->machine learing apt detection

ilk olarak bölüm 4 pla apt1 örnekleri ve çeşitli suç yazılımı örnekleri olan bu bölümde kullanacağımız test kötü amaçlı yazılım örneklerini tanıtıyor. ardından matematiksel benzerlik karşılaştırması ve ortak özellikleri açısından kötü amaçlı örnekerini karşılaştırmak için bir dizi teorik yöntem olan jaccard indexini kullanır. ardından tahmin için bu paylaşım miktarının değerini jaccard indeksi ile birlikte nasıl kullanabileceğinizi gösterir. ayrıca kötü amaçlı yazılım özelliklerini yararlılıkları açısından nasıl değerleneceği sonra görselleştirme

Bu veri kümeleri, bu örnekleri küratörlüğünü yapan ve araştırma topluluğunun kullanımına sunan Mandiant ve Mila Parkour sayesinde elde edildi.

İlk örnekler, paylaşılan kaynak analizini göstermek için Bölüm 4'te kullandığımız APT1 veri kümesinden gelir. Diğer örnekler, suçlular tarafından insanların kredi kartlarını çalmak, bilgisayarlarını botnet'lere bağlanmış zombi ana bilgisayarlara dönüştürmek için geliştirilen binlerce kötü amaçlı yazılım örneğinden oluşur. Bunlar, tehdit istihbaratı araştırmacıları için ücretli bir hizmet olarak sağlanan ticari bir kötü amaçlı yazılım beslemesinden elde edilen gerçek dünya örnekleridir. Aile adlarını belirlemek için her örneği Kaspersky antivirüs motoruna girdim. Kaspersky, bu örneklerden 30.104'ünü sağlam hiyerarşik sınıflandırmalarla (jorik.win32.jorik.skor.akr gibi) sınıflandırabildi, 41.830 örneğe "bilinmeyen" bir sınıf atadı ve jenerik etiketler atadı ( Örneğin, genel olarak "win32 Truva Atı") kalan 28.481 örneğe. Kaspersky etiketlerinin tutarsızlığı nedeniyle (jorik ailesi gibi bazı Kaspersky etiket gruplamaları çok yaygın bir kötü amaçlı yazılım yelpazesini temsil ederken, webprefix gibi diğerleri çok özel bir varyantlar kümesini temsil eder) ve Kaspersky'nin genellikle gözden kaçırdığı gerçeği nedeniyle veya kötü amaçlı yazılımları yanlış etiketlediğinde, Kaspersky'nin yüksek güvenle algıladığı yedi kötü amaçlı yazılım sınıfı seçtim. Bunlar özellikle dapato, pasta, skor, vbna, webprefix, xtoober ve zango ailelerini içerir.(VERİ kümesi hakkında bilgi)

Bir kötü amaçlı yazılım örneğini bir dizi özellik olarak temsil ettikten sonra, söz konusu örneğin özellik çantası ile başka bir örneğin özellik çantası arasındaki benzerlik derecesini ölçmeniz gerekir.

örneklerde jacxard indeksini kullanarak string özelliğini çıkarıp benzerlik karşılaştırıyor. bunu her kötü amaçlı yazılım çifti için yapıyor ve ortaya şekil çıkar.

sonra bunu jaccard index yerine başka bir yollarla yapmaya bakıyor.

JACCARD index string method:

#!/usr/bin/python

import argparse

import os

import networkx

from networkx.drawing.nx\_pydot import write\_dot

import itertools

def jaccard(set1, set2):

"""

Compute the Jaccard distance between two sets by taking

their intersection, union and then dividing the number

of elements in the intersection by the number of elements

in their union. İki küme arasındaki Jaccard mesafesini, kesişimlerini, birleşimlerini alarak ve ardından kesişimdeki eleman sayısını birleşimindeki elemanların sayısına bölerek hesaplayın.

"""

intersection = set1.intersection(set2)

intersection\_length = float(len(intersection))

union = set1.union(set2)

union\_length = float(len(union))

return intersection\_length / union\_length

def getstrings(fullpath):

"""

'Fullpath' parametresiyle gösterilen ikiliden dizeleri ayıklayın ve ardından ikili dosyadaki benzersiz dizeler kümesini döndürün.

"""

strings = os.popen("strings '{0}'".format(fullpath)).read()

strings = set(strings.split("\n"))

return strings

def pecheck(fullpath):

"""

Do a cursory sanity check to make sure 'fullpath' is

a Windows PE executable (PE executables start with the

two bytes 'MZ')  'Fullpath'in bir Windows PE yürütülebilir dosyası olduğundan emin olmak için üstünkörü bir mantık kontrolü yapın (PE yürütülebilir dosyaları iki bayt' MZ 'ile başlar)

"""

return open(fullpath).read(2) == "MZ"

If \_\_name\_\_ == "\_\_main\_\_":

parser = argparse.ArgumentParser(

description="Identify similarities between malware samples and build similarity graph"

)

parser.add\_argument(

"target\_directory",

help="Directory containing malware"

)

parser.add\_argument(

"output\_dot\_file",

help="Where to save the output graph DOT file"

)

parser.add\_argument(

"--jaccard\_index\_threshold", "-j", dest="threshold", type=float,

default=0.8, help="Threshold above which to create an 'edge' between samples"

)

args = parser.parse\_args()

malware\_paths = [] # where we'll store the malware file paths

malware\_features = dict() # where we'll store the malware strings

graph = networkx.Graph() # the similarity graph

for root, dirs, paths in os.walk(args.target\_directory):

# walk the target directory tree and store all of the file paths

for path in paths:

full\_path = os.path.join(root, path)

malware\_paths.append(full\_path)

# filter out any paths that aren't PE files

malware\_paths = filter(pecheck, malware\_paths)

# get and store the strings for all of the malware PE files

for path in malware\_paths:

features = getstrings(path)

print "Extracted {0} features from {1} ...".format(len(features), path)

malware\_features[path] = features

# add each malware file to the graph

graph.add\_node(path, label=os.path.split(path)[-1][:10])

# iterate through all pairs of malware

for malware1, malware2 in [itertools.com](http://itertools.com)binations(malware\_paths, 2):

# compute the jaccard distance for the current pair

jaccard\_index = jaccard(malware\_features[malware1], malware\_features[malware2])

# if the jaccard distance is above the threshold, add an edge

if jaccard\_index > args.threshold:

print malware1, malware2, jaccard\_index

graph.add\_edge(malware1, malware2, penwidth=1+(jaccard\_index-args.threshold)\*10)

