için başka ilginç örnekler veriyor. Örneğin, sanal gerçeklik sistemleri için insan kafasını takip etmenin dinamikleri araştırılıyor. Finansal verilerin ve elektrik enerjisi talebinin tahmin edilmesi diğer çalışmaların nesneleridir. Su kalitesini izlemek ve suyu filtrelemek için gereken katkı maddelerini en aza indirmek için tekrarlayan sinir ağları kullanılıyor. Ve müzik notalarının zaman dizileri tekrarlayan sinir ağları ile çalışıldı.

Tablo 1. Tekrarlayan sinir ağı uygulamalarına örnekler.

Topic	Authors	Reference
Predictive head tracking for virtual reality systems	Saad, Caudell, and Wunsch, II	[Saad, 1999]
Wind türbine power estimation	Li, Wunsch, O'Hair, and Giesselmann	[Li, 1999]
Financial prediction using recurrent neural networks	Giles, Lawrence, Tsoi	[Giles, 1997]
Music synthesis method for Chinese plucked-string instruments	Liang, Su, and Lin	[Liang, 1999]
Electric load forecasting	Costa, Pasero, Piglione, and Radasanu	[Costa, 1999]
Natural water inflows forecasting	Coulibaly, Anctil, and Rousselle	[Coulibaly, 1999]

2.3.1. Yinelenen Sinir Ağı Mimarileri

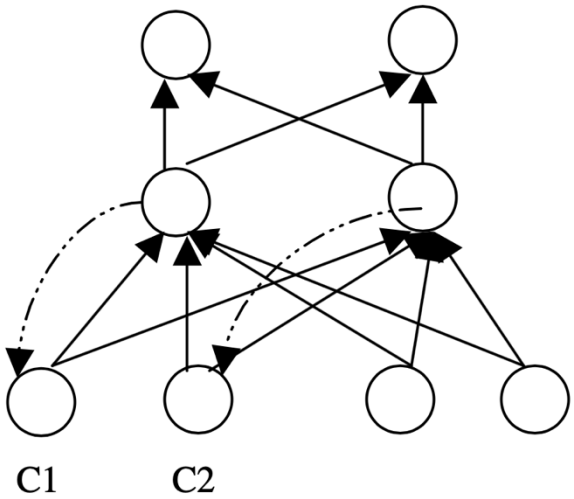
Mimariler, farklı giriş ve çıkış katmanlarına sahip çok katmanlı ileri beslemeli ağlar dahil olmak üzere, tamamen birbirine bağlı (Şekil 1) kısmen bağlı ağlara (Şekil 2) kadar çeşitlilik gösterir. Tamamen bağlı ağlar, farklı düğüm giriş katmanlarına sahip değildir ve her düğüm, diğer tüm düğümlerden gelen girdiye sahiptir. Düğümün kendisine geri bildirim mümkündür.



Şekil 1. Tamamen bağlı tekrarlayan sinir ağına bir örnek.

Karakter dizilerini öğrenmek için basit, kısmen tekrarlayan sinir ağları (Şekil 2) kullanılmıştır. ALBazı düğümler ileri besleme yapısının parçası olmasına rağmen, diğer düğümler sıralı bağlamı sağlar ve diğer düğümlerden geri bildirim alır. Bağlam birimlerinden (C1 ve C2) alınan ağırlıklar, örneğin geri yayılım kullanılarak giriş birimleri için olanlara benzer şekilde işlenir. Bağlam birimleri, Şekil 2 durumunda, ikinci katman birimlerinden zaman gecikmeli geri bildirim alır. Eğitim verileri girdilerden ve bunların istenen ardıl çıktılarından oluşur. Ağ, bir karakter dizisindeki sonraki harfi tahmin etmek ve bir karakter dizisini doğrulamak için eğitilebilir.

İleri beslemeli çok katmanlı sinir ağlarına geri bildirim eklemek için iki temel yol kullanılabilir. Elman [61] gizli katmandan girdi katmanının bağlam kısmına geri bildirim getirdi. Bu yaklaşım, girdi değerlerinin sırasına daha fazla dikkat eder. Jordan tekrarlayan sinir ağları [62] çıktı katmanından girdi katmanının bağlam düğümlerine geri bildirimi kullanır ve çıktı değerleri dizisine daha fazla vurgu yapar.



Şekil 2. Basit bir tekrarlayan ağ örneği.

2.3.2. Yinelenen Sinir Ağlarında Öğrenme

Öğrenme, sinir ağlarının temel bir yönüdür ve sinirsel yaklaşımı, başlangıçtan beri yapay zekâ için zor bir hedef olan uygulamalar için bu kadar çekici kılan önemli bir özelliktir. Öğrenme algoritmaları uzun zamandır araştırmanın odak noktası olmuştur [63,64].

Hebbian öğrenme ve gradyan kökenli öğrenme, sinir ağı tekniklerinin dayandığı temel kavramlardır. Gradyan inişinin popüler bir tezahürü, Rumelhart [60] ve Werbos [65] tarafından sunulan geri-hata yayılımıdır. Geri yayılımın uygulanması nispeten basit olsa da, pratik uygulamalarda kullanımında, yerel minimumda tuzaktan kaçınmanın zorluğu da dahil olmak üzere çeşitli sorunlar ortaya çıkabilir. Girdi verilerinin zaman gecikmeli güncellemesinden tekrarlayan sinir ağlarında dinamik işlemenin ek karmaşıklığı, öğrenmeyi temsil etmek için daha karmaşık algoritmalar gerektirir.

Tekrarlayan sinir ağlarının dinamik olarak işlenmesinin avantajını gerçekleştirmek için bir yaklaşım, sabit kalıpları işleyen ileri beslemeli ağların etkinliğini geliştirmektir. Araştırmacılar, gradyan yöntemlerinin ve özellikle geri yayılım öğrenmenin tekrarlayan sinir ağlarına genişletilebileceği çeşitli şemalar geliştirdiler. Werbos, gradyan yöntemlerini kullanan bir dizi statik ağlar olarak yinelenen bir sinir ağının zaman evrimini yaklaşık olarak tahmin eden zaman yaklaşımı yoluyla geri yayılımı [66] tanıttı. Başka bir yaklaşım, orijinal dinamik köle ağının çekicilerinin programlanmasında gerekli hesaplamaları gerçekleştirmek için ikinci bir ana, sinir ağını konuşlandırır [67]. Araştırılan diğer teknikler, Pineda [68], Almeida [69], Williams ve Zipser [70], Sato [71] ve Pearlmutter [72] 'da bulunabilir. Geri yayılım öğrenimini tekrarlayan ağlara genişletmeye yönelik çeşitli girişimler Pearlmutter'da [73] özetlenmiştir.

2.4. Tasarım Konuları ve Teorisi

2.4.1. Optimizasyon

Optimizasyon problemlerinin gerçek zamanlı çözümlerine sinyal işleme, sistem tanımlama, filtre tasarımı, fonksiyon yaklaşımı ve regresyon analizi dahil olmak üzere bilimsel ve mühendislik problemlerinde sıklıkla ihtiyaç duyulur ve sinir ağları bu amaçla geniş çapta araştırılmıştır. Karar değişkenlerinin ve kısıtlamalarının sayısı genellikle çok büyüktür ve büyük ölçekli optimizasyon prosedürleri, dinamik bir sistemin performansını optimize etmek için gerçek zamanlı olarak yapılması gerektiğinde daha da zordur. Bu tür uygulamalar için, klasik optimizasyon teknikleri, problem boyutluluğu ve hesaplama süresinin katı gereksinimleri nedeniyle yeterli olmayabilir. Sinir ağı yaklaşımı, optimizasyon problemlerini, genel amaçlı dijital bilgisayarlarda yürütülen en popüler optimizasyon algoritmalarından daha büyük büyüklük sıralarındaki çalışma sürelerinde çözebilir.

Xia ve Wang'ın araştırmaları, bu sorunlar için sinir ağlarının kullanımını incelemekte ve küresel yakınsama ile optimizasyon sinir ağı modellerini tasarlamak için birleşik bir yöntem sunmaktadır. Doğrusal ve ikinci dereceden programlamayı çözmek ve doğrusal tamamlayıcı problemleri çözmek için sürekli zaman tekrarlayan sinir ağlarını tartışırlar ve ardından ayrık zamanlı sinir ağlarına odaklanırlar. Atama sinir ağları ayrıntılı olarak tartışılmış ve sinir ağlarının çalışma özelliklerini göstermek için bazı simülasyon örnekleri sunulmuştur.

Çalışmalarında ilk olarak doğrusal ve ikinci dereceden programlama problemlerini (LP ve QP) çözmek için ilk çift sinir ağlarını sunar ve doğrusal tamamlayıcı problemleri (LCP) çözmek için sinir ağını geliştirmişlerdir. Sinir ağı modellerini tasarlamak için birleşik bir yöntemi takiben, bölümün ilk kısmı LP ve QP'yi çözmek için sürekli zamanlı ilk-ikili tekrarlayan sinir ağlarını ayrıntılı olarak açıklamaktadır. Çalışmalarının ikinci kısmında ise, QP ve LCP için birincil-ikili ayrık zamanlı sinir ağlarına odaklanmışlardır.

Optimizasyon için sinir ağlarının kullanımında büyük ilerleme kaydedilmiş olmasına rağmen, birçok teorik ve pratik problem çözülmeden kalmıştır. Optimizasyon problemleri için tekrarlayan sinir ağlarının dinamikleri, tekrarlayan sinir ağlarının pratik problemlere daha fazla uygulanması ve optimizasyon için tekrarlayan sinir ağlarının donanım prototiplemesi üzerine gelecekteki araştırmalar için alanlar tanımlanmıştır.

2.4.2. Ayrık Zaman Sistemleri

Santos ve Von Zuben, parametreleri ayarlamak için optimizasyon prosedürlerine dayanan verimli denetimli öğrenme algoritmaları için pratik gereksinimi tartışıyor. Performansı iyileştirmek için ikinci dereceden bilgiler eğitimdeki hatayı en aza indirmek için düşünülmüştür.

Çalışmalarının ilk amacı, bir dizi yinelenen sinir ağı konfigürasyonu için kesin ikinci dereceden bilgi elde etmenin sistematik yollarını, birinci dereceden bilgi edinme maliyetinden yalnızca iki kat daha yüksek bir hesaplama maliyetiyle açıklamaktır. İkinci amaç, mevcut ikinci dereceden bilgileri etkili bir şekilde araştırmak için kullanılabilen eşlenik gradyan algoritmasının geliştirilmiş bir versiyonunu sunmaktır.

Tekrarlayan bir sinir ağının dinamikleri zaman içinde sürekli veya ayrık olabilir. Bununla birlikte, dijital hesaplama cihazlarında sürekli zamanlı tekrarlayan bir sinir ağının simülasyonu, ayrık zamanlı eşdeğer bir modelin benimsenmesini gerektirir. Uzamsal-zamansal temsil için ortaya çıkan doğrusal olmayan modeller, doğrusal olmayan fark denklemleri sistemi aracılığıyla bir dijital bilgisayarda doğrudan simüle edilebilir. Denklemlerin doğası, benimsenen tekrarlayan mimarinin türüne bağlıdır, ancak daha az sayıda parametre ve ilişkili denklemlerle bile çok karmaşık davranışlara yol açabilir.

Pratik önemi olan tekrarlayan sinir ağlarının analizi ve sentezi çok zorlu bir görevdir ve eğitim sürecinde ikinci dereceden bilgiler dikkate alınmalıdır. Çok çeşitli tekrarlayan sinir ağı mimarileri için kesin ikinci dereceden bilgi elde etmek için düşük maliyetli bir prosedür sunarlar. Ayrıca, mevcut ikinci dereceden bilgileri keşfetmek için etkili bir şekilde kullanılabilen, ölçeklendirilmiş eşlenik gradyan algoritmasının geliştirilmiş bir versiyonu olan çok verimli ve genel bir öğrenme algoritması sunarlar. Sabit olanların yerine bir dizi uyarlanabilir katsayı sunarlar ve algoritmanın yeni parametreleri otomatik olarak ayarlanır. Bazı simülasyon sonuçlarını gösterir ve yorumlarlar.

