



Yayın Numarası: 2101  
1. Baskı: Ağustos 2021  
ISBN 978-605-06248-2-3

Sertifika Numarası: 41825  
Buzdağı Yayınevi  
Altay Mahallesi 2668. Cadde  
Fırat Life Style Rezidans 1F/61  
Eryaman/ANKARA  
t: +90 312 219 77 98  
f: +90 312 219 55 43  
info@buzdagiyayinevi.com

Baskı  
Ankamat Matbaacılık  
46700

© 2021 Türkçe yayın hakları Buzdağı Yayınevi'ne aittir.

Authorized Turkish translation of the English edition of *Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow, Second Edition*, ISBN 9781492032649 © 2019 Aurélien Géron

This translation is published and sold by permission of O'Reilly Media, Inc., which owns or controls all rights to publish and sell the same.

Bu kitabın hiçbir bölümü, yazarın ve yayınevinin izni alınmadan basılı ve dijital olarak çoğaltılamaz, yayımlanamaz.

# Scikit-Learn, Keras ve TensorFlow ile Uygulamalı Makine Öğrenmesi

Aurélien Géron

Genel Yayın Yönetmeni  
Fatih ÖZDEMİR

Yazar  
Aurélien Géron

Çevirmen  
Bilgin AKSOY  
Özgür KAYA

Editör  
Arş. Gör. Dr. Mustafa Murat ARAT  
Doç. Dr. Vedat ÇELİK

Son Okuma  
Muhammed TOPRAK  
Ahmet Okan ŞEKER

Dizgi  
Veysel TOPRAK

# İçindekiler

Ön Söz

xvii

<b>I</b>	<b>Makine Öğrenmesinin Temelleri</b>	<b>1</b>
<b>1</b>	<b>Makine Öğrenmesi Dünyası</b>	<b>2</b>
1.1	Makine Öğrenmesi Nedir? . . . . .	3
1.2	Neden Makine Öğrenmesi Kullanılmalı? . . . . .	4
1.3	Uygulama Örnekleri . . . . .	7
1.4	Makine Öğrenmesi Sistemlerinin Türleri . . . . .	9
1.4.1	Denetimli/Denetimsiz Öğrenme . . . . .	9
1.4.2	Yığın ve Çevrim İçi Öğrenme . . . . .	17
1.4.3	Örnek Tabanlı ve Model Tabanlı Öğrenme . . . . .	19
1.5	Makine Öğrenmesinin Temel Zorlukları . . . . .	26
1.5.1	Eğitim Verilerinin Yetersiz Oluşu . . . . .	26
1.5.2	Temsil Edemeyen Eğitim Verileri . . . . .	28
1.5.3	Düşük Kaliteli Veriler . . . . .	30
1.5.4	Alakasız Öznitelikler . . . . .	30
1.5.5	Eğitim Verilerine Aşırı Uydurma . . . . .	31
1.5.6	Eğitim Verilerine Yetersiz Uydurma . . . . .	33
1.5.7	Geriye Çekilip Düşünmek . . . . .	33
1.6	Test ve Doğrulama . . . . .	34

1.6.1	Hiperparametre Ayarı ve Model Seçimi . . . . .	35
1.6.2	Veri Uyumsuzluğu . . . . .	36
1.7	Alıştırmalar . . . . .	38
<b>2</b>	<b>Uçtan Uca Makine Öğrenmesi Projesi</b>	<b>40</b>
2.1	Gerçek Veriyle Çalışmak . . . . .	40
2.2	Büyük Resme Bak . . . . .	41
2.2.1	Problemi Tanıyın . . . . .	42
2.2.2	Performans Ölçütü Seçin . . . . .	45
2.2.3	Varsayımları Kontrol Edin . . . . .	47
2.3	Verileri Toplayın . . . . .	47
2.3.1	Çalışma Alanı Oluşturun . . . . .	48
2.3.2	Verileri İndirin . . . . .	51
2.3.3	Veri Yapısına Hızlı Bakış . . . . .	52
2.3.4	Test Veri Seti Oluşturma . . . . .	57
2.3.5	Fikir Sahibi Olmak için Veriyi Keşfedin ve Görselleştirin	61
2.3.6	Korelasyonların Bulunması . . . . .	64
2.3.7	Nitelik Kombinasyonlarının Denenmesi . . . . .	66
2.4	Makine Öğrenmesi Algoritmaları için Verileri Hazırlamak . . .	68
2.4.1	Veri Temizleme . . . . .	69
2.4.2	Metinsel ve Kategorik Niteliklerle Uğraşmak . . . . .	72
2.4.3	Özel Dönüştürücüler . . . . .	75
2.4.4	Özniteliklerin Ölçeklenmesi . . . . .	76
2.4.5	Dönüşüm İletim Hatları . . . . .	77
2.5	Model Seçimi ve Eğitimi . . . . .	79
2.5.1	Eğitim Veri Setinde Eğitim ve Değerlendirme . . . . .	79
2.5.2	Çapraz Doğrulama Kullanarak Daha İyi Değerlendirme	81
2.6	Modelinize İnce Ayar Yapmak . . . . .	83
2.6.1	Izgara Arama . . . . .	84

