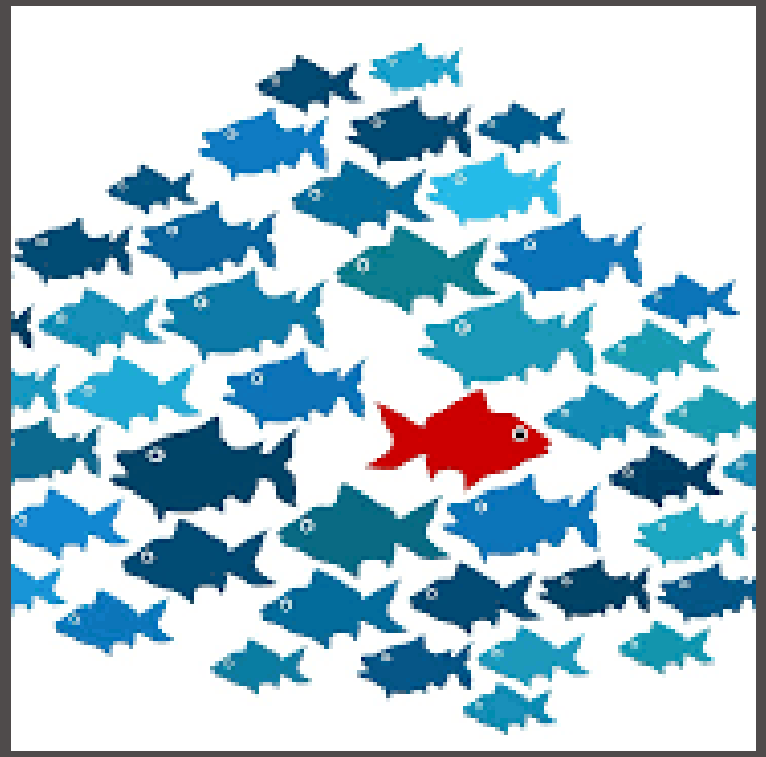
PYTORCH İLE ANOMALİ TESPİTİ

OSMAN TAHİR KUZU

19360859030

- Anomali Nedir?
- Anomali Tespiti Nedir?
  - Kategorileri
  - Popüler Teknikleri
  - Zaman serisi veri anomali tespiti
    - Küresel Aykırı veri (Outliers)
    - Bağlamsal Aykırı veri
    - Kollektif Aykırı veri
  - Kullanım Alanları
  - Kısa Hikaye
- Pytorch
  - Nedir?
  - Neden PyTorch?
  - Tensorflow ile farkı
  - Model Oluşturma



ANOMALİ (ANOMALY/OUTLİER)

- **Anomaliler**, veri setinde çok nadiren meydana gelen ve özellikleri verilerin çoğundan önemli ölçüde farklı olan veri örnekleri veya koleksiyonlarıdır.
- **Anomaliler** ayrıca aykırı değerler, yenilikler, gürültü, sapmalar ve istisnalar olarak da adlandırılmaktadır

Anomali tespiti (aykırı değer analizi olarak da bilinir), bir veri kümesinin normal davranışından sapan veri noktalarını, olayları ve/veya gözlemleri tanımlayan veri madenciliğinde bir adımdır.

Anormal veriler, teknik bir aksaklık gibi kritik olayları veya örneğin tüketici davranışındaki bir değişiklik gibi potansiyel fırsatları gösterebilir. Makine öğrenimi, anormallik algılamayı otomatikleştirmek için aşamalı olarak kullanılmaktadır.

İlk olarak, örneğin ortalama veya standart sapmayı hesaplamak için istatistiksel analize yardımcı olmak için verilerden açık bir şekilde reddedilme veya çıkarılma için anormallikler arandı.

ÜÇ GENİŞ ANOMALİ TESPİT TEKNİĞİ KATEGORİSİ BULUNUR:

- **Denetimli anomali** tespit teknikleri, "normal" ve "anormal" olarak etiketlenmiş bir veri seti gerektirir ve bir sınıflandırıcının eğitimini içerir. Bununla birlikte, bu yaklaşım, etiketli verilerin genel olarak mevcut olmaması ve sınıfların doğal dengesiz doğası nedeniyle, anormallik tespitinde nadiren kullanılır.