Bu çalışmanın yenilikçi yönleri, düşük bir hesaplama maliyetiyle bir dizi farklı tekrarlayan sinir ağı mimarileri için kesin ikinci dereceden bilgi elde etmek için sistematik bir prosedürün önerilmesi ve ölçeklendirilmiş bir eşlenik gradyan algoritmasının geliştirilmiş bir versiyonudur. Önemli bir husus, kesin ikinci dereceden bilgi verildiğinde, öğrenme algoritmasının, belirli bağlama herhangi bir uyarlama olmaksızın doğrudan uygulanabilmesidir.

2.4.3. Bayesian İnanç Revizyonu

Hopfield sinir ağı, nesne tanımadan grafik düzlemleştirmeye ve yoğunlaştırıcı atamasına kadar çok sayıda optimizasyon problemi için kullanılmıştır. Bununla birlikte, Hopfield enerji fonksiyonunun ikinci dereceden düzende olması, uygulanabileceği sorunları sınırlar. Bazen,

Hopfield'ın ikinci dereceden enerji işlevine indirgenemeyen nesnel işlevler, ikinci dereceden bir enerji işlevi ile makul bir şekilde yaklaşık olarak tahmin edilebilir. Diğer problemler için, amaç fonksiyonu daha yüksek seviyeli bir enerji fonksiyonu ile modellenmelidir.

Abdelbar, tekrarlayan sinir ağlarını anlatıyor ve seyrek yüksek sıralı ağlar için verimli bir uygulama veri yapısı sağlıyor. Ayrıca, bu tür ağların Bayesçi inanç revizyonu için ve belirsizlik altında teşhis muhakemesi ve sağduyulu muhakemedeki önemli problemlerde nasıl kullanılabileceğini açıklar.

2.4.4. Bilgi Temsili

Giles, Omlin ve Thornber çalışmalarında birçok uygulama alanında faydalı hale gelen nöro-bulanık sistemleri (yapay sinir ağlarının bulanık mantıkla birleşimini) incelemişlerdir. Bununla birlikte, geleneksel nöro-bulanık modellerin genellikle bağlam ve durum gerektiren uygulamalar için (örneğin, konuşma, zaman serisi tahmini ve kontrol) gelişmiş temsil gücüne ihtiyaç duyduğunu açıklarlar. Bu uygulamalardan bazıları, sonlu durum otomatı olarak kolayca modellenebilir. Önceden, deterministik sonlu durum otomatının (DFA), DFA yapısını doğrudan sinir ağının ağırlıklarına programlayarak tekrarlayan sinir ağları tarafından sentezlenebileceği veya eşlenebileceği kanıtlanmıştı. Bu sonuçlara dayanarak, bulanık sonlu durum otomatını (FFA) tekrarlayan sinir ağlarına eşlemek için bir sentez yöntemi öneriyorlar. Bu eşleme, VLSI'de doğrudan uygulama, yani VLSI sistemlerinde DFA'nın kodlamasının bir genellemesi olarak FFA'nın kodlanması için uygundur.

Sentez yöntemi, FFA'nın tekrarlayan ağlarla eşleştirilmeden önce bir dönüşüme uğramasını gerektirir. Nöronlar, FFA durumlarının bulanık bir temsilini barındırmak için zenginleştirilmiş bir işlevsellikle sağlanır. Bu zenginleştirilmiş nöron işlevselliği, FFA'nın bulanık parametrelerinin doğrudan sinir ağının parametreleri olarak temsil edilmesine de izin verir.

Ayrıca, ağ ağırlığının sonlu değerleri için inşa edilen sinir ağlarının bulanık sonlu durum dinamiklerinin kararlılığını kanıtlar ve simülasyonlar yoluyla kanıtların ampirik doğrulamasını sağlarlar. Bu, sinirsel ve bulanık sistemler ve otomata modelleri arasındaki çeşitli bilgi denkliği temsillerini kanıtlıyor.

2.4.5. Uzun Vadeli Bağımlılıklar

Tekrarlayan sinir ağları için gradyan-iniş öğrenme algoritmalarının, uzun vadeli bağımlılıkları içeren görevlerde, yani istenen çıktının geçmişte çok uzak zamanlarda sunulan girdilere bağlı olduğu problemlerde kötü performans gösterdiği bilinmektedir. Lin, Horne, Tino

ve Giles bunu incelemişler ve güçlü temsil yeteneklerine sahip NARX tekrarlayan sinir ağları adı verilen bir mimari sınıf için uzun vadeli bağımlılıklar sorununun azaldığını göstermişlerdir.

Daha önce gradyan-iniş öğrenmenin NARX ağlarında, gramatik çıkarım ve doğrusal olmayan sistem tanımlama dahil olmak üzere problemlerde "gizli durumlara" sahip tekrarlayan sinir ağlarından daha etkili olabileceğini bildirdiler. Tipik olarak ağ çok daha hızlı yakınsar ve diğer ağlardan daha iyi genelleştirir ve bu bölüm aynı türden sonuçları gösterir.

NARX ağlarının bilgileri geleneksel tekrarlayan sinir ağlarından iki ila üç kat daha uzun süre saklayabildiğini gösteren bazı deneysel sonuçları da sunuyorlar. NARX ağlarının uzun vadeli bağımlılık sorununu aşmamasına rağmen, uzun vadeli bağımlılık problemlerinde performansı büyük ölçüde artırabileceklerini gösteriyorlar. Bilgiyi sağlam bir şekilde tutmanın ne anlama geldiğine ilişkin bazı varsayımları ayrıntılı olarak açıklarlar ve bu varsayımları gevşetmek için olası yollar önerirler.

2.5. Uygulamalar

Bu başlık altında, tekrarlayan sinir ağlarının ilginç modifikasyonlarına ve uygulamalarına bakılmıştır. Yörüngeler, kontrol sistemleri, robotik ve dil öğrenimiyle ilgili problemler, kaotik sistemlerde tekrarlayan sinir ağlarının ilginç bir kullanımı ile birlikte dahil edilmiştir.

2.5.1. Kaotik Yeniden Kazanan Ağlar

Dayhoff, Palmadesso ve Richards, kaotik sistemler için tekrarlayan sinir ağlarının kullanımına ilişkin çalışmaları yapmışlardır. Dinamik sinir ağları, sonlu durum salınımları, sınır döngüleri ve kaotik davranış gibi çok çeşitli salınımlar yapabilirler. Mümkün olan farklı salınımlar, kendi kendini sürdüren muazzam bir aktivite paternleri repertuvarı yaratır. Bu repertuar çok ilgi çekicidir çünkü salınımlar ve değişen aktivite modelleri potansiyel olarak hesaplama amaçlı ve fiziksel olayları modellemek için kullanılabilir.

Bir dış model bir uyarıcı olarak kullanıldığında kaotik bir ağda gözlemlenen eğilimleri araştırıyorlar. Model uyaranı, tek katmanlı tekrarlayan bir ağdaki tüm nöronlara sabit bir harici girdidir. Uyaranın gücü, uyarılmış salınımların karmaşıklığında değişiklikler ve eğilimler üretmek için çeşitlidir. Daha güçlü uyaranlar, daha basit ve daha az çeşitli salınımları uyandırabilir. Gürültüye karşı direnç, gürültülü uyaranlar aynı veya benzer salınımları uyandırdığında ortaya çıkar. Daha güçlü uyaranlar gürültüye karşı daha dayanıklı olabilir. Bu gözlemlerin her birinin örneklerini gösterirler. Bir model-salınım haritası sonunda model tanıma ve diğer hesaplama amaçları için kullanılabilir. Böyle bir paradigmada, dış model uyarıcısı, bir model ilişkilendirme problemine yanıt olarak ağdan okunan bir salınımı çağrıştırır. Bu tür bir hesaplama

paradigmasının çok katmanlı statik ileri beslemeli bir ağdan daha yüksek desen kapasitesi ve sınır esnekliği potansiyeline sahip olduğuna dair kanıtlar sunarlar.

2.5.2. Dil Öğrenimi

Kremer, dilbilgisi indüksiyonu veya dil öğrenimi ile tekrarlayan sinir ağları arasındaki ilişkiyi inceler ve resmi dil öğrenmenin anlaşılmasının tekrarlayan sinir ağlarının tasarlanmasına ve uygulanmasına nasıl yardımcı olabileceğini sorar. Bu sorunun cevabı dört ders şeklinde gelir: (1) RNN'leri eğitmek zordur, (2) arama alanını azaltmak öğrenmeyi hızlandırabilir veya mümkün kılabilir, (3) arama alanını sipariş etmek öğrenmeyi hızlandırabilir ve (4) eğitim verilerinizi sipariş etmek yardımcı olur. Bu bölüm, zamanla değişen girdilerle sunulan ve zamanın çeşitli noktalarında çıktıları işlemek için tasarlanmış dinamik tekrarlayan sinir ağlarıyla ilgilidir. Bu durumda, ağın çalışması, bir girdi dizisini bir çıktı değerine veya değerler dizisine eşleyen bir işlevle tanımlanabilir ve girdilerin geçerli değerlerden oluşan ayrı bir alfabeden seçildiği ve çıktı değerlerinin ayrı ayrı düştüğü soruna uygulanır. kategoriler. Her bir öğenin bir girdi alfabesinden seçildiği girdi dizileriyle başa çıkma sorunu da biçimsel bir dil sorunu olarak değerlendirilebilir. Bu çalışma, bir giriş dilinin alt kümelerini sınıflandırmak için tekrarlayan sinir ağlarını kullanıyor ve dil öğrenimi için etkili teknikleri ortaya koyuyor.

2.5.3. Sıralı Otomatik Birleştirme

Bağlantıcı Doğal Dil İşleme (NLP) üzerine artan araştırmalara rağmen, uygun dilsel temsillerin geliştirilmesi gibi bir dizi sorunun çözülmesi gerekiyor. Doğal dil, altta yatan hiyerarşik yapıya ve ardışık dış görünüme sahip dinamik bir sistemdir ve yeterli bir hiyerarşik sistematik dilsel temsil yöntemine ihtiyaç duyar. Elman [61] tarafından Jordan Yinelenen Ağlar [62] ve Basit Yinelenen Ağlar (SRN) gibi küresel bellek yinelenen sinir ağlarının geliştirilmesi, bu küresel sistemdeki sıralı girdilerinin temsillerini kademeli olarak oluşturan modellerin geliştirilmesini teşvik etti.

Stoianov, karmaşık sıralı verilerin statik dağıtılmış temsillerinden oluşan hiyerarşik bir sistem oluşturmak ve işlemek için tasarlanmış yeni bir bağlantısal mimari sunuyor. Giriş dizisinin karmaşık statik temsillerini oluşturma fikrini takip eder, ancak her girdi dizisi için benzersiz temsiller oluşturarak bu statik gösterimleri orijinal biçimlerinde yeniden üretmek üzere genişletilmiştir. Model, Tekrarlayan Otomatik İlişki Ağları (RAN'lar) adı verilen sıralı otomatik ilişkilendirme modüllerinden oluşur. Bu modüllerin her biri, girdi dizilerini yeniden üretmeyi öğrenir ve bir yan etki olarak dizilerin statik dağıtılmış temsillerini geliştirir. İstenirse, bu modüller statik gösterimleri orijinal sıralı biçimlerine açarlar. Sıralı olarak temsil edilen

hiyerarşik girdi verilerini işlemeye yönelik eksiksiz mimari, bir dizi RAN'dan oluşur. Bu kademeli şemadaki en düşük seviyeden herhangi birinden bir RAN modülünün girdi jetonları, daha düşük seviyeden RAN modülünün ürettiği statik temsillerdir. En düşük seviye RAN modülünün giriş verileri dış dünyadan algılanır. En düşük seviyeden bir modülün çıktısı bir efektör ile ilişkilendirilebilir. Daha sonra, RAN gizli katmanına ayarlanmış bir statik temsil verildiğinde, bu efektör, paket açma işlemi sırasında sırayla komutları alacaktır.

RAN, doğal dillerin dinamiklerine uyan tekrarlayan bir sinir ağıdır ve RAN'lar, dizilerin temsillerini üretir ve bunları sıralı biçimlerine geri döndürerek yorumlar. Bir RAN dizisi olan daha genişletilmiş mimari, doğal dillerdeki hiyerarşiye benzer. Ayrıca, temsili bir eğitim ortamı verildiğinde, bu mimari, dağıtılmış temsilleri sistematik bir şekilde geliştirme kapasitesine sahiptir. RAN'ların bir sistematiklik açıklaması sağladığını ve bu nedenle RAN ve RAN kademesinin, ürettikleri dağıtılmış temsillerin kapsamlı bir şekilde dönüştürüldüğü ve ilişkilendirildiği daha küresel bir bilişsel modele katılabileceğini savunuyor.