---

2.6.2	Rastgele Arama . . . . .	86
2.6.3	Topluluk Yöntemleri . . . . .	87
2.6.4	En İyi Modelleri ve Hatalarını Analiz Etmek . . . . .	87
2.6.5	Sisteminizi Test Veri Setinde Değerlendirmek . . . . .	88
2.7	Sisteminizi Çalıştırmak, Takip Etmek ve Bakımını Yapmak . . . . .	89
2.8	Siz Deneyin! . . . . .	93
2.9	Alıştırmalar . . . . .	94
<b>3</b>	<b>Sınıflandırma</b>	<b>95</b>
3.1	MNIST . . . . .	95
3.2	İkili Sınıflandırıcı Eğitmek . . . . .	98
3.3	Performans Ölçütleri . . . . .	99
3.3.1	Çapraz Doğrulama ile Doğruluk Oranının Hesaplanması . . . . .	99
3.3.2	Hata Matrisi . . . . .	100
3.3.3	Kesinlik ve Duyarlılık . . . . .	102
3.3.4	Kesinlik/Duyarlılık Dengesi . . . . .	104
3.3.5	ROC Eğrisi . . . . .	108
3.4	Çok Sınıflı Sınıflandırma . . . . .	111
3.5	Hata Analizi . . . . .	114
3.6	Çok Etiketli Sınıflandırma . . . . .	117
3.7	Çok Çıktılı Sınıflandırma . . . . .	119
3.8	Alıştırmalar . . . . .	120
<b>4</b>	<b>Modelleri Eğitmek</b>	<b>122</b>
4.1	Doğrusal Bağlanım . . . . .	123
4.1.1	Normal Denklem . . . . .	125
4.1.2	Hesaplama Karmaşıklığı . . . . .	128
4.2	Gradyan İnişi . . . . .	129
4.2.1	Yığın Gradyan İnişi . . . . .	132
4.2.2	Stokastik Gradyan İnişi . . . . .	135

4.2.3	Mini-Yığın Gradyan İnişi . . . . .	138
4.3	Polinom Bağlanım . . . . .	139
4.4	Öğrenme Eğrileri . . . . .	141
4.5	Düzenleştirilmiş Doğrusal Modeller . . . . .	145
4.5.1	Ridge Bağlanımı . . . . .	145
4.5.2	Lasso Bağlanımı . . . . .	148
4.5.3	Elastic Net . . . . .	151
4.5.4	Erken Durdurma . . . . .	152
4.6	Lojistik Bağlanım . . . . .	153
4.6.1	Olasılık Tahmini . . . . .	153
4.6.2	Eğitim ve Maliyet Fonksiyonu . . . . .	155
4.6.3	Karar Sınırları . . . . .	156
4.6.4	Softmax Bağlanım . . . . .	158
4.7	Alıştırmalar . . . . .	162
<b>5</b>	<b>Destek Vektör Makineleri</b>	<b>164</b>
5.1	Doğrusal SVM Sınıflandırma . . . . .	164
5.1.1	Yumuşak Sınır Sınıflandırması . . . . .	165
5.2	Doğrusal Olmayan SVM Sınıflandırma . . . . .	168
5.2.1	Polinom Kerneli . . . . .	169
5.2.2	Niteliklerin Benzerliği . . . . .	171
5.2.3	Gauss RBF Kerneli . . . . .	172
5.2.4	Hesaplama Karmaşıklığı . . . . .	173
5.2.5	SVM Bağlanım . . . . .	174
5.3	Kaputun Altına Bakmak . . . . .	176
5.3.1	Karar Fonksiyonu ve Tahminler . . . . .	176
5.3.2	Eğitim Amacı . . . . .	177
5.3.3	Karesel Programlama . . . . .	179
5.3.4	Dual Problem . . . . .	180