- **Yarı denetimli anomali** tespit teknikleri, verilerin bir kısmının etiketlendiğini varsayar. Bu, normal veya anormal verilerin herhangi bir kombinasyonu olabilir, ancak çoğu zaman teknikler, belirli bir normal eğitim veri setinden normal davranışı temsil eden bir model oluşturur ve ardından model tarafından oluşturulacak bir test örneğinin olasılığını test eder.

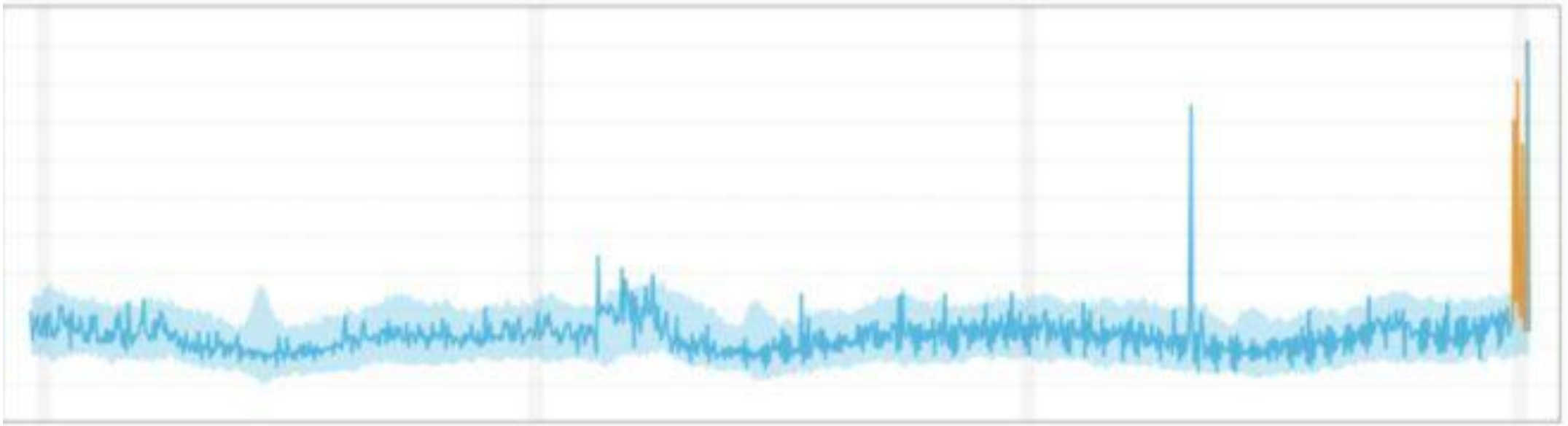
- **Denetimsiz anormallik** tespit teknikleri, verilerin etiketsiz olduğunu varsayar ve daha geniş ve ilgili uygulamaları nedeniyle açık ara en yaygın olarak kullanılanlardır.

POPÜLER TEKNİKLER

- Statistical (Z-score, Tukey's range test and Grubbs's test)
- Density-based techniques (k-nearest neighbor,local outlier factor,isolation forests, and many more variations of this concept
- One-class support vector machines
- Replicator neural networks, autoencoders, variational autoencoders, long short-term memory neural networks
- Bayesian networks
- Hidden Markov models (HMMs)
- Minimum Covariance Determinant
- Clustering: Cluster analysis-based outlier detection
- Fuzzy logic-based outlier detection
- Ensemble techniques, using feature bagging, score normalization and different sources of diversity
- … vb
- **NOT:** Yöntemlerin performansı, veri kümesine ve parametrelere bağlıdır ve birçok veri kümesi ve parametre arasında karşılaştırıldığında yöntemlerin diğerine göre çok az sistematik avantajı vardır.