Bu çalışmada dinamik verilerdeki hiyerarşi tartışmasını içerir ve hecelerin temsillerini geliştirmek için küçük bir RAN örneği sunulmuştur. Model, hiyerarşik olarak yapılandırılmış dizilerin temsillerini geliştirme sorununu çözse de özellikle otonom bir bilişsel model geliştirmek için bazı sorular açık kalmaktadır. Yine de önerilen model, bağlantısal modellemede önemli bir adım olabilir.

2.5.4. Eğitim Sorunları

Tekrarlayan sinir ağlarının önemli bir uygulaması, belirli gerekli zaman ilişkileri olan olayların iyi örnekleri olan yörüngeleri içeren dinamik sistemlerin modellenmesidir. Tipik test durumları, dairenin ve sekiz şeklinin ünlü doğrusal olmayan ve otonom dinamik sistemleridir.
Tekrarlayan ağları eğitmedeki zorluk, genellikle verimsiz eğitimle sonuçlanabilecek tahminlerin kullanılmasıyla sonuçlanır. Sundareshan, Wong ve Condarcure çalışmalarında gradyan değerlendirmeleri gerektirmeyen iki alternatif öğrenme prosedürünü açıklamaktadır. Sürekli yörüngeler üretmek için karmaşık bir uzay-zamansal öğrenme görevini kullanarak iki algoritmanın performansını gösterirler. Uygulamada önemli avantajlar gösterirler.

İki farklı yaklaşımı tanımlarlar. Biri otomatik öğrenme teorisindeki kavramları kullanır, diğeri ise klasik simpleks optimizasyon yaklaşımına dayanır. Eğitimli bir sinir ağı tarafından uzay-zamansal sinyal üretimi görevi ile bu yaklaşımların eğitim verimliliğini gösterirler. Bu görevin karmaşıklığı, tekrarlayan sinir ağlarının zamansal dinamiklere yaklaşma konusundaki benzersiz yeteneğini ortaya koyuyor.

Hagner, Hassoun ve Watta tek katmanlı tamamen tekrarlayan ağlar ve harici yinelemeli çok katmanlı ağlar dahil olmak üzere farklı ağ mimarilerini ve öğrenme kurallarını

karşılaştırmışlardır: artımlı gradyan inişi, eşlenik gradyan inişi ve genişletilmiş Kalman filtresinin üç versiyonu. Daire yörüngesinin, sekiz şeklindeki yörünge zor olduğu halde nispeten kolay öğrenildiği gösterilmiştir. Bu dahili ve harici olarak tekrarlayan otonom sistemlerin sinir ağı yaklaşımlarının kalitatif ve kantitatif bir analizini verirler.

2.5.5. Adaptif Robot Davranışı

Ziemke, robot kontrolü ve öğrenimi için tekrarlayan sinir ağlarının kullanımını tartışıyor ve bunun bilişsel bilim, yapay zekâ ve robot kontrol sistemleri mühendisliği dahil olmak üzere farklı araştırma alanlarıyla olan ilişkisini araştırıyor. Şimdiye kadar robotlarda nadiren kullanılan ikinci dereceden RNN'ler özellikle ayrıntılı olarak tartışılmış ve uyarlanabilir robot davranışını gerçekleştirme kapasiteleri gösterilmiş ve deneysel olarak analiz edilmiştir.

2.6. Gelecekteki Yönlendirmeler

Bu çalışma, tekrarlayan sinir ağlarına olan ilginin genişliğini ve derinliğini temsil ediyor ve devam eden araştırmalar için çeşitli yönlere işaret ediyor. Bölümler hem yeni hem de geliştirilmiş algoritmaları ve tasarım tekniklerini ve ayrıca yeni uygulamaları ele almaktadır. Konular dil işleme, kaotik ve gerçek zamanlı sistemler, optimizasyon, yörünge problemleri, filtreleme ve kontrol ve robotik davranış ile ilgilidir.

Tekrarlayan sinir ağlarında yapılan araştırmalar, 1980'lerin sonlarında önemli temel çalışmalara dayanarak, esas olarak 1990'larda gerçekleştirildi. Önümüzdeki on yıl, teori ve tasarımda önemli gelişmelerin yanı sıra önemli pratik sorunların yaratıcı çözümü için daha birçok uygulama üretmelidir. Tekrarlayan sinir ağlarının yaygın olarak uygulanması, araştırma ve geliştirmeye daha fazla ilgi uyandırmalı ve daha fazla teorik ve tasarım sorusu doğurmalıdır. Hibrit sistemlere olan ilginin devam etmesi, tekrarlayan sinir ağlarının yeni ve daha güçlü kullanımlarıyla sonuçlanmalıdır.

2.7. Taşınabilir Yürütülebilir Formatın Tanımlanması

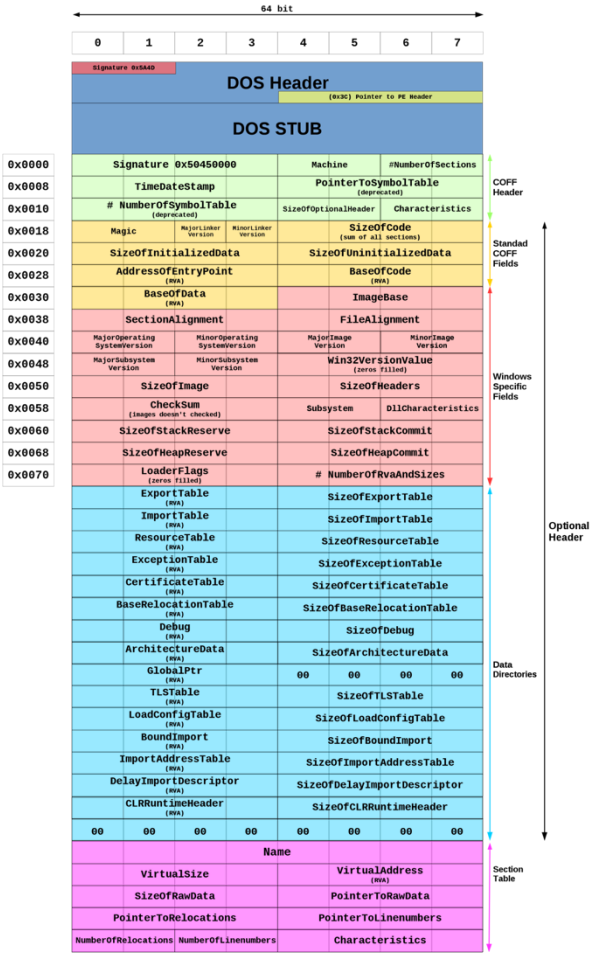
Taşınabilir yürütülebilir (PE) biçimi (Şekil 3), Microsoft tarafından Windows NT 3.1 işletim sistemiyle tanıtıldı. Başlangıcından bu yana, onu Windows'un daha yeni sürümlerine dahil etmek için çeşitli iyileştirmeler gördü. Unix, Windows PE formatına benzer ELF formatını kullanır.

Bu tezin kapsamı, Unix tabanlı işletim sistemlerinde çalışan kötü amaçlı yazılımlar için mevcut veriler sınırlı olduğundan, Windows çalıştırılabilir dosyalarıyla sınırlıdır. Ancak, PE dosyalarında bulunan COFF başlığı hem Unix hem de Windows ortamları için ortaktır [74].

Önerdiğimiz model, dosyanın kötü niyetli mi yoksa zararsız mı olduğunu belirlemek için PE dosyalarından çıkarılan özellikleri analiz eder. Bu bölüm PE dosyalarından elde edilebilecek bilgileri açıklamaktadır.

2.7.1. MS-DOS Koçanı

Bu saplama, dosya bir MS-DOS ortamında her yürütüldüğünde yürütülür. Tek amacı, dosyanın MS-DOS ortamında çalıştırılamayacağını belirten bir mesaj yazdırmaktır. MS-DOS Stub'dan sonra eklenen bir imza, dosyanın PE biçiminde olduğunu gösterir. [75]



Şekil 3. PE Dosya Formatı [85]

2.7.1.1. Ortak Nesne Dosya Formatı

Ortak Nesne Dosyası Biçimi (COFF) başlığı, MS-DOS Koçanının hemen ardından bulunur. COFF Başlığı yapısı Tablo 2'de tanımlanmıştır. COFF başlığındaki Makine alanı ve Özellikler alanı için tüm olası değerler sırasıyla Tablo 3 ve Tablo 4'te tanımlanmıştır. Dosya yalnızca makine alanı üzerinde yürütülecek hedef makineyle eşleşirse bir makinede yürütülebilir.

Tablo 2. COFF: Yapısı [75]

Offset	Size	Field	Description
0	2	Machine	Identifies the target machine that the executable can run on. Refer to Table 3.2
2	2	NumberOfSections	Size of the section table. (follows the header table)
4	4	TimeDateStamp	Date of Creation. Represented as seconds after January 1, 1970.
8	4	PointerToSymbolTable	File offset of COFF symbol table. 0 for no table.
12	4	NumberOfSymbols	Number of entries in the symbol table.
16	2	SizeOfOptionalHeader	Size of the optional header (required for executables)
18	2	Characteristics	Indicates the attributes of the file. Refer to Table 3.3.

2.7.1.2 İsteğe Bağlı Başlık

Yürütülebilir dosyalar (resimler) olarak kabul edilen dosyaların ek bir isteğe bağlı başlığı vardır. Bu başlık, çalıştırılabilir dosyaların yürütülmesinden sorumlu olan işletim sisteminde bulunan yükleyiciye bilgi sağlar. Bu başlık çalıştırılabilir dosyalar için gerekli olsa da, nesne dosyalarında da mevcut olabilir. Nesne dosyalarındaki isteğe bağlı başlıklar, dosya boyutunu artırmak dışında hiçbir amaca hizmet etmez.

İsteğe bağlı başlığın boyutu, COFF başlığındaki SizeOfOptionalHeader alanında tanımlanır. Tablo 5'te gösterildiği gibi, isteğe bağlı başlıkta bulunan sihirli bir sayı, yürütülebilir dosyanın PE32 mi yoksa PE32 + mı olduğunu belirler.

E32 + yürütülebilir dosyaları 64 bit bellek adres alanına izin verir, ancak boyut olarak 2 gigabayttan fazla olamaz. İsteğe bağlı başlık, Tablo 6'da tanımlanan 3 ana bölüme ayrılmıştır.
İsteğe bağlı başlıktaki standart alanlar, her COFF uygulaması (Windows ve Unix) için tanımlanmıştır. Bu bölümde yer alan bilgilerin bir özeti aşağıdadır:

- Dosyanın normal bir çalıştırılabilir (0x10B), bir ROM görüntüsü (0x107) veya bir PE32 + çalıştırılabilir (0x20B) olduğunu gösteren sihirli sayı.
- Bu PE dosyası için kullanılacak bağlayıcı sürümü.