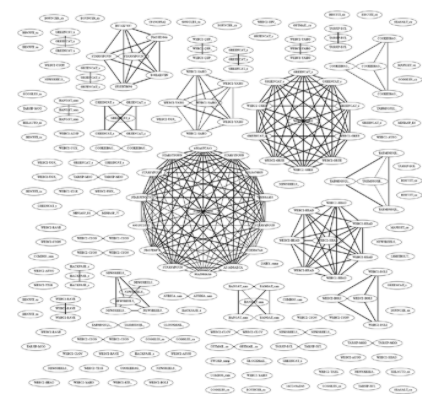
# write the graph to disk so we can visualize it

write\_dot(graph, args.output\_dot\_file)

fdp -Tpng network.dot -o network.png

bölüm 5 kodundan bulabilirsin bunun konsole olanını yaparız

bunun kodunun düzenli halini ekleyecem.



alternatif methodlara bakmak lazım

-->deep learing ile ilgili bilgiler:

\*makaleden:

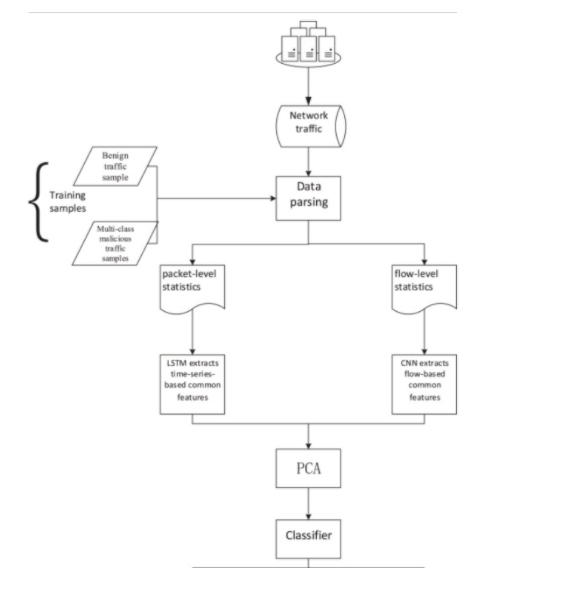
bilinmeyen C&C kanalını tespit etmek için ağ akışlarını paylaşılan özelliklerini kullanırız. yöntemimiz bilinmeyen APT saldırılarını keşfetmeye yardımcı olur.

ilk olarak, bilinen çok sınıflı saldırı akışlarından paylaşılan ağ akışı özelliklerini çıkarmak için derin öğrenme tekniklerini kullanıyoruz.  Daha sonra, C&C ağ akışını tespit etmek için uygun bir sınıflandırıcı kullanırız. son olarak, yöntemimizi halka açık veri kümesi üzerinde test ediyoruz.  Kötü amaçlı yazılım ve C&C sunucusu arasındaki ağ akışlarının paylaşılan özelliklerini yapay olarak çıkarmak zordur. burada ilk olarak derin öğrenmeyi seçiyorlar.  Bilinmeyen C&C kanallarını tespit etmek için en iyi sınıflandırıcı deneyler yoluyla seçilir. Ayrıca, zaman serisine dayalı özellikleri çıkarmak için sinir ağlarını kullanıyoruz.

bu makalenin 2. bölümünde apt sistemlerini tespit etmek için kullanlan çeşitli yöntemlerden bahsetmiş. 2.bölüm: algılama yöntemleri şu kategorilere ayrılabilir: yazılım örneklerine dayalı algılama, sistem günlüklerine göre algılama, dns kayıtlarına dayalı algılama ve ağ trafiğine göre algılama. bunları ve bunların nasıl yapıldığını ne yöntemlerin kullanıldığını anlatmış

4.bölüm: ilk önce orijinal pcap formatı verilerinden akış seviyesi istatistiklerini ve paket seviyesi istatistiklerini ayrıştırırız. Ardından, akış düzeyindeki istatistiklerden flow based shared features  akış tabanlı paylaşılan özellikleri çıkarmak için evrişimli sinir ağ(CNN) kullanırız ve paket düzeyindeki istatistiklerden zaman serisi tabanlı  paylaşılan özellikleri çıkarmak için uzun kısa süreli bellek sinir ağı (LSTM) kullanırız. son olarak; C&C kanalını tespit etmek için deneyler yoluyla uygun bir sınıflandırıcı seçilir.

deneysel çerçevemiz şekilde gösterilmektedir:  Tespit yöntemi temel olarak [veri ön işleme](https://www.sciencedirect.com/topics/computer-science/data-preprocessing) , sinir ağı ile özellik çıkarma, PCA boyutluluk indirgeme ve sınıflandırıcı dahil olmak üzere birkaç adımdan oluşur . (burada özellikleri çıkarmak için LSTM ve CNN neural tiplerini kullanarak verileri çıkarıyor. sonra çıkardığı özellikleri kullanıp sınıflandırıcı ile karar verme iştemini yapıyor deep + machine ilerliyor gibi)





4.1: veri ön işleme bölümü: veri ön işlemde ham veriyi ayrıştırmayı verilerin standardizasyonunu ve iyi huylu ve kötü niyeti örneklerin etiketlenmesini içerir.

ham veri ayrıştırma: çok fazla gereksiz veriyi içermektedir.  orijinal verileri ayrıştırmak, akış düzeyi istatistiklerini ve paket düzeyi istatistiklerini ayıklamak için scapy kullanırız.

makine öğrenmesi modellerinin eğitimini kolaylaştırmak için çıkarılan istatistiksel bilgilerin standart hale getirilmesi gerekmektedir. burada verilerin ortalamasını ve varyansını ortadan kaldırmak ve verileri standart normal dağılımı izlemesini sağlamak için (ortalama değer olacak) standart sapma standardizasyonu kullanıyoruz.

( data mining adımlarından verilerin özelliklerini deep ile çıkarmdan önce bunları yapıyor sanırım)(bunu biz verimize uygulamamıza gerek yok bizim verimizde direkt yazılımın ortak stringlerinden çıkartacağız)

4.2: özellik çıkarma: feature extraction:

ilk önce kötü amaçlı yazılım ile C&C  arasındaki ağ akışından bazı ortak özellikleri çıkarmamız gerekiyor. Bu tüm yöntemlerin en önemli özelliğidir.(bizde bunu yapacağız gidip diğer apt saldırı örneklerinde kullanıan exelerden stringleri çıkarmamız gerekiyor)

özellik çıkarmak için iki yöntem vardır: manuel özellik çıkarma ve sinir  ağı ile özellik çıkarma. manuel olanı açıklamayı geçtim.

2.Sinir ağı ile özellik çıkarma: ağın derinliği arttıkça gizli özellikleri de çıkartabilir.