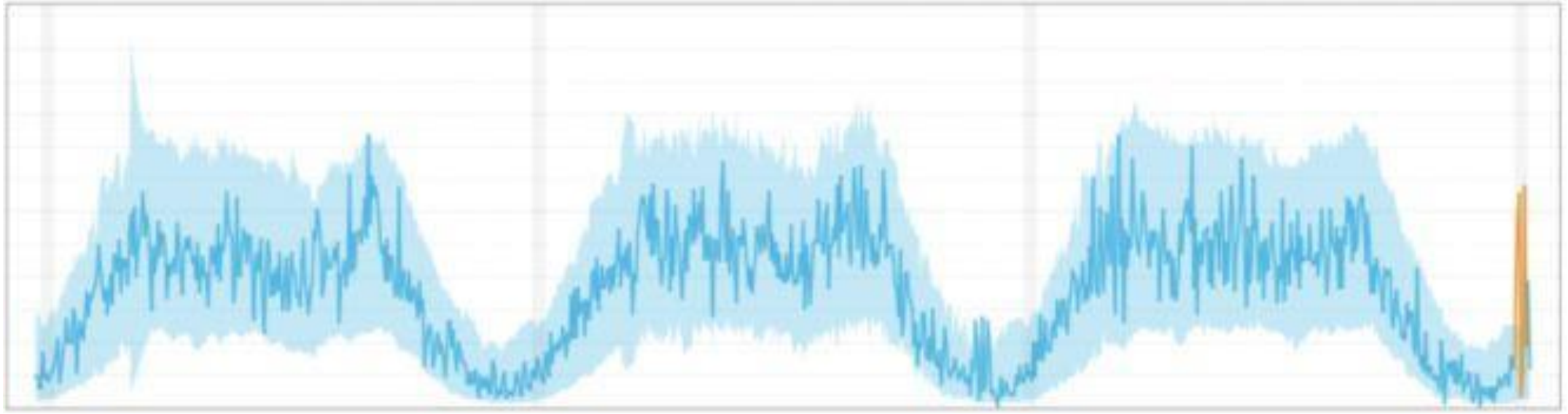
ZAMAN SERİSİ VERİ ANOMALİSİ TESPİTİ

- Başarılı anomali tespiti, zaman serisi verilerini gerçek zamanlı olarak doğru bir şekilde analiz etme yeteneğine bağlıdır.
- Zaman serisi verileri, zaman içindeki bir dizi değerden oluşur. Bu, her noktanın tipik olarak iki öğeden oluşan bir çift - metriğin ne zaman ölçüldüğüne ilişkin bir zaman damgası ve o sırada o metrikle ilişkili değer - olduğu anlamına gelir .
- Üç tür zaman serisi veri anomalisi tespiti tekniği vardır. Küresel,Bağlamsal ve Kollektif outliers