Tablo 3. COFF: Makine Tipleri [75]

Constant	Value	Description
IMAGE_FILE_MACHINE_UNKNOWN	0x0	Applicable to any machine
IMAGE_FILE_MACHINE_AM33	0x1d3	Matsushita AM33
IMAGE_FILE_MACHINE_AMD64	0x8664	X64
IMAGE_FILE_MACHINE_ARM	0x1c0	ARM little endian
IMAGE_FILE_MACHINE_ARM64	0xaa64	ARM64 little endian
IMAGE_FILE_MACHINE_ARMNT	0x1c4	ARM Thumb-2 little endian
IMAGE_FILE_MACHINE_EBC	0xebc	EFI byte code
IMAGE_FILE_MACHINE_I386	0x14c	Intel 386 or equivalent
IMAGE_FILE_MACHINE_IA64	0x200	Intel Itanium processor family
IMAGE_FILE_MACHINE_M32R	0x9041	Mitsubishi M32R little endian
IMAGE_FILE_MACHINE_MIPS16	0x266	MIPS16
IMAGE_FILE_MACHINE_MIPSFPU	0x366	MIPS with FPU
IMAGE_FILE_MACHINE_MIPSFPU16	0x466	MIPS16 with FPU
IMAGE_FILE_MACHINE_POWERPC	0x1f0	Power PC little endian
IMAGE_FILE_MACHINE_POWERPCFP	0x1f1	Power PC with floating point support
IMAGE_FILE_MACHINE_R400	0x166	MIPS little endian
IMAGE_FILE_MACHINE_RISCV32	0x5032	RISC-V 32-bit address space
IMAGE_FILE_MACHINE_RISCV64	0x5064	RISC-V 64-bit address space
IMAGE_FILE_MACHINE_RISCV128	0x5128	RISC-V 128-bit address space
IMAGE_FILE_MACHINE_SH3	0x1a2	Hitachi SH3
IMAGE_FILE_MACHINE_SH3DSP	0x1a3	Hitachi SH3 DSP
IMAGE_FILE_MACHINE_SH4	0x1a6	Hitachi SH4
IMAGE_FILE_MACHINE_SH5	0x1a8	Hitachi SP5
IMAGE_FILE_MACHINE_THUMB	0x1c2	Thumb
IMAGE_FILE_MACHINE_WCEMIPSV2	0x169	MIPS little-endian WCE v2

Tablo 4. COFF: Mevcut Özellik İşaretleri [75]

Flag	Value	
IMAGE_FILE_RELOCS_STRIPPED	0x0001	The file must be loaded at its preferred base address because it does not allow base relocation.
IMAGE_FILE_EXECUTABLE_IMAGE	0x0002	Set for valid files. Linker error if this is not set.
IMAGE_FILE_LINE_NUMS_STRIPPED	0x0004	Deprecated. Set to zero.
IMAGE_FILE_LOCAL_SYMS_STRIPPED	0x0008	Deprecated. Set to zero.
IMAGE_FILE_FILE_AGRESSIVE_WS_TRIM	0x0010	Obsolete for Windows 2000 and later. Set to zero.
IMAGE_FILE_LARGE_ADDRESS_AWARE	0x0020	Capable of handling addresses more than 2GB.
	0x0040	Reserved.
IMAGE_FILE_BYTES_RESERVED_LO	0x0080	Little Endian. Deprecated. Set to zero.
IMAGE_FILE_32BIT_MACHINE	0x0100	Machine uses 32-bit architecture.
IMAGE_FILE_DEBUG_STRIPPED	0x0200	File does not have debug infor- mation.
IMAGE_FILE_REMOVABLE_RUN_FROM_SWAP	0x0400	Copy the image to memory if it is on removable media.
IMAGE_FILE_NET_RUN_FROM_SWAP	0x0800	Copy the image to memory if it is on network media.
IMAGE_FILE_SYSTEM	0x1000	System File
IMAGE_FILE_DLL	0x2000	DLL File. Cannot be executed.
IMAGE_FILE_SYSTEM_ONLY	0x4000	Only support uniprocessor ma- chine.
IMAGE_FILE_BYTES_RESERVED_HI	0x8000	Big Endian. Deprecated. Set to zero.

Tablo 5. Optimal Başlık Sihir Numarası [75]

Magic number	PE format
0x10b	PE32
0x20b	PE32+

Tablo 6. Optimal Başlık Parçaları [75]

Offset (PE32/PE32+)	Size (PE32/PE32+)	Header part	Description
0	28/24	Standard fields	Common for Windows and Unix COFF implementations
28/24	68/88	Windows-spesific fields	Defines Windows spesific features.
96/112	Variable	Data directories	Address and size of special tables used by OS.

• Kod bölümünün boyutu. Kod bölümü, dosya yürütüldüğünde çalıştırılacak gerçek yazılımı içeren bir PE dosyasının metin bölümünü ifade eder. Dosya içinde bu tür birden çok kod bölümü olabilir, bu durumda başlık alanı, birleştirilmiş tüm kod bölümlerinin toplam boyutunu gösterecektir. Kod bölümleri, bir PE dosyasının .text bölümü olarak da adlandırılır.
• Dosyada bulunan başlatılmış ve başlatılmamış verilerin boyutu. Bu aynı zamanda bir PE dosyasının .data bölümü olarak da adlandırılır.
• Dosyanın giriş noktasının adresi. PE dosyası belleğe yüklendiğinde komut işaretçisinin başlayacağı yer burasıdır. [75]

Windows'a özgü alanlar, özellikle Windows ortamları için gerekli olan belirli bilgileri içerir. İşletim sistemi sürümünü, görüntü sürümünü (örneğin Word sürüm 8.0), başlıkların boyutunu, görüntünün boyutunu, DLL özelliklerini, yükleyici bayraklarını, veri dizininin uzunluğunu, veri dizininin kendisini ve sağlama toplamını içerir. Görüntünün boyutu, görüntünün çalışması için işletim sistemi tarafından ne kadar adres alanı ayrılması gerektiğini belirler. [74]

Veri dizinleri, Windows için gerekli olan dizinlerin adresini ve boyutunu verir. Buna, bunlarla sınırlı olmamak üzere, içe / dışa aktarma tabloları, kaynak tablosu, istisna tablosu vb. Dahildir. [75]

2.7.1.3. Bölüm Tablosu

PE dosyasındaki her bölüm, boyutu 40 bayt olan bir bölüm başlığı içerir. Bu, bölümün adını, sanal boyutunu, satır sayısını ve çeşitli işaretçileri (çizgiler, ham veriler, yer değiştirmeler, vb.) Tanımlar [75].

Yukarıda açıklanan bölümlerin yanı sıra, PE dosyası yazılımın çalıştırılabilir kodunu içerir. Dosyaya bağlı olarak dahil edilebilecek birkaç başka bölüm vardır, ancak bu tezin kapsamı dışındadır.

2.7.1.4. x86/x64 mimarisi

Temel montaj kodunu anlamak için aşina olunması gereken birkaç husus vardır. Bunlar yazmaçlar, veri türleri, komut seti ve Windows temelleridir.
Mimaride sekiz genel amaçlı kayıt (GPR) vardır. Kayıtları ve normalde ne için kullanıldıklarını listeleriz:
EAX - Aritmetik işlemler.

EBX - Veri işaretçisi.

ECX - Döngülerde sayaç.

EDX - Dizi / bellek işlemlerinde kaynak.

EDI - Dizi / bellek işlemlerinde hedef.

ESI - Akış işlemlerinde kaynağa yönelik işaretçi.

EBP - Temel çerçeve işaretçisi. Bu, yığın içindeki karelere işaret eder. Çerçeveler, işlevler için verileri depolar.

ESP - Yığın işaretçisi. Bu, işlem yığınının tepesine işaret eder.

Veri türleri

Yaygın veri türleri

• Bayt - 8 bit, örneğin AL, BL ve CL'de saklanır.

• Word - 16 bit, örneğin AX, BX ve CX'te saklanır.

• Çift kelime - 32 bit, örneğin EAX, EBX ve ECX'te saklanır.

Dört sözcük de kullanılabilir. 64 bit elde etmek için iki kaydı birleştirerek oluşturulurlar.

Komut seti

Veriler beş şekilde taşınabilir ve depolanabilir. Kaydetmek için hemen, hemen belleğe, kayıttan kayıda taşınabilir, kayıt ve bellek arasında hareket ettirilebilir ve bellekten belleğe taşınabilir. Verileri taşırken, sözdizimi bir işlem kodu, hedef ve bir kaynak işlenenden oluşur.

Aritmetik işlemler kullanılarak gerçekleştirilir.

• ADD - belirli bir değer ekler.

• SUB - belirli bir değeri çıkarır.

• INC - 1 ekler.

• Aralık - 1'i çıkarır.

Ve bir dizi mantıksal talimat:

• AND - ve verilen bir değer.

• OR - veya belirli bir değer.

• XOR - belirli bir değeri xors.

• NOT - belirli bir değerdeki bitleri ters çevirir.

Yığın da belirtilmelidir. Yığın, push ve pop'u destekleyen son giren ilk çıkar veri yapısıdır. İtme, yığının en üstüne bir şey koyar ve pop yığının üstünden bir şey çıkarır. ESP tarafından işaret edilen bitişik bir hafıza bölgesidir ve aşağı doğru büyür.

Akış kontrolü söz konusu olduğunda, if / else, switch / case ve while / for gibi üst düzey yapılar:

• CMP - Birini diğerinden çıkararak iki işlenen karşılaştırır

• TEST - Aralarında AND kullanarak iki işlenen karşılaştırır

• JMP - ESP'yi belirli bir adresle günceller

- JCC - Bir dizi atlama komutu
- EFLAGS

3. MATERYAL VE YÖNTEM

3.1. Veri seti

Yinelenen sinir ağları genellikle zaman serisi ya da sekans halinde ilerleyen veriler için kullanılan bir yapay sinir ağı yöntemidir. Veri setinin kaliteli olması özellikle bu tarz çalışmalarda oldukça önemlidir ve titizlikle davranılmalıdır. Bu yüzden en çok saldırılan işletim sistemi olan Windows seçilmiş ve yine derli toplu kaliteli bir veri seti olması dolayısıyla Microsoft'un sunduğu ve Microsoft Malware Classification Challenge (BIG 2015) ekinliği için hazırlanan veri seti kullanılmıştır [76].

Windows işletim sistemi için ele alınacak taşınabilir yürütülebilir dosya formatındaki zararlı yazılımların bu yöntemle işlenebilmesi için ham verinin işlenmesi gerekmektedir. Bu yüzden de zararsız olmayan taşınabilir yürütülebilir dosyaların makine dili kodlarına dönüştürülme işlemi IDA Pro adlı program kullanılarak gerçekleştirilmiştir. Zararlı olan yazılımlar hali hazırda Microsoft tarafından dönüştürülmüş haliyle veri setinde sunulduğu için bu işleme tabi tutulmamıştır. Buradan elde edilen işlem kodu sekanslarında örüntü aranarak zararlı yazılım tespit edilmeye çalışılacaktır. Yaklaşık 500 GB boyutundaki veri seti sıkıştırılarak boyutu küçültülmüş ve test ile eğitim verileri ayrılmıştır. Veri setinde 9 farklı virüs ailesinden 10868 adet .bytes dosyası, 10868 adet de .asm dosyası olmak üzere 21.736 adet dosya bulunmaktadır. Her bir zararlı yazılımın kimlik numarası, 20 karakterlik özet değeri, sınıfı ve bu 9 aileyi temsil eden bir sayı değeri bulunmaktadır. Bu zararlı yazılım aileleri ve sayı değerleri tablodaki gibidir:

Tablo 7. Zararlı yazılım aileleri ve sayı değerleri

Sıra	Aile
1	Ramnit
2	Lollipop
3	Kelihos_ver3
4	Vundo
5	Simda
6	Tracur
7	Kelihos_ver1
8	Obfuscator.ACY
9	Gatak

Her bir statik yürütülebilir zararlı yazılımın ham verileri, ikili dosyaların onaltılık sistemde gösterimi ile taşınabilir yürütülebilir başlığı verilmeden hazırlanmıştır. Ayrıca günlükleri içeren meta veri bildirimleri (fonksiyon çağrıları vs.) de bu veri setinde ikili şekilde bulunmaktadır.

Veri setinde bulunan 9 aileden kaç adet örnek olduğuna baktığımız zaman aşağıdaki grafik ortaya çıkmıştır.

Şekil 4. Zararlı yazılım örneklerinin dağılımları

Şekil 4'e bakıldığında özellikle Ramnit, Lollipop ve Kelihos_ver3 türünden zararlı yazılım türü örnek sayısının diğer türlere göre oldukça fazla olduğu ve bu da veri setinde bir sınıf dengesizliği oluşturduğu gözlemlenmektedir. Bu nedenle Simda adlı virüs ailesinden 42 adet bulunmasından ötürü tüm ailelerden 42 adet rastgele örnek seçilmiştir.

Üzerinde çalışılacak veri seti hazırlanırken toplamda 378 adet zararlı yazılım, 125 adet ise zararlı olmayan yazılım örneği kullanılmıştır. Zararlı olmayan yazılımlar taşınabilir yürütülebilir formatta Windows üzerinde çalışabilen rastgele dosyalardan seçilmiştir.

3.2. Yöntem

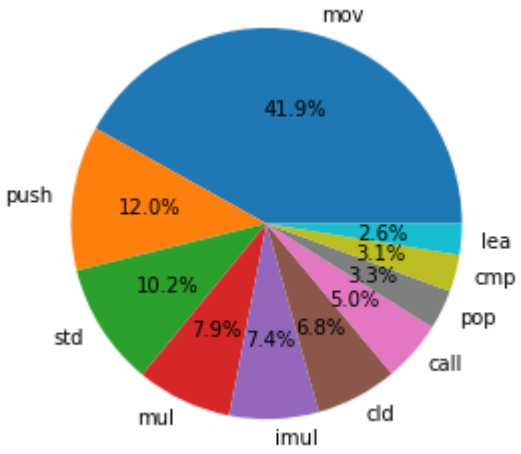
Kullanılacak veri seti seçilirken yinelenen sinir ağlarının kullanılabilirliğine dikkat edilmiştir. Veri seti zararlı yazılımların işlem kodu sekansları şeklinde düzenlendikten sonra yinelenen sinir ağlarının kullanımına uygun hale getirilmiştir. Bu başlık altında veri setini hazırlamak için yapılan tüm işlemler ve kullanılan platformlar anlatılacaktır.