çok sınıflı apt trafik verilerinden akış tabanlı özellikleri ve zaman serisi tabanlı özellikleri çıkarmak için CNN ve LSTM kullanıyoruz.  Bu özellikler, çok sınıflı APT ağ akışlarının paylaşılan özellikleridir. Bundan sonra, bilinmeyen C&C kanal tespiti için çıkarılan öznitelikler birleştirilir.  Amacımız, bilinmeyen APT saldırılarını tespit etmek için sinir ağının çok sınıflı APT saldırılarının paylaşılan özelliklerini çıkarabildiğini doğrulamaktır.

akış tabanlı özellik çıkarma: flow based feature extraction:

akış tabanlı paylaşılan özellikleri çıkarmak için bir CNN tasarladık.  Sinir ağı, veriye dayalı bir yöntemdir. CNN'ye girdiğimiz veriler, çok sınıflı kötü amaçlı ağ akışlarının akış düzeyindeki istatistikleridir. CNN'nin bu örneklerin akış düzeyi istatistiklerinden bazı akış tabanlı paylaşılan özellikleri öğrenebileceğini umuyoruz. Yapay özellik çıkarımı ile karşılaştırıldığında, ilki yalnızca birkaç ağ akışı istatistiğinin birleşik özelliklerini çıkarabilir ve [evrişim katmanı](https://www.sciencedirect.com/topics/computer-science/convolutional-layer) ve tam bağlı CNN katmanı, birden çok istatistiğin birleşik özelliklerini çıkarabilir. Ağın derinleştirilmesiyle, özellik arama alanı genişletilebilir ve bazı gizli üst düzey özellikler keşfedilebilir, böylece kötü amaçlı yazılım ve C&C sunucuları arasındaki ağ akışlarının bazı ortak özellikleri keşfedilebilir. Bu özellikleri yapay olarak bulmak zordur.

Zaman serisi tabanlı özellik çıkarma:

lstm, geliştirilmiş bir tekrarlayan sinir ağıdır.   Zaman serisi problemleriyle başa çıkmakta iyidir ve [doğal dil işlemede](https://www.sciencedirect.com/topics/engineering/natural-language-processing) büyük başarı elde etmiştir .  tekrarlayan bir sinir ağındaki bir dizinin mevcut çıkışı önceki durumla ilişkilidir, bu nedenle aynı dizideki ön ve arka girişler arasındaki korelasyonu bulmak mümkündür.  Bu bölümde, paket düzeyinde istatistiklerden zaman serisi tabanlı paylaşılan özellikleri çıkarmak için LSTM kullanıyoruz.  Bir ağ akışı, aralarında zaman serileri olan birden çok paketten oluşur. Bu nedenle, LSTM, bir ağ akışındaki paketler arasında zaman serisine dayalı özellikleri çıkarmak için kullanılabilir.

4.3:pca boyutluluk azaltma:

Verilere dahil edilebilecek veri gürültüsünü ortadan kaldırmak ve verileri uygun bir boyuta eşlemek için, sinir ağı tarafından çıkarılan özellikleri işlemek için temel bileşen analizi (PCA) kullanıyoruz.

4.4.sınıflandırıcı:

PCA, sinir ağından çıkarılan özellikleri işledikten sonra, bilinmeyen C&C kanallarını tespit etmek için en iyi sonucu veren bir sınıflandırıcı bulmayı umuyoruz. deneyler sonucu gradyan artırma karar ağacının maksimum derinlik ile en iyi sonuçları elde edebileceğini bulduk.

5.veri kümesi:

Toplam 36 APT örneği içerir.  veriyi bulmalı

6.deney:

İlk olarak, akış düzeyindeki istatistikleri ve paket düzeyindeki istatistikleri çıkarmak için ham dosyayı ayrıştırmak için scapy kullanırız.

lstn ile verilerimizi çıkartırırız boyut azaltmaya gerek yok sonra karar ağacına veririz biter

özellik çıkartıyor deep learning ile -> sonra boyutlarını azaltıyor verimli olması için -> sınıflandırıcı oluşturuyor -> sonra deneme yapıyor

\*\*sinir ağları türleri:

her farklı problem sınıfları için kullanabileceğiniz başka birçok yararlı ağ yapısı vardır.  ağ yaygın sinir ağlarına balalım:

\*feed forward neural network: ileri beslemeli sinir ağı:

en basit ve ilk tür sinir ağıdır. diğerleri bunun çeşitlerinden oluşur.  bağlantılar asla geri gitmediğinden adını böyle koymuşlar.

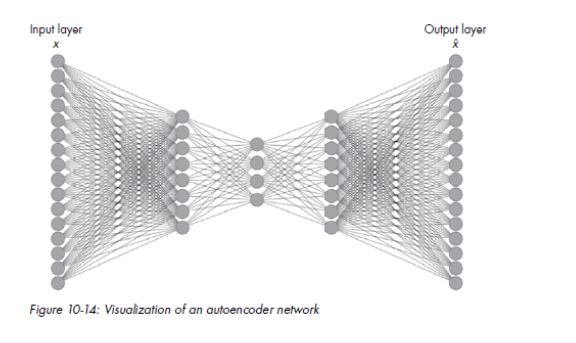
\*convolutional neural network: evrişimli sinir ağı: CNN

piksel piksel kayan bir pencere var gibi alıp bunları nörona kaydediyor.  yerel özellikleri çıkarmanın özellikle önemli olduğu alanlarda öğrenmeyi geliştirir ve hızlandırır. bu yüzden bu ağ yapısı görüntü tanıma ve sınıflandırmada son derece etkilidir.  bu nöronlar oluşturulduktan sonrada yine bir pencere kaydırma işlemi yapılır.  bunların sonucuda ağırlıksız ve maksimum aktivasyon işlevi olmayan nöronlara yeniden beslenir.

Evrişimli sinir ağları, bir veya daha fazla evrişimli ve havuz katman kümesine sahip olabilir. Standart bir mimari, bir evrişimli katman, bir havuzlama katmanı, ardından başka bir dizi evrişimsel ve havuzlama katmanları ve son olarak ileri beslemeli ağlarda olduğu gibi birkaç tam bağlantılı katman içerebilir. Bu mimarinin amacı, bu son tam bağlantılı katmanların girdi olarak oldukça yüksek seviyeli özellikler alması (tek tekerlekli bisiklet üzerinde düşünmek) ve sonuç olarak karmaşık verileri (görüntüler gibi) doğru bir şekilde sınıflandırabilmesidir.

\*Autoencoder Neural Network:Otomatik kodlayıcı Sinir Ağı

Başka bir deyişle, otomatik kodlayıcılar, optimize edilmiş kayıplı sıkıştırma programları gibi davranır, burada giriş verilerini daha küçük bir temsilde sıkıştırır ve ardından orijinal giriş boyutuna geri döndürür.



Otomatik kodlayıcı ağları, basitçe verimli sıkıştırma / açma programları olarak kullanılabilir. Örneğin, görüntü dosyalarını sıkıştırmak için eğitilen otomatik kodlayıcılar, JPEG ile aynı boyutta sıkıştırılan aynı görüntüden çok daha net görünen görüntüler oluşturabilir.