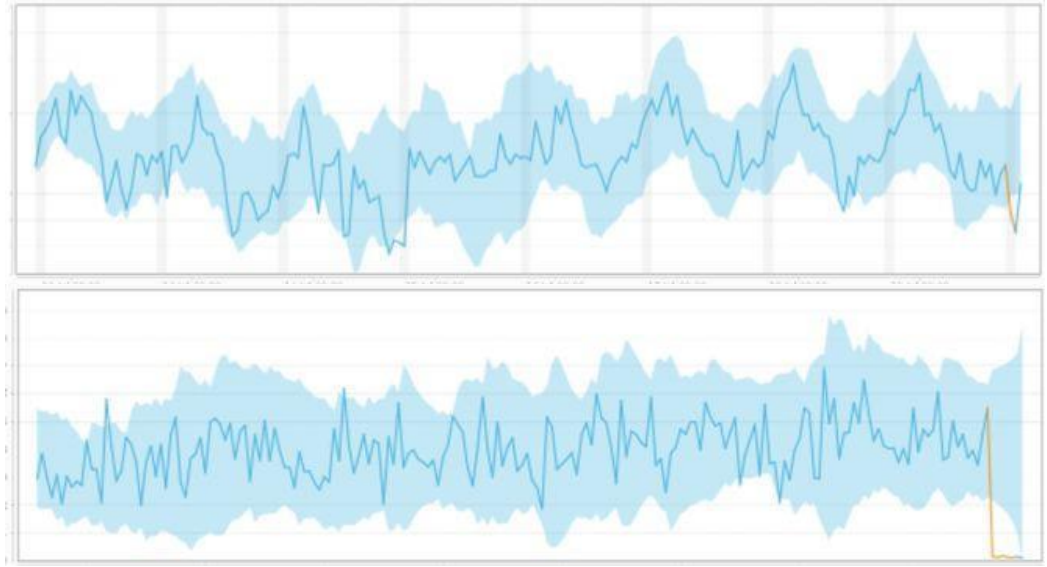
1. KÜRESEL AYKIRI DEĞERLER

- Nokta anomalileri olarak da bilinen bu aykırı değerler, bir veri kümesinin tamamının çok dışında bulunur.



2. BAĞLAMSAL AYKIRI DEĞERLER

- Koşullu aykırı değerler olarak da adlandırılan bu anormallikler, aynı bağlamda var olan diğer veri noktalarından önemli ölçüde sapan değerlere sahiptir. Bir veri kümesi bağlamındaki bir anormallik, başka bir veri kümesindeki bir anormallik olmayabilir. Bu aykırı değerler, zaman serisi verilerinde yaygındır, çünkü bu veri kümeleri, belirli bir dönemdeki belirli miktarların kayıtlarıdır. Değer, küresel beklentiler dahilinde mevcuttur ancak belirli mevsimsel veri kalıplarında anormal görünebilir.



3. KOLLEKTİF AYKIRI DEĞERLER

- Bir küme içindeki veri noktalarının bir alt kümesi, tüm veri kümesi için anormal olduğunda, bu değerler toplu aykırı değerler olarak adlandırılır.
- Bu kategoride, bireysel değerler küresel veya bağlamsal olarak anormal değildir. Farklı zaman serilerini birlikte incelerken bu tür aykırı değerleri görmeye başlarsınız. Bireysel davranış, belirli bir zaman serisi veri setinde normal aralıktan sapmayabilir. Ancak başka bir zaman serisi veri seti ile birleştirildiğinde, daha önemli anormallikler ortaya çıkıyor.

KULLANIM ALANLARI

- Tıp,
- Bilgisayarlı görü,
- Doğal dil işleme,
- İstatistik,
- Sinirbilim,
- İstihbarat,
- Emniyet,
- Siber güvenlik,
- .vb alanlarda kullanılır.

Ancak biraz daha düşündüğümüzde anomali tespitinin kullanılamayacağı herhangi bir alan yok gibi.(Eğitim, gıda, tarım, turizm, müzik, performans değerlendirme, …)

VS

ANOMALİ TESPİTİ (HİKAYE)

PYTORCH NEDİR?