3.2.1. Verilerin Hazırlanması ve Özellik Çıkarımı

Özellik çıkarımı örüntü tanıma aşamalarından biridir ve sınıflandırma işleminin kalitesini belirlemede doğrudan etkilidir. Python programlama dili kullanılarak taşınabilir yürütülebilir dosya örnekleri özellik çıkarımına uygun hale gelecek şekilde parçalara ayrılmıştır. Tüm dosyalar

sadece işlem kodları kalacak şekilde ayıklanmış ve bu ayıklama işleminin doğruluğunu sağlamak amacıyla da Intel ve AMD işlemcilerin kullanma kılavuzlarındaki referans listesi kullanılmıştır [77]. Elde edilen bu parçalar metin dizisine dönüştürülerek işlem kodu sekansları referans seti ile eşleştirilmiş ve böylece assembly dosyasından elde edilecek gereksiz kodlar elenmiştir.

Kod sekanslarına bakıldığında assembly dosyaları içerisinde bulunan işlem kodlarının tekrar eden kısımlara sahip olduğu, API çağrı isimlerini içerdiği ve her dosyanın birbirinden bağımsız büyüklüklere sahip olduğu görülmüştür. Bu nedenle bu kod sekansları içerisinden tekrar eden ve gereksiz olan işlem kodları silinmiş, her bir dosyadan gelen kod sekansının büyüklüğü belirli bir boyut ile sınırlandırılmıştır. Bu boyuttan büyük olan dosyalar için geri kalan kodlar alınmazken, küçük olanlar içinse eksik kalan kısımlar sıfır kullanılarak doldurulmuştur. Bunun en büyük sebebi makine öğrenme algoritmaları için verilecek verinin boyutunun düzenli olmasıdır.



Şekil 5. Ramnit adlı virüs ailesine ait bir zararlı yazılımın assembly komutlarındaki en sık kullanılan 10 işlem kodu

Şekil 5'te veri seti içerisindeki Ramnit ailesine ait bir zararlı yazılım içerisinde en çok kullanılan 10 adet işlem koduna ait kullanım frekansı grafiği verilmiştir. Genellikle "mov" ve "push" komutlarının oldukça sık kullanıldığı görülmüştür. Bazı işlem kodları ise oldukça az kullanıma sahiptir. Frekans belirleme işlemi tüm veri setindeki assembly dosyalarına uygulanarak çok az kullanılan işlem kodları elenmiş ve veri setinin boyutu küçültülerek işlem hızı artırılmaya çalışılmış ve daha doğru sonuçlar elde etmek amaçlanmıştır.

3.2.2. Word2Vec

Yinelenen sinir ağları, yapısı gereği metin dizilerini yorumlayamazken sadece sayısal değerleri girdi alabilir. Bu yüzden de işlem kodu sekansları metin olarak değil, sayısal vektörler

olarak ifade edilmelidir. Bunu yapmak için kullanılan en basit yöntemlerden biri "one-hot encoding" yöntemidir. Bu yöntem genellikle makine öğrenmesi algoritmalarında çalıştırılmak üzere hazırlanan verinin kategorik değişkenler içermesi durumunda bu değişkenleri ikili olarak temsil ederken kullanılır.

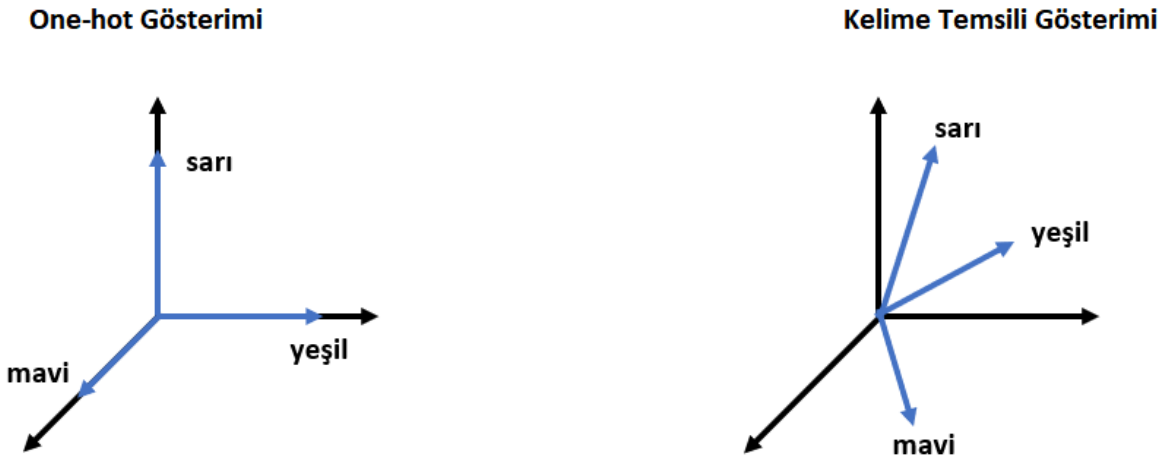
Renk
Sarı
Yeşil
Kırmızı
Mavi
Sarı

Sarı	Yeşil	Kırmızı	Mavi
1	0	0	0
0	1	0	0
0	0	1	0
0	0	0	1
1	0	0	0

Şekil 6. One-hot Encoding örneği

One-hot encoding öncelikle kullanılacak sözlüğün boyutunda boş bir vektör yaratır. Burada sözlük olarak bahsedilen şey aslında her bir kategorik değişkeni içeren bir kümedir ve bu kümedeki veriler sütunlara yerleştirilir. Ardından verinin kullanıldığı yerde 1 kullanılmadığı yerde 0 olarak vektörü doldurur [78]. Ancak bu yöntemin bazı dezavantajları bulunmaktadır.

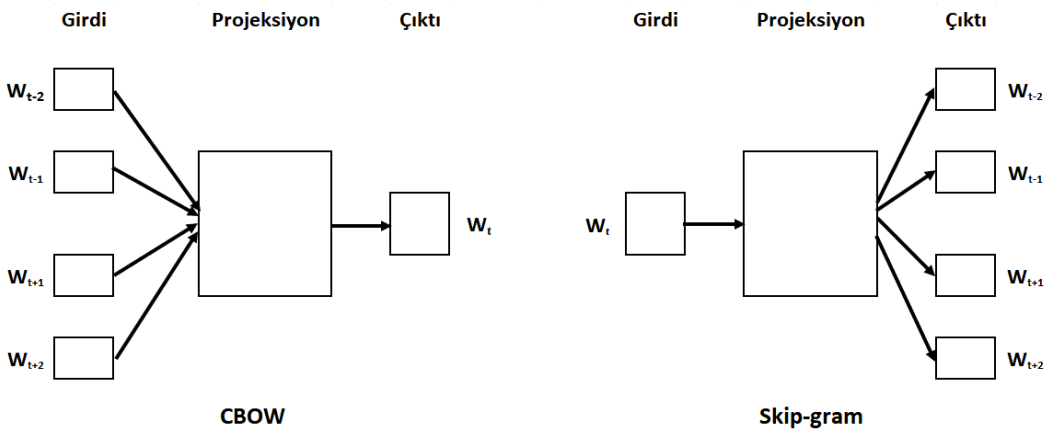
Bunlardan ilki tüm işlem kodlarını aynı vektörle ifade etmektir ki bir işlem kodu eğer diğer bir işlem koduyla ilişkili ise one-hot encoding bunu tespit edemez. İkincisi ise her bir veri için uzun bir vektör oluşacağından hesaplama süresi artacaktır. Bu yüzden de daha iyi bir temsil yöntemi olarak Word2Vec kullanılmıştır.



Şekil 7. One-hot encoding ile kelime temsili farkı

Kelime temsili (word embedding) dil modelleme yöntemlerinden biri olup girdi olarak aldığı metinleri her bir sözcüğün arasındaki mesafeyi de dikkate alarak temsil uzayında yüksek boyutlu sayısal vektörlere dönüştürür. Bu çalışmada popüler gözetimsiz ve tahmin temelli doğal

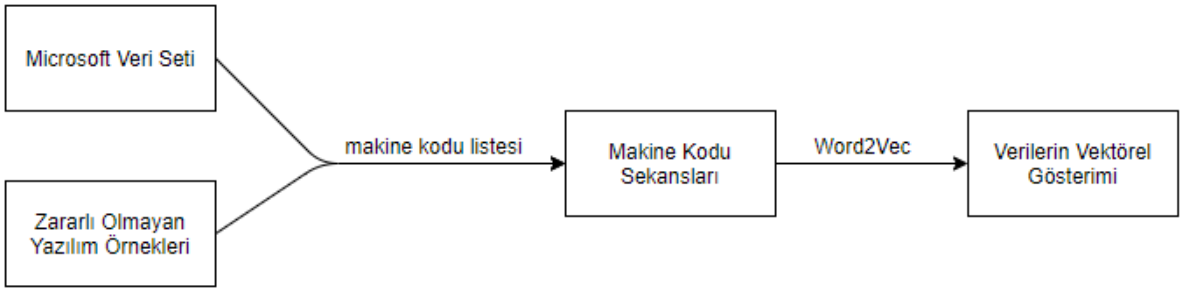
dil işleme modellerinden Word2Vec kullanılmıştır [79]. Bu yöntemin en büyük avantajı, vektör uzayında temsil edilen her bir kelimenin birbirlerine olan benzerliğini kosinüs benzerliği denilen hesaplama ile matematiksel olarak tespit edebilmesidir. Bu model çıktı olarak bir tüm kelime vektörlerini barındıran bir kelime haznesi verir. Word2Vec uygulamaları sadece doğal dil işleme alanında değil genetik, müzik listesi, beğeniler, sosyal medya grafları gibi birçok alanda da kullanılmaktadır.



Şekil 8. CBOW ve Skip-gram modelleri

Word2Vec çıktı olarak verdiği kelime haznesini kullanarak farklı şekillerde yorumlamalar yapabilir. Bir kelimenin hangi bağlama ait olduğunu ya da bir bağlama ait en olası kelimeleri tahminleyebilir. Genellikle Word2Vec 2 farklı mimari içerir:

- **Sürekli Kelime Torbası Modeli (Contextual Bag-of-Words - CBOW):** Bu model metin içerisindeki her bir cümleyi girdi olarak alıp daha sonrasında bir sözcüğün bu bağlamla alakalı olup olmadığını tahmin etmeye çalışır.
- **Gram Atlama (Skip-Gram - SG):** Bu model ise tam tersi olarak bir kelimeyi kullanarak bu kelimenin bağlı bulunduğu bağlama ait kelimeleri tahmin eder.



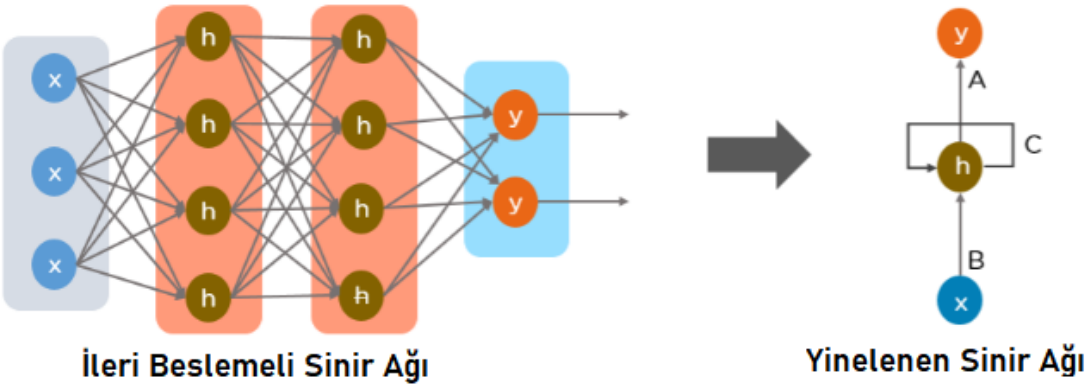
Şekil 9. Verilerin işlenmesini özetleyen akış şeması

Şekil 9.'da verilerin işlenip hazırlanmasına dair geçen sürecin bir görsel akış şeması verilmiştir. Vektörel gösterim elde edildikten sonra LSTM modeline girdi olarak verilmiştir.