---

5.3.5	Kernelli SVM'ler . . . . .	181
5.3.6	Çevrim İçi SVM'ler . . . . .	183
5.4	Alıřtırmalar . . . . .	185
<b>6</b>	<b>Karar Ağaçları</b>	<b>186</b>
6.1	Karar Ağacını Eđitme ve Grselleřtirmek . . . . .	186
6.2	Tahmin Yapmak . . . . .	188
6.3	Sınıf Olasılıklarını Kestirme . . . . .	190
6.4	CART Eđitim Algoritması . . . . .	191
6.5	Hesaplama Karmařıklıđı . . . . .	192
6.6	Gini Safsızlıđı ya da Entropi? . . . . .	192
6.7	Dzenleme Hiperparametreleri . . . . .	193
6.8	Bađlanım . . . . .	194
6.9	Dengesizlik . . . . .	196
6.10	Alıřtırmalar . . . . .	198
<b>7</b>	<b>Topluluk đrenme ve Rastgele Ormanlar</b>	<b>200</b>
7.1	Oylama Sınıflandırıcıları . . . . .	200
7.2	Torbalama ve Yapıřtırma . . . . .	204
7.2.1	Scikit-Learn'de Torbalama ve Yapıřtırma . . . . .	205
7.2.2	Torba-Dıřı Deđerlendirme . . . . .	206
7.3	Rastgele Parçalar ve Rastgele Alt Uzaylar . . . . .	208
7.4	Rastgele Ormanlar . . . . .	208
7.4.1	Fazladan Ağaçlar . . . . .	209
7.4.2	Nitelik nemi . . . . .	210
7.5	Hızlandırma . . . . .	211
7.5.1	AdaBoost . . . . .	211
7.5.2	Gradyan Hızlandırma . . . . .	215
7.6	Yıđma . . . . .	220
7.7	Alıřtırmalar . . . . .	223

<b>8</b>	<b>Boyut Azaltma</b>	<b>224</b>
8.1	Boyut Laneti . . . . .	225
8.2	Boyut Azaltmaya Temel Yaklaşımlar . . . . .	226
8.2.1	Projeksiyon . . . . .	226
8.2.2	Manifold Öğrenme . . . . .	228
8.3	PCA . . . . .	229
8.3.1	Değişirliği Korumak . . . . .	230
8.3.2	Ana Bileşenler . . . . .	231
8.3.3	d Boyuta Projeksiyon . . . . .	232
8.3.4	Scikit-Learn'ü Kullanmak . . . . .	233
8.3.5	Değişirlik Oranı . . . . .	233
8.3.6	Doğru Boyutun Seçimi . . . . .	234
8.3.7	Sıkıştırma için PCA . . . . .	235
8.3.8	Rastgele PCA . . . . .	236
8.3.9	Artırmalı PCA . . . . .	236
8.4	Kernel PCA . . . . .	237
8.4.1	Kernel Seçimi ve Hiperparametreleri Ayarlama . . . . .	238
8.5	LLE . . . . .	240
8.6	Diğer Boyut Azaltma Teknikleri . . . . .	242
8.7	Alıştırmalar . . . . .	244
<b>9</b>	<b>Denetimsiz Öğrenme</b>	<b>245</b>
9.1	Kümeleme . . . . .	246
9.1.1	K-ortalamlar . . . . .	248
9.1.2	K-ortalamların Kısıtları . . . . .	258
9.1.3	Görüntü Bölütleme için Kümeleme Kullanma . . . . .	259
9.1.4	Önişleme için Kümelemeyi Kullanma . . . . .	261
9.1.5	Yarı Denetimli Öğrenme için Kümelemeyi Kullanma . . . . .	263
9.1.6	DBSCAN . . . . .	266