\*Generative Adversarial Network:Üretken Tartışmalı Ağ

Üretken bir düşmanlık ağı (GAN), ilgili görevlerinde kendilerini geliştirmek için birbirleriyle rekabet eden iki sinir ağından oluşan bir sistemdir. Tipik olarak, üretici ağ rastgele gürültüden sahte örnekler (örneğin, bir tür görüntü) oluşturmaya çalışır. Daha sonra ikinci bir ayırt edici ağ, gerçek örnekler ile sahte, üretilen örnekler arasındaki farkı söylemeye çalışır (örneğin, bir yatak odasının gerçek görüntüleri ile oluşturulan görüntüler arasında ayrım yapma).  GAN'lar, gerçek görünümlü veriler oluşturmak veya düşük kaliteli veya bozuk verileri geliştirmek için kullanılabilir.

\*Recurrent Neural Network:Tekrarlayan Sinir Ağı

Tekrarlayan ağlar (RRN'ler), nöronlar arasındaki bağlantıların aktivasyon işlevleri zaman adımlarına bağlı olan yönlendirilmiş döngüler oluşturduğu nispeten geniş bir sinir ağları sınıfıdır. Bu, ağın veri dizilerindeki kalıpları öğrenmesine yardımcı olan bir bellek geliştirmesine izin verir. RNN'lerde, girdiler, çıktılar veya hem girdi hem de çıktı bir çeşit zaman serisidir.

RNN'ler, bağlantılı el yazısı tanıma, konuşma tanıma, dil çevirisi ve zaman serisi analizi gibi veri sırasının önemli olduğu görevler için mükemmeldir. Siber güvenlik bağlamında, ağ trafiği analizi, davranışsal algılama ve statik dosya analizi gibi sorunlarla ilgilidir. Program kodu, bu sırayla doğal dile benzer olduğu için, bir zaman serisi olarak ele alınabilir.

RNN'lerle ilgili bir sorun, kaybolan gradyan probleminden dolayı, bir RNN'ye eklenen her zaman adımının, ileri beslemeli bir sinir ağındaki tüm ekstra katmana benzer olmasıdır. Geri yayılım sırasında, kaybolan gradyan problemi, daha düşük seviyeli katmanlardaki (veya bu durumda, daha önceki zaman adımlarındaki) sinyallerin inanılmaz derecede zayıf olmasına neden olur.

Uzun kısa süreli bellek  long short-term memory (LSTM) ağı, bu sorunu çözmek için tasarlanmış özel bir RNN türüdür. LSTM'ler, hangi bilgilerin hatırlanacağına ve hangi bilgilerin unutulacağına karar vermeye çalışan bellek hücreleri ve özel nöronlar içerir. Çoğu bilginin atılması, kaybolan gradyan sorununu büyük ölçüde sınırlar çünkü yol patlamasını azaltır.

\*ResNet:

ResNet (artık ağın kısaltması), ağın erken / sığ katmanlarındaki nöronlar arasında, bir veya daha fazla ara katmanı atlayarak daha derin katmanlara atlama bağlantıları oluşturan bir tür sinir ağıdır

-->deep learning html detection: building a neaural network malware detector with keras:

keras  deep learning için  sinir ağı oluşturmak için. Python paketi sklearn'ı kullanarak, modelin doğrulama verileri üzerindeki doğruluğunu nasıl değerlendireceğinizi öğrenirsiniz.  kerasta  bir modeln mimarisini nasıl tanımlayacağını sonnra bu modeli iyi huylu ve kötü niyetli html dosyalarını ayırt etmek için eğitiyoruz ve bu tür modelleri nasıl nasıl kaydediğ yükleyeceğinizi. en sonda python sklearn ı kullanarak modelin doğrulama verileri üzerindeki doğruluğunu nasıl değerlendireceğinizi öğrenirsiniz. son olarak doğrulama doğruluğu raporlamasını model eğtim sürecine entegre etmek için öğrendiklerinizi kullanırız.

\*bir modelin mimarisini tanımlama:

ilk olarak bir sinir ağı oluşturmak için mimarisini tanımlamanız gerekir.  Keras tüm bunları tanımlamak için basit bir arayüz sağlar.

basit bir model tanımalamak için:

(1) from keras import layers

(2) from keras.models import Model

#3 de katmanlara bir şekil değeri 4 de ve bir veri türü ileterek bu modelin bir gözlem için ne tür verileri kabul edeceğini belirtiyoruz.

input = layers.Input((3)shape=(1024,), (4)dtype='float32')

(5) middle = layers.Dense(units=512, activation='relu')(input)

(6) output = layers.Dense(units=1, activation='sigmoid')(middle)

(7) model = Model(inputs=input, outputs=output)

[model.com](http://model.com)pile((8)optimizer='adam',

(9)loss='binary\_crossentropy',

(10)metrics=['accuracy'])

yoğun keras modellerini geliştirirken kullanacağınız en yaygın katman türüdür.

Diğerleri, verilerin şeklini değiştirme (Yeniden Şekillendirme Reshape) ve kendi özel katmanınızı (Lambda) uygulama gibi şeyler yapmanıza izin verir. Yoğun fonksiyonuna iki argüman aktarıyoruz: bu katmanda 512 nöron istediğimizi belirtmek için birim = 512 ve bu nöronların düzeltilmiş doğrusal birim (ReLU) nöronlar olmasını istediğimizi belirtmek için aktivasyon = 'relu'.

Sonraki satırda, yine Yoğun işlevini Dense function kullanan modelimizin çıktı katmanını tanımlıyoruz. Ancak bu sefer katmana yalnızca tek bir nöron atıyoruz ve bir 'sigmoid' aktivasyon işlevi (6) kullanıyoruz; bu, çok sayıda veriyi 0 ile 1 arasındaki tek bir skorda birleştirmek için harikadır. Çıkış katmanı, (ortadaki ) girdi olarak nesne, orta katmanımızda ki 512 nöronumuzun çıktılarının hepsinin bu nörona gönderilmesi gerektiğini bildirir.

en sonda modeli oluşturduklaırmızı vererek compile ederiz

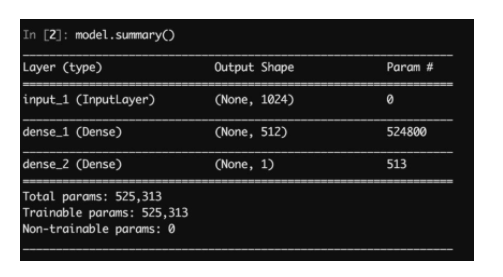
\*Modeli Derlemek Compiling:

eğitimini nasıl gerçekleştirmesini istediğimizi belirtmek için compile method kullanacağız bu da üç parametre alıyor.  ilk parametre olan optimizer,  kullanılacak geri yayılım algoritmasının türünü belirtir. Kullanmak istediğiniz algoritmanın adını burada yaptığımız gibi bir karakter dizisi aracılığıyla belirleyebilir veya doğrudan keras.optimizers dan bir algoritmayı içe aktararak belirli parametreleri algoritmaya aktarabilir veya hatta kendi algoritmanızı tasarlayabilirsiniz.