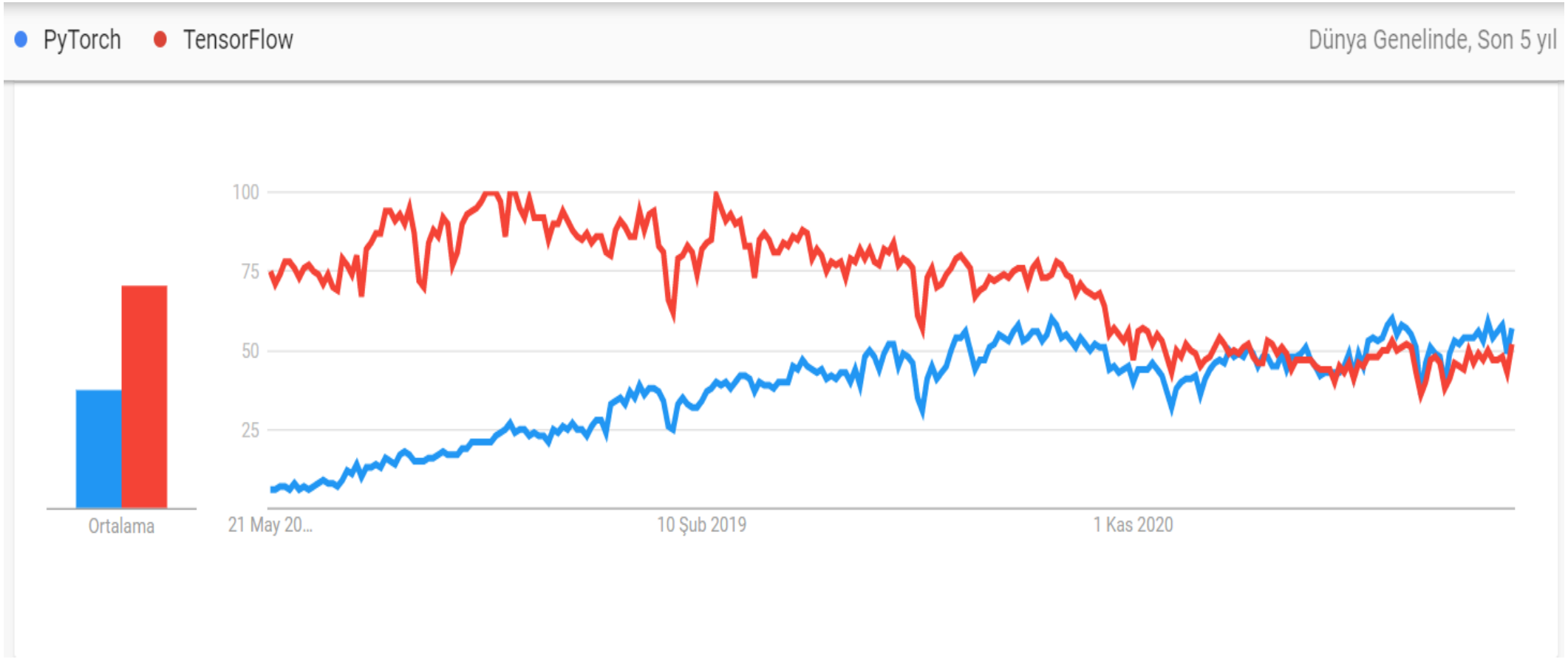
- PyTorch, öncelikle Facebook'un AI Araştırma laboratuvarı (FAIR) tarafından geliştirilen, bilgisayarla görme ve doğal dil işleme gibi uygulamalar için kullanılan Torch kitaplığına dayalı açık kaynaklı bir makine öğrenimi çerçevesidir. Değiştirilmiş BSD lisansı altında yayınlanan ücretsiz ve açık kaynaklı bir yazılımdır.
- Diğer popüler makine öğrenimi kitaplıklarıyla karşılaştırıldığında PyTorch, hassas bir öğrenme eğrisine sahiptir. Bu nedenle, makine öğrenimi ve veri bilimine yeni başlayanlar için uygun bir seçenektir. Ayrıca kütüphane, bilgisayarla görme, makine öğrenimi ve NLP için bir dizi araç sunar.
- Tesla Autopilot, Uber's Pyro, Hugging Face's Transformers, PyTorch Lightning, ve Catalyst dahil olmak üzere PyTorch'un üzerine bir dizi derin öğrenme yazılımı inşa edilmiştir.

NEDEN PYTORCH?

- Derin öğrenme ve yapay zekada en fazla kullanılan kütüphaneler belki de Tensorflow ve PyTorch'tur. PyTorch, Tensorflow'un bugüne kadar ki en büyük rakibi. Peki neden PyTorch'u tercih etmeliyiz?
- **Dinamik Hesaplamalı Grafikler:** PyTorch, Tensorflow gibi çerçevelerde kullanılan statik grafiklerden kaçınır, böylece geliştiricilerin ve araştırmacıların ağın anında nasıl davrandığını değiştirmesine izin verir. İlk benimseyenler PyTorch'u tercih ediyor çünkü TensorFlow'a kıyasla öğrenmek biraz daha rahat.
- **Farklı Back-End Desteği:** PyTorch, tek bir backend kullanmak yerine CPU, GPU ve çeşitli işlevsel özellikler için farklı backendler kullanır.
- **Genişletilebilir:** PyTorch, C ++ koduyla derinlemesine entegredir ve derin öğrenme çerçevesi Torch ile bazı C ++ backendlerini paylaşır. Böylece, kullanıcıların Python için cFFI'ye dayalı ve GPU işlemi için CPU için derlenmiş bir uzantı API'si kullanarak C / C ++ 'da programlama yapmalarına izin verilir.
- **Numpy Dostu:** PyTorch , tümü GPU uyumlu olan hesaplamalar için NumPy benzeri tensör yapıları ile çalışır.