3.2.3. Yinelenen Sinir Ağları (Recurrent Neural Network)

Yinelemeli sinir ağı ya da RNN, ileri beslemeli sinir ağlarının içerisinde hafıza barındıran versiyonu olarak tanımlanabilir. Aslında diğer sinir ağları gibi her bir girdi için aynı fonksiyonu kullanarak bir çıktı üretir ancak bu çıktı bir sonraki aşamanın girdisi olarak verilir. Çıktı hesaplandıktan sonra kopyalanır ve yinelenen ağa geri verilir. Ancak girdilerin tümünün birbirinden bağımsız olduğu diğer ileri beslemeli sinir ağlarının aksine bir hafıza barındırarak girdileri sekanslar halinde işler. Bu yapısından ötürü genellikle el yazması tespiti, ses tanıma gibi alanlarda sıklıkla kullanılırlar [80].

Şekil 10.'da ileri beslemeli bir sinir ağının nasıl yinelemeni bir sinir ağına dönüştürüldüğü gösterilmektedir.



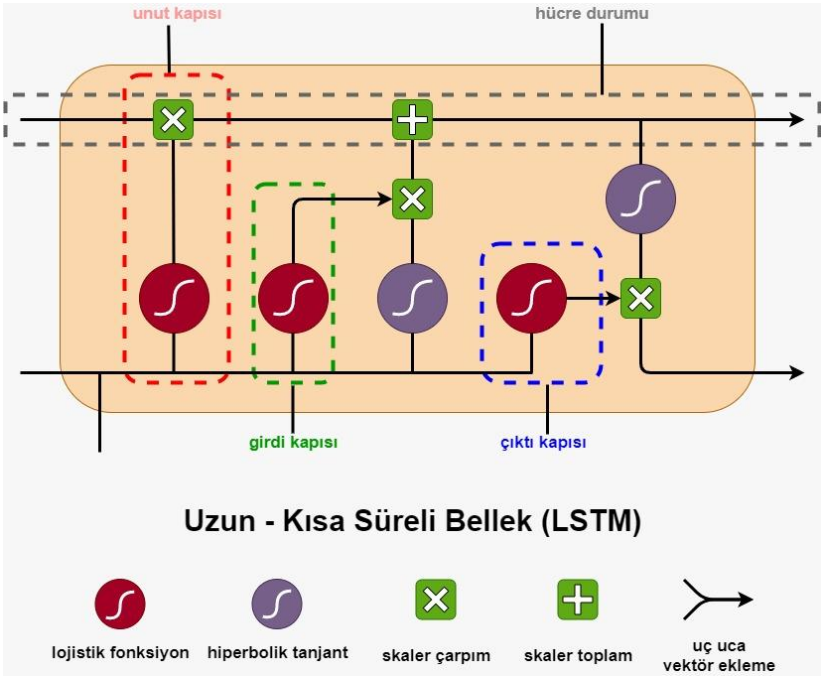
Şekil 10. İleri beslemeli sinir ağının yinelenen sinir ağına dönüşümü

RNN modeli birbirine bağlı sekanslar için yapısı itibariyle oldukça kullanışlıdır. Herhangi bir girdi uzunluğu ile kullanılabilir. Modelin kendi büyüklüğü girdi büyüklüğü ile doğru orantılı bir şekilde artmaz, ancak geçmiş veri sayısı arttıkça işlem hızı da etkilenir. Bu yüzden RNN ağını eğitmek zordur ve bazı aktivasyon fonksiyonları uzun sekansları işlemeye uygun değildir [81]. Bu nedenle tez çalışmasında RNN'in farklı bir versiyon olan LSTM modeli kullanılacaktır.

3.2.3.1. LSTM

Sınıflandırma işlemi için yinelenen sinir ağları modeli kullanılmıştır. Kullanılacak veri seti oldukça uzun sekanslar içerdiğinden bu tip ağlarda sıklıkla görülen kaybolan gradyan problemine

yani uzun sekanslar içeren derin ağlarda türevlerin sıfıra yaklaşması problemine (vanishing gradient) yol açabilir. Bu yüzden de buna bir çözüm olarak önerilen ve yinelenen sinir ağlarının farklı bir versiyonu olan LSTM (kısa ve uzun süreli hafıza) modeli kullanılmıştır. Bu modelin klasik yinelenen sinir ağlarından farkı hem kısa hem de uzun süreli hafızaya sahip olmasıdır. Bu hafızanın oluşmasını sağlayan da LSTM modelinde bulunan farklı kapılardır. LSTM modellerinde 4 farklı birim bulunur: girdi kapısı, çıktı kapısı, unut kapısı ve hücre durumu. Hücre durumu verinin geçmişini tutan hafıza kısmıdır ve diğer LSTM hücrelerine güncel veri sağlamakla görevlidir. Girdi kapısı ise gelen verileri 0 ve 1 arasındaki değerlere normalize ederek güncellenmesi gereken değerlere karar veren birimdir. Unut kapısı hafızadan veri atma işlemini gerçekleştirir. Çıktı kapısı ise hiperbolik tanjant fonksiyonu kullanarak işleme giren veri ile güncel hücre durumundan hangi verinin seçileceğine karar veren birimdir [82].



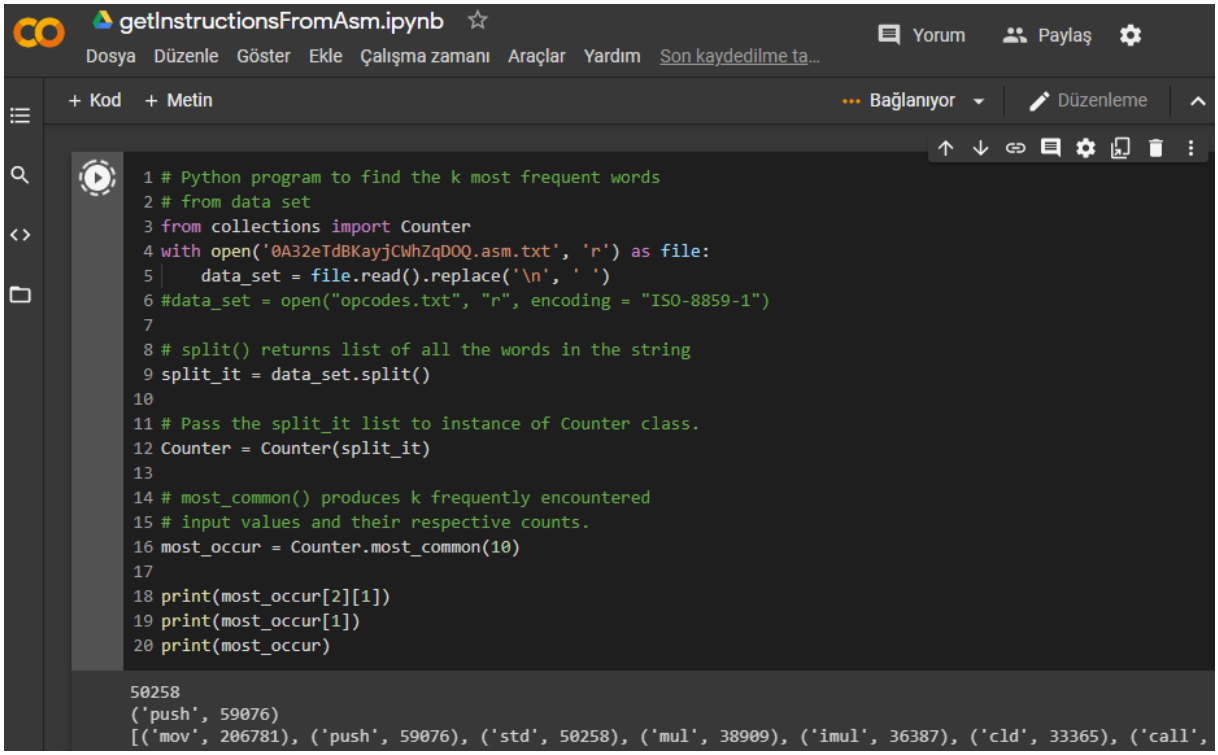
Şekil 11. Uzun-Kısa Süreli Bellek (LSTM)

İşlem kodu sekansları metin şeklinde olduğundan öncelikle bu verilerin işlenebilmesi için nümerik şekilde ifade edilmesi gerekmektedir. Farklı işlem kodu değerleri için farklı sayılar belirlenerek ve bunları bir sözlük olarak depolayarak bu dönüştürme işlemi gerçekleşmiştir. Daha sonra bu sekanslar yinelenen sinir ağları kullanılarak sınıflandırma işlemine geçilmiştir. Hazırlanan model işlem kodunda şüpheli bulduğu sekansları ve örüntüleri tespit edip öğrenerek kararlar vermeye çalışacaktır. Bunun için ise verinin %10'u test için, %90'ı ise öğretim aşaması için ayrılmıştır. Yinelenen sinir ağının çıktısı bize taşınabilir yürütülebilir dosyanın yararlı mı zararlı mı olduğuna karar verecektir.

LSTM modeli oluşturulurken nöron sayısı 128, seyreltme değeri olarak da 0.2 belirlenmiştir. Nöron sayısının fazla olması genellikle işlem hızını düşürmekle beraber daha iyi sonuçlar verir. Ancak belirli bir doğruluk oranına ulaşmış bir modelin nöron sayısını artırmak doğruluk oranını artırmamakla beraber sadece performansın düşmesine neden olacaktır. Bu yüzden nöron sayısı için 128 kullanılması uygun bulunmuştur. Seyreltme oranı ise LSTM içerisindeki nöronlara belirli oranda gürültü eklenerek aşırı uyum gösterme problemini ortadan kaldırma amaçlı kullanılan bir parametredir. Yine nöron sayısında olduğu gibi eğitim tur sayısı arttıkça çok küçük ölçeklerde performans etkileneceğinden tüm bu parametrelerle LSTM ağı farklı eğitim turları ile eğitilerek performansın artık değişmediği sayıda bırakılacaktır.

3.2.3.2. Google Colaboratory

Google Colaboratory ya da kısa adıyla Colab, Google'ın sunduğu bulut hizmetlerinden biri olup derin öğrenme algoritmalarını kullanmak için gerekli tüm altyapıya sahip ve tarayıcı üzerinden kullanılabilen ücretsiz bir platformdur. Hiçbir kurulum gerektirmeden Python programlama dili geliştirme ortamlarından Jupyter Notebook [https://jupyter.org/] servisinin kullanımına imkân tanır. Colab hizmeti ile kullanıcılara bir Linux sanal makinesi atanır ve bu makine üzerinde derin öğrenme algoritmalarının Python kütüphaneleri Keras, Tensorflow, PyTorch gibi oldukça sık kullanılan kütüphaneler bulunur [83].



Şekil 12. Google Colaboratory servisi arayüzü

Yapılan tüm çalışmalar Google'ın bulut depolama servisi Google Drive kullanılarak kaydedilebilir. Ayrıca Github gibi platformlardan da proje aktarımı yapılabilir.
İşlem gücü olarak da oldukça iyi performanslı bir makine sunan Colab, NvidiaK80, T4, P4 ve P100 gibi birçok güçlü grafik kartının gücünden yararlanmaya olanak tanır [84].

3.2.3.3. Değerlendirme Ölçütleri

Yapılacak olan sınıflandırma zararlı yazılım olup olmamasına dair olduğundan bu bir ikili sınıflandırma problemidir. Bu yüzden sonuçlar değerlendirilirken karmaşıklık matrisi kullanılarak bir tabloda sınıflandırılmıştır.