-Kayıp parametresi (9), eğitim süreci (geri yayılım) sırasında en aza indirilen şeyi belirtir. Özellikle bu, gerçek eğitim etiketleriniz ile modelinizin öngörülen etiketleri (çıktı) arasındaki farkı temsil etmek için kullanmak istediğiniz formülü belirtir. Yine, bir kayıp işlevinin adını belirtebilir veya keras.losses.mean\_squared\_error gibi gerçek bir işlevi iletebilirsiniz.

-Son olarak, metrik parametresi (10) için, eğitim sırasında ve sonrasında model performansını analiz ederken Keras'ın raporlamasını istediğiniz metriklerin bir listesini iletebilirsiniz. Yine, ['categorical\_accuracy', keras.metrics.top\_k\_categorical\_accur cy] gibi dizeleri veya gerçek metrik işlevleri iletebilirsiniz.

ekranınıza yazdırılan model yapısını görmek için model.summary () öğesini çalıştırın.



Her katmanın açıklaması, o katmanla ilişkili parametrelerin sayısıyla birlikte ekrana yazdırılır. Modelimizi henüz eğitmemiş veya doğrulama verileri üzerinde test etmemiş olsak da bu, eğitilmeye hazır derlenmiş bir Keras modelidir!

\*Modeli Eğitmek:

Modelimizi eğitmek için eğitim verilerine ihtiyacımız var. html dosyaları var iyi huylu ve kötü huylu bunları kullanacak.

->Extrecting Features: Unsurları Çıkarma:

özellik çıkarma kodu:

import numpy as np

import murmur

import re

import os

def read\_file(sha, dir):

with open(os.path.join(dir, sha), 'r') as fp:

file = fp.read()

return file

def extract\_features(sha, path\_to\_files\_dir,

hash\_dim=1024, (1)split\_regex=r"\s+"):

(2) file = read\_file(sha=sha, dir=path\_to\_files\_dir)

(3) tokens = re.split(pattern=split\_regex, string=file)

# now take the modulo(hash of each token) so that each token is replaced

# by bucket (category) from 1:hash\_dim.

token\_hash\_buckets = [

(4) (murmur.string\_hash(w) % (hash\_dim - 1) + 1) for w in tokens

]

# Finally, we'll count how many hits each bucket got, so that our features

# always have length hash\_dim, regardless of the size of the HTML file:

token\_bucket\_counts = np.zeros(hash\_dim)

# this returns the frequency counts for each unique value in

# token\_hash\_buckets:

buckets, counts = np.unique(token\_hash\_buckets, return\_counts=True)

# and now we insert these counts into our token\_bucket\_counts object:

for bucket, count in zip(buckets, counts):

(5) token\_bucket\_counts[bucket] = count

return np.array(token\_bucket\_counts)

Extract\_features işlevi, bir HTML dosyasını büyük bir dizge (2) olarak okuyarak başlar ve daha sonra bu dizeyi normal ifadeye (3) dayalı olarak bir dizi jetona böler. Daha sonra, her bir jetonun sayısal karması alınır ve bu karmalar, her bir karmanın (4) modulosu alınarak kategorilere ayrılır. Son özellik seti, histogram bin sayısı gibi, her kategorideki (5) karma sayısıdır. İsterseniz, ortaya çıkan belirteçleri ve özellikleri nasıl etkilediğini görmek için HTML dosyasını parçalara ayıran split\_regex (1) normal ifadesini değiştirmeyi deneyebilirsiniz.

Bunları atladıysanız veya anlamadıysanız, sorun değil: extract\_features fonksiyonumuzun girdi olarak bir HTML dosyasına giden yolu aldığını ve ardından 1.024 uzunluğunda bir özellik dizisine veya hash\_dim ne olursa olsun dönüştürdüğünü bilin.

alıyor 1024 hashliyor

\*Creating a Data Generator: Veri Oluşturucu oluşturma:

keras modelimizin bu özellikler üzerinde çalışmasına ihtiyaç var.  basit verilerle çalışırken

# first you would load in my\_data and my\_labels via some means, and then:

model.fit(my\_data, my\_labels, epochs=10, batch\_size=32)

Bunu aşmak için biraz daha karmaşık ancak daha ölçeklenebilir model.fit\_generator işlevini kullanıyoruz. Tüm eğitim verilerini bir kerede bu işleve aktarmak yerine, eğitim verilerini gruplar halinde veren bir oluşturucu iletirsiniz, böylece bilgisayarınızın RAM'i tıkanmaz. 11-4 listesi, özellik çıkarma işlevimizi kullanarak kendi veri oluşturucumuzu nasıl oluşturabileceğimizi gösterir.

veri oluşturucu oluşturmak için:

def my\_generator(benign\_files, malicious\_files,

path\_to\_benign\_files, path\_to\_malicious\_files,

batch\_size, features\_length=1024):

n\_samples\_per\_class = batch\_size / 2

(1) assert len(benign\_files) >= n\_samples\_per\_class

assert len(malicious\_files) >= n\_samples\_per\_class

(2) while True:

ben\_features = [

extract\_features(sha, path\_to\_files\_dir=path\_to\_benign\_files,

hash\_dim=features\_length)

for sha in np.random.choice(benign\_files, n\_samples\_per\_class,

replace=False)

]

mal\_features = [

(3) extract\_features(sha, path\_to\_files\_dir=path\_to\_malicious\_files,

hash\_dim=features\_length)

(4)for sha in np.random.choice(malicious\_files, n\_samples\_per\_class,

replace=False)

]

(5) all\_features = ben\_features + mal\_features

labels = [0 for i in range(n\_samples\_per\_class)] + [1 for i in range(

n\_samples\_per\_class)]

idx = np.random.choice(range(batch\_size), batch\_size)

(6) all\_features = np.array([np.array(all\_features[i]) for i in idx])

labels = np.array([labels[i] for i in idx])

(7) yield all\_features, labels

11-5 listesi, bu kitapla birlikte gelen verileri kullanarak bir eğitim verisi oluşturucunun nasıl oluşturulacağını ve oluşturucuyu modelimizin fit\_generator yöntemine geçirerek modelimizi nasıl eğiteceğimizi gösterir.

import os

batch\_size = 128

features\_length = 1024

path\_to\_training\_benign\_files = 'data/html/benign\_files/training/'

path\_to\_training\_malicious\_files = 'data/html/malicious\_files/training/'

steps\_per\_epoch = 1000 # artificially small for example-code speed!