9.1.7	Diğer Kümeleme Algoritmaları . . . . .	269
9.2	Gauss Karışımları . . . . .	270
9.2.1	Gauss Karışımları Kullanarak Anomali Tespiti . . . . .	276
9.2.2	Küme Sayısını Seçme . . . . .	278
9.2.3	Bayeşçi Gauss Karışım Modelleri . . . . .	281
9.2.4	Anomali ve Yenilik Tespiti için Diğer Algoritmalar . . . . .	284
9.3	Alıştırmalar . . . . .	286
 <b>II Sinir Ağları ve Derin Öğrenme</b>		<b>288</b>
 <b>10 Keras ile Yapay Sinir Ağlarına Giriş</b>		<b>289</b>
10.1	Biyolojik Sinir Hücrelerinden Yapay Sinir Hücrelerine . . . . .	290
10.1.1	Biyolojik Sinir Hücreleri . . . . .	291
10.1.2	Sinir Hücreleriyle Mantıksal Hesaplamalar . . . . .	293
10.1.3	Perseptron (Algılayıcı) . . . . .	294
10.1.4	Çok Katmanlı Perseptron ve Geri Yayılım . . . . .	298
10.1.5	Bağlanım MLP'leri . . . . .	303
10.1.6	Sınıflandırma MLP'leri . . . . .	304
10.2	Keras ile MLP'leri Uygulamak . . . . .	306
10.2.1	TensorFlow 2'yi Kurmak . . . . .	307
10.2.2	Sequential API Kullanarak Bir Görüntü Sınıflandırıcı Oluşturmak . . . . .	308
10.2.3	Sequential API Kullanarak Bağlanım MLP'si Oluşturmak	320
10.2.4	Fonksiyonel API Kullanarak Karmaşık Modeller Oluş- turmak . . . . .	322
10.2.5	Dinamik Modeller Oluşturmak için Altsınıflama API'sini Kullanmak . . . . .	326
10.2.6	Modeli Kaydedip Geri Yükleme . . . . .	328
10.2.7	Geri Çağırma Kullanmak . . . . .	329
10.2.8	Görselleştirme için TensorBoard Kullanmak . . . . .	331

10.3	Sinir Ağı Hiperparametrelerine İnce Ayar Çekmek . . . . .	335
10.3.1	Gizli Katman Sayısı . . . . .	339
10.3.2	Gizli Katman Başına Düşen Sinir Hücresi Sayısı . . . . .	340
10.3.3	Öğrenme Oranı, Yığın Büyüklüğü ve Diğer Hiperparametreler . . . . .	341
10.4	Alıştırmalar . . . . .	343
<b>11</b>	<b>Derin Sinir Ağlarını Eğitmek</b>	<b>347</b>
11.1	Yok Olan/Aşırı Büyüyen Gradyanlar Problemi . . . . .	348
11.1.1	Glorot ve He İlk Değer Ataması . . . . .	349
11.1.2	Doyuma Ulaşmayan Aktivasyon Fonksiyonları . . . . .	351
11.1.3	Yığın Normalleştirme . . . . .	356
11.1.4	Gradyan Kırpma . . . . .	363
11.2	Önceden Eğitilmiş Katmanları Tekrar Kullanmak . . . . .	364
11.2.1	Keras ile Aktarmalı Öğrenme . . . . .	366
11.2.2	Denetimsiz Öneğitim . . . . .	368
11.2.3	Yardımcı Bir Görev Üzerinde Öneğitim . . . . .	370
11.3	Daha Hızlı Eniyileyiciler . . . . .	370
11.3.1	Momentum Eniyilemesi . . . . .	371
11.3.2	Nesterov Hızlanan Gradyan . . . . .	372
11.3.3	AdaGrad . . . . .	374
11.3.4	RMSProp . . . . .	375
11.3.5	Adam ve Nadam Eniyilemesi . . . . .	376
11.3.6	Öğrenme Oranı Çizelgelemesi . . . . .	379
11.4	Düzenleştirme ile Aşırı Uydurmayı Önlemek . . . . .	384
11.4.1	$\ell_1$ ve $\ell_2$ Düzenleştirme . . . . .	385
11.4.2	Seyreltme . . . . .	386
11.4.3	Monte Carlo (MC) Seyreltme . . . . .	389
11.4.4	Max-Norm Düzenleştirme . . . . .	392
11.5	Özet ve Pratik Talimatlar . . . . .	393