Tablo 8. Karmaşıklık matrisi kullanılarak değerlendirme

		Gerçek Veriler	
		Zararlı	**Zararsız**
Tahmin Edilen Veriler	**Zararlı**	Doğru Pozitif (DP)	Yanlış Pozitif (YP)
	Zararsız	Yanlış Negatif (YN)	Doğru Negatif (DN)

Karmaşıklık matrisinde kullanılan değerlere bakacak olursak:

- Doğru Pozitif (DP): Modelin bir örneği zararlı yazılım olarak tahmin etmesi ve o örneğin gerçekten de zararlı yazılım olması durumunu temsil eder.
- Doğru Negatif (DN): Modelin bir örneği zararsız yazılım olarak tahmin etmesi ve o örneğin gerçekten de zararsız yazılım olması durumunu temsil eder.
- Yanlış Pozitif (YP): Modelin bir örneği zararlı yazılım olarak tahmin etmesi ancak o örneğin zararsız yazılım olması durumunu temsil eder.
- Yanlış Negatif (YN): Modelin bir örneği zararsız yazılım olarak tahmin etmesi ancak o örneğin zararlı yazılım olması durumunu temsil eder.

Bu tablodaki veriler kullanılarak doğruluk, kesinlik ve duyarlılık değerleri aşağıdaki formüllere göre hesaplanmış, sonuçlara bakılarak da modelin performansı değerlendirilmiştir.

- Doğruluk: Doğru sınıflandırmaların göreceli sıklığını ifade eder ve doğru tahminlerin tüm veri setine oranı ile hesaplanır. Genellikle basit ve yeterli olmasından ötürü performans değerlendirmelerinde sıkça kullanılır. Ancak veri setinde sınıf dengesizliği varsa bu değer performansı net olarak yansıtmayabilir.
  Doğruluk = (DP + DN) / (DP + DN + YP + YN)

- Duyarlılık: Doğru olarak sınıflandırılması gereken işlemlerin ne kadarının doğru olarak tahmin edildiğini gösteren bir orandır.
  Duyarlılık = DP / (DP + YN)

- Kesinlik: Doğru sınıflandırılmış pozitif örneklerin göreceli sıklığını ifade eder. Kesinlik değeri doğru sınıflandırmaların oranını gösterir. Düşük bir keskinlik değeri veri setindeki sınıf dengesizliğine işaret olabilir.
  Kesinlik = DP / (DP + YP)

Modelin ne kadar iyi çalıştığını yüksek doğruluk değerinden, yüksek duyarlılıktan ve yüksek kesinlik değerlerinden anlamak mümkün olacağından bu değerler incelenmiştir.

4. BULGULAR VE TARTIŞMA

BIG2015 veri seti bir önceki başlıkta anlatıldığı üzere makine öğrenme algoritmalarında kullanılacak üzere yeniden düzenlenip birtakım işlemler sonucu hazır hale gelmiştir.
Yapılan çalışmalarda genellikle Python programlama dili kullanılmış olup, yapay zeka uygulamalarını çalıştırmak için sağladığı ücretsiz altyapısı ve hazır kütüphanelerinin bulunmasından ötürü yüksek işlem gücü de sağlayan Google Colaboratory hizmeti kullanılmıştır.

Word2Vec modelinden bahsederken değinilen Geri Atlama ve Sürekli Kelime Torbası modellerinin farklı pencere boyutlarında nasıl sonuçlar verdiği ve bu Word2Vec modellerinden hangisinin zararlı yazılım tespit ederken daha iyi sonuçlar vereceğine dair testler yapılmıştır. Genel itibariyle CBOW modeli, Skip-gram modeline göre daha iyi sonuçlar vermiştir. Ayrıca CBOW modeli uygun görüldükten sonra Tablo 9. incelendiği takdirde en iyi sonuçların pencere boyutunun 15 olduğu çalışmada elde edildiği görülmektedir. Bu yüzden pencere boyutu olarak da 15 belirlenmiştir.

Tablo 9. Kelime Penceresi Boyutuna göre Sürekli Kelime Torbası ve Geri Atlama modellerinin sonuçları.

	Sürekli Kelime Torbası	**Geri Atlama**
5	94.21	94.01
10	94.23	94.22
15	94.80	94.43
20	94.65	94.50
35	94.02	94.00

Word2Vec için gerekli parametrelere karar verildikten sonra LSTM modeli için kullanılacak parametreler belirlenmiştir. Bu parametreler ve değerleri Tablo 10.'daki gibidir.

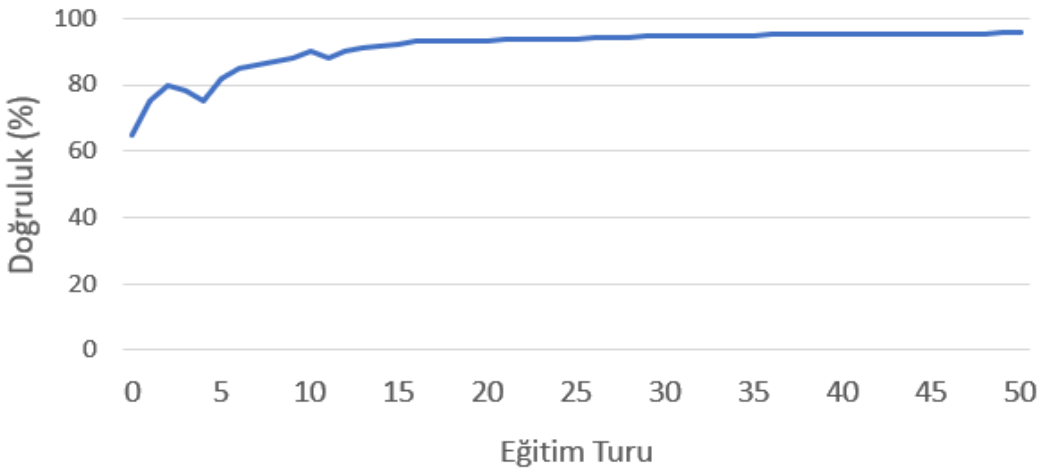
Tablo 10. LSTM modeli için kullanılacak parametreler

Parametre	**Değer**
max sequence length	600
batch size	64
embedding size	300
learning rate	1e-3
dropout rate	0.2
number of layers	2
hidden layer neuron	128

Parametrelere değinecek olursak;

- max sequence length: Maksimum opcode sekansı
- batch size: Modelin her bir iterasyonda girdi olarak alacağı örnek sayısı
- embedding size: İşlem kodu sekanslarının vektör olarak ifade edildiği temsil uzayının boyutu
- learning rate: Modelin optimizasyon için ayarladığı öğrenme oranı
- dropout rate: Seyreltme oranı
- number of layers: Sinir ağındaki katman sayısı
- hidden layer neuron: LSTM'nin gizli katmanlarındaki nöron sayısı

LSTM modeli farklı eğitim ve test turları sonuçları Şekil13. ve Şekil 14.' te verilmiştir.



Şekil 13. LSTM modeli farklı eğitim turları çalışma sonucu

Şekil 14. LSTM modeli farklı test turları çalışma sonucu

Çıkan sonuçlar incelendiğinde LSTM modelinin oldukça az sayıda eğitim turu yapılsa dahi öğrenmeye gayet yüksek doğruluk oranları ile başladığını, ardından eğitim turu arttıkça da doğruluk oranının dramatik bir şekilde yükseldiği görülmektedir. Beklenildiği gibi 10 eğitim turu sonrasında ise artık model doygunluğa eriştiği için performansındaki gelişmeler çok sınırlı kalmıştır. 50 eğitim turu sonrası elde edilen en iyi doğruluk değeri ise %95,8 olarak elde edilmiştir. Testler sonucunda elde edilen sayısal sonuç verileri Karmaşıklık matrisine yerleştirilmiştir. Sonuçlar Tablo 11'de gösterilmiştir.

Tablo 11. Karmaşıklık matrisi sonuçları

n=378		**Gerçek Veriler**	
		Zararlı	**Zararsız**
Tahmin Edilen Veriler	**Zararlı**	362	16
	Zararsız	15	110

5. SONUÇLAR VE ÖNERİLER

Bu çalışmada farklı özellik çıkarma yöntemleriyle desteklenerek taşınabilir yürütülebilir zararlı yazılımların tespiti için LSTM hücreleri kullanılarak güçlendirilen yinelenen sinir ağları kullanılmıştır. Sonuçlar göstermiştir ki zararlı yazılım tespiti derin öğrenme algoritmaları ile yapılabilmektedir ve sonuçların geliştirilmesi için de farklı katmanlar eklenmesi faydalıdır. LSTM'nin oldukça iyi sonuçlar verdiği görülmekle beraber büyük boyutlu dosyalarda uzun işlem kodu sekanslarının oluşmasının performansa etki ettiği görülmüştür. Veri seti tümü ile çalışmaya dahil edilmesi yerine periyotlar halinde temin edilerek, çalışmanın zaman sekanslarına bölünerek yapılabileceği görülmüştür.

Tekrar eden işlem kodu sayısının fazla olması da özellik çıkarma işleminin daha da özelleştirilip geliştirilmesi gerektiğini göstermektedir. Kullanılan veri seti Microsoft tarafından 2015 yılında yayınlandığından daha yeni veri setleri oluşturup sürekli güncellenen zararlı yazılımlara karşı daha iyi bir model geliştirilebilir. Ayrıca işlem kodları programa ait davranışı genel biçimde gayet iyi bir şekilde gösterse de işlenen (operand) kısmı da bu davranışı betimlemede hassas bilgiler içerebilir. Bu yüzden ilerleyen çalışmalarda işlenen kısmı da veri çıkarma işlemine dahil edilerek daha güçlü bir model elde edilmeye çalışılacaktır.

Veri seti içerisinden LSTM modeline uygun olacak şekilde her virüs ailesinden simetrik sayıda zararlı yazılım örneği almak yerine her virüs ailesinden asimetrik sayıda zararlı yazılımın seçilerek sonuçlara etkisi araştırılabilir.

İlerleyen çalışmalarda LSTM modelinden farklı olarak Google'ın önerdiği BERT modeli kullanılarak daha farklı özellik çıkarma yöntemlerinin de kombinasyonları ile daha güçlü bir model çıkarılması hedeflenmektedir. Çıkarılmaya çalışılacak olan modelde zararlı yazılımların oluşturacağı tahribatlar da tahmin edilmeye çalışılacaktır. Ayrıca yinelenen sinir ağlarının yanı sıra ikili verilerin sınıflandırılması ya da verilerin elde edilmesi aşamasında yinelenen sinir ağları dışında farklı derin öğrenme metotları da modele dahil edilip derinleşmiş ve daha özelleşmiş bir model oluşturulması hedeflenmektedir.

KAYNAKLAR

[1]. M. Sikorski and A. Honig. Practical Malware Analysis: The Hands-On Guide to Dissecting Malicious Software. No Starch Press, 2012.

[2]. Manuel Egele, Theodoor Scholte, Engin Kirda, and Christopher Kruegel. A survey on automated dynamic malware-analysis techniques and tools. ACM computing surveys (CSUR), 44(2):6, 2012.

[3]. Skoudis, E. 2004. Malware: Fighting malicious code. Prentice Hall Professional.

[4]. Dilshan Keragala. Detecting malware and sandbox evasion techniques. SANS Insti- tute InfoSec Reading Room, 16, 2016.

[5]. Igor Santos, Jaime Devesa, Felix Brezo, Javier Nieves, and Pablo Garcia Bringas. Opem: A static-dynamic approach for machine-learning-based malware detection. In International Joint Conference CISIS'12-ICEUTE 12-SOCO 12 Special Sessions, pages 271–280. Springer, 2013.

[6]. Wen-Chieh Wu and Shih-Hao Hung. Droiddolphin: A dynamic android malware detection framework using big data and machine learning. In Proceedings of the 2014 Conference on Research in Adaptive and Convergent Systems, RACS '14, pages 247–252, New York, NY, USA, 2014. ACM.

[7]. Microsoft Technet. Accessed may 2015. Diskmon for windows v2.01. https://technet. microsoft.com/en-us/sysinternals/bb896646.

[8]. Microsoft Technet. Accessed may 2015. Process explorer v16.05. https://technet. microsoft.com/en-us/sysinternals/bb896653.

[9]. Microsoft Technet. Accessed may 2015. Tcpview v3.05. https://technet.microsoft. com/en-us/sysinternals/bb897437.

[10]. Wireshark Foundation. Accessed may 2015. Wireshark homepage. https://www. wireshark.org/.

[11]. Santos, I., Brezo, F., Ugarte-Pedrero, X., & Bringas, P. 2013. Opcode sequences as representation of executables for data-mining-based unknown malware detection. Information Sciences.

[12]. Li, W.-J., Wang, K., Stolfo, S., & Herzog, B. 2005. Fileprints: Identifying file types by n-gram analysis. Proceedings of the 2005 IEE Workshop on Assurance and Security.