(1)train\_benign\_files = os.listdir(path\_to\_training\_benign\_files)

(2) train\_malicious\_files = os.listdir(path\_to\_training\_malicious\_files)

# make our training data generator!

(3)training\_generator = my\_generator(

benign\_files=train\_benign\_files,

malicious\_files=train\_malicious\_files,

path\_to\_benign\_files=path\_to\_training\_benign\_files,

path\_to\_malicious\_files=path\_to\_training\_malicious\_files,

batch\_size=batch\_size,

features\_length=features\_length

)

(4) model.fit\_generator(

(5)generator=training\_generator,

(6)steps\_per\_epoch=steps\_per\_epoch,

(7)epochs=10

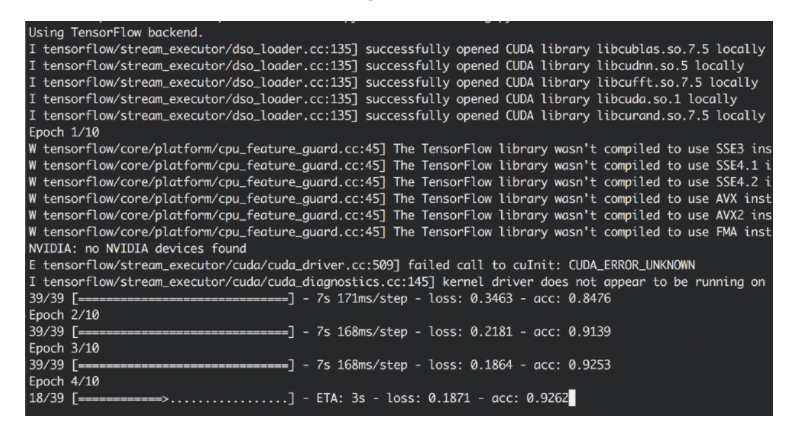
)

Liste 11-5: Eğitim oluşturucuyu oluşturma ve modeli eğitmek için kullanma

Gerekli bir paketi içe aktardıktan ve bazı parametre değişkenlerini oluşturduktan sonra, zararsız (1) ve kötü niyetli eğitim verilerimizin (2) dosya adlarını belleğe okuruz (ancak dosyaların kendileri değil). Eğitim veri oluşturucumuzu almak için bu değerleri yeni my\_generator işlevimize (3) aktarıyoruz. Son olarak, Liste 11-1'deki modelimizi kullanarak, eğitime başlamak için modelin yerleşik fit\_generator yöntemini (4) kullanıyoruz.

Fit\_generator yöntemi üç parametre alır. Oluşturucu parametresi (5), her parti için eğitim verisi üreten veri oluşturucuyu belirtir. Eğitim sırasında parametreler, o grup için tüm eğitim gözlemlerinin sinyallerinin ortalaması alınarak grup başına bir kez güncellenir. Steps\_per\_epoch parametresi (6), modelin her dönemi işlemesini istediğimiz parti sayısını ayarlar. Sonuç olarak, modelin dönem başına gördüğü toplam gözlem sayısı batch\_size \* steps\_per\_epoch şeklindedir. Kural olarak, bir modelin dönem başına gördüğü gözlem sayısı, veri kümesi boyutuna eşit olmalıdır, ancak bu bölümde ve sanal makine örnek kodunda, kodumuzun daha hızlı çalışmasını sağlamak için steps\_per\_epoch'u azaltıyorum. Epochs parametresi (7) çalıştırmak istediğimiz epoch sayısını belirler.

Bu kodu, bu kitaba eşlik eden ch11 / dizininde çalıştırmayı deneyin.Süreci iptal ettikten sonra (veya kod tamamlandıktan) sonra eğitimli bir modeliniz olacak



Şekil 11-2: Bir Keras modelinin eğitiminden elde edilen konsol çıktısı

(GPU'lar sinir ağlarını eğitmek için genellikle yaklaşık 2–20 kat daha hızlıdır, ancak bu kitabın amaçları doğrultusunda CPU tabanlı eğitim iyidir)

Son olarak, her dönem için, belirli bir dönemin ne kadar daha uzun süreceğini gösteren bir ilerleme çubuğu ile dönemin kayıp ve doğruluk ölçümlerini göreceksiniz.

\*Incorporating Validation Data:Doğrulama Verilerini Dahil Etme

Önceki bölümde, ölçeklenebilir fit\_generator yöntemini kullanarak HTML dosyalarında bir Keras modelinin nasıl eğitileceğini öğrendiniz. ikincisi Şekil 11-2'de gösterilmiştir. Daha da iyisi, bir üretim ortamını daha iyi simüle etmek için eğitim verilerinden sonraki tarihlerden gelen doğrulama dosyalarını kullanmaktır.

Liste 11-6, Liste 11-4'teki my\_generator işlevimizi kullanarak doğrulama özelliklerimizi belleğe nasıl yükleyeceğimizi gösterir.

import os

path\_to\_validation\_benign\_files = 'data/html/benign\_files/validation/'

path\_to\_validation\_malicious\_files = 'data/html/malicious\_files/validation/'

# get the validation keys:

val\_benign\_file\_keys = os.listdir(path\_to\_validation\_benign\_files)

val\_malicious\_file\_keys = os.listdir(path\_to\_validation\_malicious\_files)

# grab the validation data and extract the features:

(1)validation\_data = my\_generator(

benign\_files=val\_benign\_files,

malicious\_files=val\_malicious\_files,

path\_to\_benign\_files=path\_to\_validation\_benign\_files,

path\_to\_malicious\_files=path\_to\_validation\_malicious\_files,

(2) batch\_size=10000,

features\_length=features\_length

(3) ).next()

Listing 11-6: Reading validation features and labels into memory by using the

Dolayısıyla, sadece oluşturucuyu oluşturmak yerine, üzerinde doğrulamak istediğimiz dosya sayısına eşit büyük bir batch\_size (2) ile bir doğrulama veri oluşturucusu (1) oluşturuyoruz ve hemen next () (3) yöntemini sadece bir kez çağırıyoruz. .