TENSORFLOW İLE FARKI

- PyTorch, genellikle Google tarafından geliştirilen bir derin öğrenme çerçevesi olan Tensorflow ile karşılaştırılır. Tensorflow daha uzun süredir var olduğundan, daha geniş bir geliştirici topluluğuna ve daha fazla belgeye sahiptir.
- Ancak PyTorch, Tensorflow'a göre avantajlarıyla birlikte gelir. PyTorch, Tensorflow'un statik yaklaşımından farklı olarak hesaplama grafiklerini dinamik bir şekilde tanımlar. Dinamik grafikler, sonunda yerine gerçek zamanlı olarak manipüle edilebilir. Ek olarak, PyTorch sezgisel Python'a dayalı olduğundan Tensorflow daha dik bir öğrenme eğrisine sahiptir.
- Tensorflow, üretime hazır olma niyetiyle oluşturulduğundan, üretim modelleri ve ölçeklenebilirlik gerektiren projeler için daha uygun olabilir. PyTorch ile çalışmak daha kolay ve daha hafiftir, bu da onu hızlı prototipler oluşturmak ve araştırma yapmak için iyi bir seçenek haline getirir.


PyTorch
TensorFlow
Dünya Genelinde, Son 5 yıl
100
75
50
25
Ortalama
21 May 20...
10 Şub 2019
1 Kas 2020

PyTorch Build	Stable (1.11.0)	Preview (Nightly)	LTS (1.8.2)	
Your OS	Linux	Mac	Windows	
Package	Conda	Pip	LibTorch	Source
Language	Python	C++ / Java		
Compute Platform	CUDA 10.2	CUDA 11.3	~~ROCm 4.5.2 (beta)~~	CPU
Run this Command:	`pip3 install torch torchvision torchaudio`			

PyTorch Build	Stable (1.11.0)	Preview (Nightly)	LTS (1.8.2)	
Your OS	Linux	Mac	Windows	
Package	Conda	Pip	LibTorch	Source
Language	Python	C++ / Java		
Compute Platform	CUDA 10.2	CUDA 11.3	~~ROCm 4.5.2 (beta)~~	CPU
Run this Command:	`conda install pytorch torchvision torchaudio cudatoolkit=11.3 -c pytorch`			

INSTALL PYTORCH

PYTORCH İLE MODEL OLUŞTURMA

- PyTorch ile model oluşturmaya başlamadan önce pip ile kurulum yapmamız gerekir. Bunun için terminale bu komutu girmeniz gerekmekte.

  pip install torch torchvision

- İlk olarak gerekli kütüphanelerimizi projemize dahil ediyoruz.

```
import torch.nn as nn
import torch.nn.functional as F
```

- PyTorch'da bir sinir ağı oluşturmak için projeye dahil ettiğimiz nn modülünü kullanmamız gerekir. Gerekli kütüphanelerin projeye dahil edilmesinin ardından şimdi sinir ağı için sınıfı oluşturabiliriz.

```
class Net(nn.Module):
        def __init__(self):
                super(Net, self).__init__()
                self.fc1 = nn.Linear(28 * 28, 200)
                self.fc2 = nn.Linear(200, 200)
                self.fc3 = nn.Linear(200, 10)
```