[13]. Yu, S., Zhou, S., Liu, L., & Yang, R. 2010. Malware variants identification based on byte frequency. Second international conference on networks security, wireless communications and trusted computing.

[14]. Bilar, D. 2007. Opcodes as predictor for malware. Int. J. Electronic Security and Digital Forensics, Vol 1, No. 2.

[15]. Moskovitch, R., Feher, C., Tzachar, N., Berger, E., Gitelman, M., Dolev, S., & Elovici, Y. 2008. Unknown malcode detection using opcode representation. Springer-Verlag Berlin Heidelberg.

[16]. Shankarapani, M. & Ramamoorthy, S. 2010. Malware detection using assembly and api call sequences. Springer-Verlag France.

[17]. Zolotukhin, M. & Hamalainen, T. 2014. Detection of zero-day malware based on the analysis of opcode sequences. The 11th Annual IEEE SSNC - Security, Privacy and Content Protection.

[18]. Symantec. Accessed april 2015. 2014 internet security threat report, volume 19. http://www.symantec.com/content/en/us/enterprise/other_resources/ b-istr_main_report_v19_21291018.en-us.pdf.

[19]. Skoudis, E. & Zeltser, L. 2010. Malware: Fighting malicious code.

[20]. LeDoux, C. & Lakhoita, A. 2015. Malware and machine learning. Intelligent Methods for Cyber Warfare.

[21]. Dube, T., Raines, R., Peterson, B., Bauer, K., & Rogers, S. 2010. An investigation of malware type classification. Proceeding of the 5th International Conference Information Warfare and Security.

[22]. Microsoft Malware Protection Center. Accessed may 2015. Naming malware. http://www. microsoft.com/security/portal/mmpc/shared/malwarenaming.aspx.

[23]. Stallings, W. & Brown, L. 2008. Computer security principle and practise. Pearson Education.

[24]. Cohen, F. 1987. Computer viruses: theory and experiments. Computers and security.

[25]. McLaughlin, L. 2004. Bot software spreads, causes new worries. IEEE Computer Society, 5(6).

[26]. Goebel, J. & Holz, T. 2007. Rishi: Identify bot contaminated hosts by irc nickname evaluation.

[27]. Plohmann, D. & Gerhards-Padilla, E. 2012. Case study of the miner botnet. International Conference on Cyber Confl ict, 4.

[28]. McAfee. 2006. Rootkits, part 1 of 3: The growing threat. Whitepaper.

[29]. Ammar AE Elhadi, Mohd A Maarof, and Ahmed H Osman. Malware detection based on hybrid signature behaviour application programming interface call graph. American Journal of Applied Sciences, 9(3):283, 2012.

[30]. Philip OKane, Sakir Sezer, and Kieran McLaughlin. Obfuscation: The hidden mal- ware. IEEE Security & Privacy, 9(5):41–47, 2011.

[31]. Guillaume Bonfante, Matthieu Kaczmarek, and Jean-Yves Marion. Control flow graphs as malware signatures. In International workshop on the Theory of Computer Viruses, 2007.

[32]. Andreas Moser, Christopher Kruegel, and Engin Kirda. Limits of static analysis for malware detection. In Twenty-Third Annual Computer Security Applications Con- ference (ACSAC 2007), pages 421–430. IEEE, 2007.

[33]. Mihai Christodorescu and Somesh Jha. Static analysis of executables to detect mali- cious patterns. Technical report, WISCONSIN UNIV-MADISON DEPT OF COM- PUTER SCIENCES, 2006.

[34]. Mila Dalla Preda, Mihai Christodorescu, Somesh Jha, and Saumya Debray. A semantics-based approach to malware detection. ACM SIGPLAN Notices, 42(1):377– 388, 2007.

[35]. David Brumley, Cody Hartwig, Zhenkai Liang, James Newsome, Dawn Song, and Heng Yin. Automatically identifying trigger-based behavior in malware. In Botnet Detection, pages 65–88. Springer, 2008.

[36]. Jon Oberheide, Michael Bailey, and Farnam Jahanian. Polypack: an automated online packing service for optimal antivirus evasion. In Proceedings of the 3rd USENIX conference on Offensive technologies, pages 9–9. USENIX Association, 2009.

[37]. Michele Banko and Eric Brill. Scaling to very very large corpora for natural lan- guage disambiguation. In Proceedings of the 39th annual meeting on association for computational linguistics, pages 26–33. Association for Computational Linguistics, 2001.

[38]. Bojan Kolosnjaji, Apostolis Zarras, George Webster, and Claudia Eckert. Deep learn- ing for classification of malware system call sequences. In Australasian Joint Confer- ence on Artificial Intelligence, pages 137–149. Springer, 2016.

[39]. Katherine Heller, Krysta Svore, Angelos D Keromytis, and Salvatore Stolfo. One class support vector machines for detecting anomalous windows registry accesses. In ICDM Workshop on Data Mining for Computer Security, 2003.

[40]. Srilatha Attaluri, Scott McGhee, and Mark Stamp. Profile hidden markov models and metamorphic virus detection. Journal in computer virology, 5(2):151–169, 2009.

[41]. Jeremy Z Kolter and Marcus A Maloof. Learning to detect malicious executables in the wild. In Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining, pages 470–478. ACM, 2004.

[42]. Royi Ronen, Marian Radu, Corina Feuerstein, Elad Yom-Tov, and Mansour Ahmadi. Microsoft malware classification challenge. arXiv preprint arXiv:1802.10135, 2018.

[43]. Naman Bagga. Measuring the effectiveness of generic malware models. Master's thesis, San Jose State University, 2017.

[44]. Karthik Raman et al. Selecting features to classify malware. InfoSec Southwest, 2012, 2012.

[45]. Ross Quinlan. C4.5: Programs for Machine Learning. Morgan Kaufmann Publishers, San Mateo, CA, 1993.

[46]. Hyrum S Anderson and Phil Roth. Ember: an open dataset for training static pe malware machine learning models. arXiv preprint arXiv:1804.04637, 2018.

[47.] Wenyi Huang and Jack W Stokes. Mtnet: a multi-task neural network for dynamic malware classification. In International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment, pages 399–418. Springer, 2016.

[48]. Christopher Manning, Prabhakar Raghavan, and Hinrich Schu¨tze. Introduction to information retrieval. Natural Language Engineering, 16(1):100–103, 2010.

[49]. Razvan Pascanu, Jack W Stokes, Hermineh Sanossian, Mady Marinescu, and Anil Thomas. Malware classification with recurrent networks. In 2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pages 1916–1920. IEEE, 2015.

[50]. Hamid Divandari, Bassir Pechaz, and Majid Vafaie Jahan. Malware detection us- ing markov blanket based on opcode sequences. In 2015 International Congress on Technology, Communication and Knowledge (ICTCK), pages 564–569. IEEE, 2015.

[51]. Daniel Bilar. Opcodes as predictor for malware. International Journal of Electronic Security and Digital Forensics, 1(2):156–168, 2007.

[52]. Joshua Saxe and Konstantin Berlin. Deep neural network based malware detection using two dimensional binary program features. In 2015 10th International Confer- ence on Malicious and Unwanted Software (MALWARE), pages 11–20. IEEE, 2015.

[53]. Kilian Weinberger, Anirban Dasgupta, Josh Attenberg, John Langford, and Alex Smola. Feature hashing for large scale multitask learning. arXiv preprint arXiv:0902.2206, 2009.

[54]. Byron P Roe, Hai-Jun Yang, Ji Zhu, Yong Liu, Ion Stancu, and Gordon McGre- gor. Boosted decision trees as an alternative to artificial neural networks for parti- cle identification. Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment, 543(2-3):577–584, 2005.

[55]. Trevor Hastie, Saharon Rosset, Ji Zhu, and Hui Zou. Multi-class adaboost. Statistics and its Interface, 2(3):349–360, 2009.

[56]. Robert E Schapire. The boosting approach to machine learning: An overview. In Nonlinear estimation and classification, pages 149–171. Springer, 2003.

[57]. Rich Caruana and Alexandru Niculescu-Mizil. An empirical comparison of supervised learning algorithms. In Proceedings of the 23rd international conference on Machine learning, pages 161–168. ACM, 2006.

[58]. Fausett, L., Fundamentals of Neural Networks, Prentice Hall, Englewood Cliffs, NJ, 1994.

[59]. Hecht-Nielsen, R., Neurocomputing, Addison-Wesley, Reading, PA, 1990.

[60]. Rumelhart, D. E., Hinton, G. E., and Williams, R. J., Learning internal representations by error propagation, in Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Rumelhart, D. E. and McClelland, J. L., Eds., MIT Press, Cambridge, 45, 1986.

[61]. Elman, J. L., Finding structure in time, Cognitive Science, 14, 179, 1990.

[62]. Jordan, M., Generic constraints on underspecified target trajectories, Proceedings of the International Joint Conference on Neural Networks, I, 217, 1989.

[63]. Nilsson, N. J., Learning Machines: Foundations of Trainable Pattern Classifying Systems, McGraw-Hill, New York, 1965.

[64]. Mendel, J. M. and Fu, K. S., Eds., Adaptive, Learning and Pattern Recognition Systems, Academic, New York, 1970.

[65]. Werbos, P., The Roots of Backpropagation: From Ordered Derivatives to Neural Networks and Political Forecasting, Wiley, New York, 1993.

[66]. Werbos, P., Backpropagation through time: what it does and how to do it, Proceedings of the IEEE, 78, 1550, 1990.

[67]. Lapedes, A. and Farber, R., Programming a massively parallel computation universal system: static behavior, in Neural Networks for Computing, Denker, J. S., Ed., AIP Conference Proceedings, 151, 283, 1986.

[68]. Pineda, F. J., Generalization of backpropagation in recurrent neural networks, Physical Review Letters, 59 (19), 2229, 1987.

[69]. Almeida, L. B., A learning rule for asynchronous perceptrons with feedback in a combinatorial environment, Proceedings of the IEEE 1st Annual International Conference on Neural Networks, San Diego, 609, 1987.

[70]. Williams, R. and Zipser, D., A learning algorithm for continually running fully recurrent neural networks, Neural Computation, 1, 270, 1989.

[71]. Sato, M., A real time running algorithm for recurrent neural networks, Biological Cybernetics, 62, 237, 1990.

[72]. Pearlmutter, B., Learning state space trajectories in recurrent neural networks, Neural Computation, 1, 263, 1989.

[73]. Pearlmutter, B., Gradient calculations for dynamic recurrent neural networks: A survey, IEEE Transactions on Neural Networks, 6, 1212, 1995.

[74]. Randy Kath. The portable executable file format from top to bottom. MSDN Library, Microsoft Corporation, 1993.

[75]. Windows Dev Center. Pe format - windows applications, Mar 2019. https://docs. microsoft.com/en-us/windows/desktop/debug/pe-format.

[76]. [R. Ronen, M. Radu, C. Feuerstein, E. Yom-Tov, and M. Ahmadi, “Microsoft malware classification challenge,” arXiv preprint arXiv:1802.10135, 2018.].

[77]. x86asm.net. Accessed may 2015. X86 opcode and instruction reference. http://ref. x86asm.net/coder-abc.html

[78]. Potdar, Kedar & Pardawala, Taher & Pai, Chinmay. (2017). A Comparative Study of Categorical Variable Encoding Techniques for Neural Network Classifiers. International Journal of Computer Applications. 175. 7-9.

[79]. Mikolov T, Chen K, Corrado G, Dean J (2013) Efficient estimation of word representations in vector space In: International Conference on Learning Representations 2013, Scottsdale.

[80]. Lipton, 2015 Z.C. Lipton A critical review of recurrent neural networks for sequence learning

[81]. Bengio, Simard, Frasconi, 1994 Learning long-term dependencies with gradient descent is difficult IEEE Trans Neural Netw, 5 (2) (1994), pp. 157-166.

[82]. Sepp Hochreiter Neural Computation , Long-Short Term Memory, 1997 https://www.researchgate.net/publication/13853244_Long_Short-term_Memory.

[83]. Bisong E (2019) Google colaboratory In: Building Machine Learning and Deep Learning Models on Google Cloud Platform: A Comprehensive Guide for Beginners, 59–64.. Apress, Berkeley, CA..

[84]. Colaboratory https://research.google.com/colaboratory/faq.html.

[85]. Wikimedia Commons. Portable executable 32 bit structure in svg fixed, 2016. https://commons.wikimedia.org/wiki/File:Portable Executable 32 bit Structure in SVG fixed.svg.