, Keras, Liste 11-7'de gösterildiği gibi eğitim sırasında doğrulama verilerimizi basitçe fit\_generator () geçirmemize izin veriyor.

model.fit\_generator(

(1) validation\_data=validation\_data,

generator=training\_generator,

steps\_per\_epoch=steps\_per\_epoch,

epochs=10

)

Listing 11-7: Using validation data for automatic monitoring during training:Liste 11-7: Eğitim sırasında otomatik izleme için doğrulama verilerini kullanma

\*Saving and Loading the Model:Modeli Kaydetme ve Yükleme

h5 dosyasına kaydetme ve sonra tekrar nasıl yeniden yükleyeceğimizi gösterir.

from keras.models import load\_model

# save the model

(1) model.save('my\_model.h5')

# load the model back into memory from the file:

(2)same\_model = load\_model('my\_model.h5')

\*Evaluating the Model:Modeli Değerlendirme

BU kısma daha gelmedim ileride bakmalı

Model eğitimi bölümünde, eğitim kaybı ve doğruluğu gibi bazı varsayılan model değerlendirme ölçütlerinin yanı sıra doğrulama kaybı ve doğruluğu gözlemledik. Modellerimizi daha iyi değerlendirmek için şimdi daha karmaşık metrikleri gözden geçirelim

İkili bir öngörücünün doğruluğunu değerlendirmek için kullanışlı bir metriğe eğri altındaki alan (AUC) denir. Eğri, tüm olası puan eşikleri için gerçek pozitif oranlara (y ekseni) karşı yanlış pozitif oranları (x ekseni) gösteren bir Alıcı Çalışma Karakteristiği (ROC) eğrisine (bkz. Bölüm 8) karşılık gelir.

Basit bir ifadeyle, 0,5'lik bir AUC, yazı tura atmanın tahmin kabiliyetini temsil ederken, 1'lik bir AUC mükemmeldir.

Şekil 11-4: Çeşitli örnek ROC eğrileri. Her bir ROC eğrisi (çizgi) farklı bir AUC değerine karşılık gelir.

Liste 11-9'daki kodu kullanarak doğrulama AUC'yi hesaplamak için doğrulama verilerimizi kullanalım.

from sklearn import metrics

(1) validation\_labels = validation\_data[1]

(2) validation\_scores = [el[0] for el in model.predict(validation\_data[0])]

(3) fpr, tpr, thres = metrics.roc\_curve(y\_true=validation\_labels,

y\_score=validation\_scores)

(4) auc = metrics.auc(fpr, tpr)

print('Validation AUC = {}'.format(auc))

Listing 11-9: Calculating validation AUC using sklearn’s metric submodule:Liste 11-9: sklearn’in metrik alt modülünü kullanarak doğrulama AUC'yi hesaplama

Burada, validation\_data demetimizi iki nesneye ayırıyoruz: validation\_labels (1) ile temsil edilen doğrulama etiketleri ve validation\_scores (2) ile temsil edilen düzleştirilmiş doğrulama modeli tahminleri. Ardından, model tahminleri için yanlış pozitif oranları, gerçek pozitif oranları ve ilişkili eşik değerlerini hesaplamak için sklearn'daki metrics.roc\_curve işlevini kullanırız (3). Bunları kullanarak, yine bir sklearn fonksiyonu (4) kullanarak AUC metriğimizi hesaplıyoruz.

gerçek ROC eğrisini Liste 11-10'da gösterildiği gibi bu kitapla birlikte verilen verilerdeki ch11 / model\_evaluation.py dosyasında bulunan roc\_plot () işlevini de kullanabilirsiniz. .

from ch11.model\_evaluation import roc\_plot

roc\_plot(fpr=fpr, tpr=tpr, path\_to\_file='roc\_curve.png')

Listing 11-10: Creating a ROC curve plot using the roc\_plot function from this book’s

accompanying data, in ch11/model\_evaluation.py:Liste 11-10: Bu kitabın eşlik eden verilerinden roc\_plot işlevini kullanarak, ch11 / model\_evaluation.py içinde bir ROC eğri grafiği oluşturma

Kodun Liste 11-10'da çalıştırılması, Şekil 11-5'e benzeyen bir çizim (roc\_curve.png'ye kaydedilmiş) oluşturmalıdır.

ekil 11-5'teki ROC eğrisindeki her nokta, 0 ile 1 arasında değişen çeşitli model tahmin eşikleri ile ilişkili spesifik bir yanlış pozitif oranı (x ekseni) ve gerçek pozitif oranı (y ekseni) temsil eder. Yanlış pozitif oranlar arttıkça, doğru -pozitif oranlar artar ve bunun tersi de geçerlidir.

\*Enhancing the Model Training Process with Callbacks:Geri Çağırma ile Model Eğitim Sürecini Geliştirme

Şimdiye kadar, Keras modellerini nasıl tasarlayacağınızı, eğiteceğinizi, kaydedeceğinizi, yükleyeceğinizi ve değerlendireceğinizi öğrendiniz. model eğitim sürecimizi daha da iyi hale getirebilecek Keras geri aramalarını da tanıtmak istiyorum.

Keras geri araması, eğitim sürecinin belirli aşamalarında Keras'ın uyguladığı bir dizi işlevi temsil eder. Örneğin, her dönemin sonunda bir h5 dosyasının kaydedildiğinden veya doğrulama AUC'nin her dönemin sonunda ekrana yazdırıldığından emin olmak için bir Keras geri araması kullanabilirsiniz. Bu, eğitim sürecinde modelinizin nasıl gittiğini daha kesin bir şekilde kaydetmenize ve size bilgi vermenize yardımcı olabilir.

Yerleşik bir geri arama kullanarak başlıyoruz ve ardından kendi özel geri aramamızı yazmaya çalışıyoruz

\*Using a Built-in Callback:Yerleşik Geri Aramayı Kullanma

Yerleşik bir geri arama kullanmak için, eğitim sırasında modelinizin fit\_generator () yöntemine bir geri arama örneğini iletmeniz yeterlidir. Geri aramaları kullanacağız. Her eğitim döneminden sonra doğrulama kaybını değerlendiren ve doğrulama kaybı önceki dönemin doğrulama kayıplarından daha küçükse mevcut modeli bir dosyaya kaydeden ModelCheckpoint geri çağrısı. Bunu yapmak için, geri aramanın doğrulama verilerimize erişmesi gerekir, bu nedenle onu Liste 11-11'de gösterildiği gibi fit\_generator () yöntemine aktaracağız.

from keras import callbacks

model.fit\_generator(

generator=training\_generator,

# lowering steps\_per\_epoch so the example code runs fast:

steps\_per\_epoch=50,

epochs=5,

validation\_data=validation\_data,

callbacks=[

callbacks.ModelCheckpoint(save\_best\_only=True,(1)

(2) filepath='results/best\_model.h5',

(3)monitor='val\_loss')

],

)

Listing 11-11: Adding a ModelCheckpoint callback to the training process:Liste 11-11: Eğitim sürecine ModelCheckpoint geri araması ekleme

Bu kod, 'val\_loss' (3) (doğrulama kaybı) yeni bir düşük seviyeye ulaştığında, modelin (1) tek bir dosyaya 'results / best\_model.h5' (2) üzerine yazılmasını sağlar. Bu, geçerli kaydedilen modelin ('results / best\_model .h5') her zaman doğrulama kaybıyla ilgili olarak tamamlanan tüm çağlarda en iyi modeli temsil etmesini sağlar.