- Sinir ağı sınıfının tanımında görüldüğü üzere ana sınıf nn.Module'den miras aldık. Ardından sınıfı başlatmak için kullandığımız init fonksiyonun ilk satırında nn.Module sınıfının bir örneğini oluşturan gerekli super() fonksiyonuna sahibiz. Ardından gelen nn.Linear fonksiyonları ise sinir ağında birbirine bağlı katmanlar oluşturmak için kullanılır. İlk parametre girecek değerin sayısını ikinci parametre ise katmanda bulunan node (düğüm) sayısını temsil eder. Görüldüğü üzere oluşturduğumuz sinir ağı 28×28 boyutundaki verileri girdi olarak alır ve ilk katmanda 200 node'a sahiptir.
- Şimdi ağ mimarimizin iskeletini oluşturduk. Ağı ayağa kaldırmak için bu verilerin ağ üzerinde nasıl akacağını tanımlamalıyız. Bunu Net sınıfımız içinde bir forward() fonksiyonu tanımlayarak yapacağız.

```
def forward(self,x):
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc1(x)
        return F.log_softmax(x)
```

- Oluşturduğumuz forward() metotu için x girdi değerini ana parametre olarak ayarladık. Ardından bu veriyle fc1 katmanımızı besledik ve F.relu() aktivasyon fonksiyonu ile aktif olacak kıvama getirdik. Ardından bu elde ettiğimiz sonuçla ikinci katmanı besledik ve ReLU aktivasyon fonksiyonunu uyguladık. Bu sonuçla da çıktı katmanımızı besledik fakat bu sefer ReLU yerine multi-class bir tahmin modeli için Softmax uyguladık.

- Artık sinir ağımızı oluşturduk ve bu sınıftan bir obje oluşturabiliriz.

```
net = Net()
print(net)
```

- Çıktı:

```
Net(
  (fc1): Linear(in_features=784, out_features=200, bias=True)
  (fc2): Linear(in_features=200, out_features=200, bias=True)
  (fc3): Linear(in_features=200, out_features=10, bias=True)
)
```

- Bu aşamadan sonra ise bir optimizer ve loss criterion atamalıyız.

```
optimizer = optim.SGD(net.parameters(), lr = 0.001, momentum=0.9)

criterion = nn.NLLLoss()
```

- İlk satırda bir SGD optimizer oluşturuyoruz ve öğrenme oranını 0.001, momentumu 0.9 olarak belirliyoruz. Optimizerımıza sağlamamız gereken diğer bileşen ise ağımızın parametreleri bunları da veriyoruz. Ardından loss fonksiyonumuzu negative log likelihood loss olarak belirliyoruz.

- **Modeli Eğitme:**
- Bunların ardından modeli eğitme aşamasına geliyoruz. Şu aşamada bir veriye sahip olmadığımız için bir dataloader kullandığımızı varsayacağız.

```
for epoch in range(epochs):
    optimizer.zero_grad()
    net_out = net(data)
    loss = criterion(net_out, target)
    loss.backward()
    optimizer.step()
    print('Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss: {:.6f}'.format(
        epoch, batch_idx * len(data), len(train_loader.dataset),
            100. * batch_idx / len(train_loader), loss.data[0]))
```

- İlk olarak modelimizi epoch sayısı kadar eğitiyoruz. Ardından optimizer.zero_grad() fonksiyonu ile modeldeki tüm gradyanları sıfırlıyoruz ve bununla birlikte bir sonraki back propogation'a (geri yayılım) hazır oluyoruz. Model her epochta gerekli değerleri yazdırıyor ve modelimiz hazır hale geliyor.

KAYNAKÇA

- https://en.wikipedia.org/wiki/Anomaly_detection
- https://www.anodot.com/blog/what-is-anomaly-detection/#:~:text=Anomaly%20detection%20(aka%20outlier%20analysis,a%20change%20in%20consumer%20behavior.
- https://hackr.io/blog/best-machine-learning-libraries
- https://en.wikipedia.org/wiki/PyTorch
- https://teknoloji.org/pytorch-kutuphanesi-nedir-nasil-kullanilir/#PyTorch_Nedir
- https://pytorch.org/get-started/locally/

DİNLEDİĞİNİZ İÇİN TEŞEKKÜRLER...