Alternatif olarak, modeli her dönemden sonra doğrulama kaybından bağımsız olarak ayrı bir dosyaya kaydetmek için Liste 11-12'deki kodu kullanabiliriz.

callbacks.ModelCheckpoint(save\_best\_only=False,(4)

                                                        (5) filepath='results/model\_epoch\_{epoch}.h5',

monitor='val\_loss')

Listing 11-12: Adding a ModelCheckpoint callback to the training process that saves the

model to a different file after each epoch:Liste 11-12: Eğitim sürecine her dönemden sonra modeli farklı bir dosyaya kaydeden bir ModelCheckpoint geri araması ekleme

Bunu yapmak için, Liste 11-11'de aynı kodu ve aynı ModelCheckpoint işlevini kullanıyoruz, ancak save\_best\_only = False (4) ve Keras'tan epoch numarasını (5) doldurmasını isteyen bir dosya yolu kullanıyoruz. Modelimizin yalnızca "en iyi" sürümünü kaydetmek yerine, Liste 11-12’nin geri araması, her çağın modelimizin sürümünü sonuç / model\_epoch\_0.h5, sonuçlar / model\_epoch\_1.h5, sonuçlar / model\_epoch\_2.h5 vb. Olarak kaydeder.

\*Using a Custom Callback:Özel Geri Aramayı Kullanma

Keras, AUC'yi desteklemese de, örneğin, her dönemden sonra ekrana AUC yazdırmamıza izin vermek için kendi özel geri aramamızı tasarlayabiliriz.

Özel bir Keras geri araması oluşturmak için, yeni geri aramalar oluşturmak için kullanılan soyut temel sınıf olan keras.callbacks.Callback'ten miras alan bir sınıf oluşturmamız gerekir. Adlarının belirttiği zamanlarda, eğitim sırasında otomatik olarak çalıştırılacak bir veya daha fazla yöntem seçimi ekleyebiliriz: on\_epoch\_begin, on\_epoch\_end, on\_batch\_begin, on\_batch\_end, on\_train\_begin ve on\_train\_end. Liste 11-13, doğrulama AUC'sini hesaplayan ve her dönemin sonunda ekrana yazdıran bir geri aramanın nasıl oluşturulacağını gösterir.

import numpy as np

from keras import callbacks

from sklearn import metrics

(1) class MyCallback(callbacks.Callback):

(2)def on\_epoch\_end(self, epoch, logs={}):

(3) validation\_labels = self.validation\_data[1]

validation\_scores = self.model.predict(self.validation\_data[0])

# flatten the scores:

validation\_scores = [el[0] for el in validation\_scores]

fpr, tpr, thres = metrics.roc\_curve(y\_true=validation\_labels,

            y\_score=validation\_scores)

(4) auc = metrics.auc(fpr, tpr)

print('\n\tEpoch {}, Validation AUC = {}'.format(epoch,

                                        np.round(auc, 6)))

model.fit\_generator(

generator=training\_generator,

# lowering steps\_per\_epoch so the example code runs fast:

steps\_per\_epoch=50,

epochs=5,

(5) validation\_data=validation\_data,

(6)callbacks=[

callbacks.ModelCheckpoint('results/model\_epoch\_{epoch}.h5',

            monitor='val\_loss',

            save\_best\_only=False,

            save\_weights\_only=False)

]

)

Listing 11-13: Creating and using a custom callback to print AUC to the screen after each

training epoch:Liste 11-13: Her eğitim döneminden sonra AUC'yi ekrana yazdırmak için özel bir geri arama oluşturma ve kullanma

Bu örnekte, önce geri aramalardan miras alan MyCallback sınıfımızı (1) oluşturuyoruz. İşleri basit tutarak, tek bir yöntemin üzerine yazıyoruz, on\_epoch\_end (2) ve ona Keras tarafından beklenen iki argüman veriyoruz: epoch ve logs (bir günlük bilgileri sözlüğü), her ikisi de Keras eğitim sırasında fonksiyonu çağırdığında sağlayacaktır.

  Ardından, geri çağırmalar sayesinde öz nesnede zaten depolanan validation\_data (3) 'yı alırız. Geri arama mirası ve "Modeli Değerlendirme" sayfa 209'da yaptığımız gibi AUC'yi (4) hesaplar ve yazdırırız. bu kodun çalışması için, geri aramanın eğitim sırasında self.validation\_data erişimine sahip olması için doğrulama verilerinin fit\_generator () 'a geçirilmesi gerekir (5). Son olarak, modele yeni geri aramamızı eğitmesini ve belirtmesini söyleriz (6). Sonuç Şekil 11-6 gibi görünmelidir.

Figure 11-6: Console output from training a Keras model with a custom AUC callback:Şekil 11-6: Özel bir AUC geri aramasıyla bir Keras modelinin eğitiminden elde edilen konsol çıktısı

Gerçekten önemsediğiniz şey doğrulama AUC'sini en aza indirmekse, bu geri arama, modelinizin eğitim sırasında nasıl çalıştığını görmenizi kolaylaştırarak eğitim sürecini durdurmanız gerekip gerekmediğini değerlendirmenize yardımcı olur (örneğin, doğrulama doğruluğu zaman içinde sürekli olarak azalırsa) ).

->ÖZET:

Bu bölümde, Keras kullanarak kendi sinir ağınızı nasıl kuracağınızı öğrendiniz. Ayrıca onu eğitmeyi, değerlendirmeyi, kaydetmeyi ve yüklemeyi de öğrendiniz. Daha sonra, yerleşik ve özel geri aramalar ekleyerek model eğitim sürecini nasıl geliştireceğinizi öğrendiniz. Model mimarisi ve özellik çıkarımının model doğruluğu üzerinde ne gibi değişikliklere sahip olabileceğini görmek için bu kitapla birlikte verilen kodla oynamanızı tavsiye ederim.

Bu bölüm ayaklarınızı ıslatmak içindir, ancak bir başvuru kılavuzu olarak tasarlanmamıştır. En güncel resmi belgeler için <https://keras.io> adresini ziyaret edin. Keras'ın ilginizi çeken yönlerini araştırmak için zaman ayırmanızı şiddetle tavsiye ediyorum. Umarım bu bölüm, tüm güvenlik derin öğrenme maceralarınız için iyi bir başlangıç noktası olarak hizmet etmiştir!

extract eğitim kısmını değiştirmem lazım

-->bu ikisinin karması

kodu yaz anla sonra buna uyarlamaya çalış