11.6	Alıştırmalar . . . . .	395
<b>12</b>	<b>Kullanıcı Tanımlı Modeller ve TensorFlow ile Eğitim</b>	<b>397</b>
12.1	TensorFlow'a Hızlıca Bir Bakış . . . . .	397
12.2	TensorFlow'u NumPy gibi Kullanmak . . . . .	401
12.2.1	Tensörler ve İşlemler . . . . .	401
12.2.2	Tensörler ve NumPy . . . . .	404
12.2.3	Tip Çevirmeleri . . . . .	404
12.2.4	Değişkenler . . . . .	405
12.2.5	Diğer Veri Yapıları . . . . .	406
12.3	Modelleri ve Eğitim Algoritmalarını Kullanıcı Tanımlı Hâle Getirmek . . . . .	407
12.3.1	Kullanıcı Tanımlı Kayıp Fonksiyonları . . . . .	407
12.3.2	Kullanıcı Tanımlı Bileşenler İçeren Modelleri Kaydetmek ve Yükleme . . . . .	408
12.3.3	Kullanıcı Tanımlı Aktivasyon Fonksiyonları, Başlatıcılar, Düzenleştiriciler ve Kısıtlar . . . . .	410
12.3.4	Kullanıcı Tanımlı Ölçütler . . . . .	412
12.3.5	Kullanıcı Tanımlı Katmanlar . . . . .	415
12.3.6	Kullanıcı Tanımlı Modeller . . . . .	419
12.3.7	Model İçeriğine Bağlı Olarak Kayıplar ve Ölçütler . . . . .	421
12.3.8	Autodiff Kullanarak Gradyanları Hesaplamak . . . . .	424
12.3.9	Kullanıcı Tanımlı Eğitim Döngüleri . . . . .	428
12.4	TensorFlow Fonksiyon ve Çizgeleri . . . . .	431
12.4.1	AutoGraph ve İzleme . . . . .	434
12.4.2	TF Fonksiyonu Kuralları . . . . .	435
12.5	Alıştırmalar . . . . .	437
<b>13</b>	<b>TensorFlow ile Veriyi Yükleme ve Ön İşlemden Geçirmek</b>	<b>439</b>
13.1	Data API'si . . . . .	440
13.1.1	Dönüşümleri Zincirleme Hâle Getirmek . . . . .	441

13.1.2	Veriyi Karıştırmak . . . . .	443
13.1.3	Veriyi Önışlemden Geçirmek . . . . .	446
13.1.4	Her Şeyi Bir Araya Getirmek . . . . .	447
13.1.5	Önceden Getirme . . . . .	448
13.1.6	Veri Setini tf.keras ile Kullanmak . . . . .	450
13.2	TFRecord Formatı . . . . .	451
13.2.1	Sıkıştırılmış TFRecord Dosyaları . . . . .	452
13.2.2	Protokol Tamponlarına Kısa Bir Giriş . . . . .	452
13.2.3	TensorFlow Protobufları . . . . .	454
13.2.4	Example'ları Yükleme ve Ayrıştırma . . . . .	456
13.2.5	SequenceExample Protobuf'ını Kullanarak Listelerin Listelerini İdare Etmek . . . . .	457
13.3	Girdi Özniteliklerini Önışlemden Geçirmek . . . . .	458
13.3.1	Bir-elemanı-bir Vektörler Kullanarak Kategorik Öznite- likleri Kodlamak . . . . .	460
13.3.2	Gömülmeler Kullanarak Kategorik Öznitelikleri Kodlamak	462
13.3.3	Keras Önışleme Katmanları . . . . .	467
13.4	TF Dönüşümü . . . . .	469
13.5	TensorFlow Veri Setleri (TFDS) Projesi . . . . .	471
13.6	Alıştırmalar . . . . .	473
<b>14</b>	<b>Evrişimsel Sinir Ağları Kullanarak Derin Bilgisayarlı Görü</b>	<b>475</b>
14.1	Görsel Korteksin Mimarisi . . . . .	476
14.2	Evrişimsel Katmanlar . . . . .	477
14.2.1	Filtreler . . . . .	479
14.2.2	Çoklu Öznitelik Haritalarını Üst Üste Yığmak . . . . .	480
14.2.3	TensorFlow Uygulaması . . . . .	482
14.2.4	Bellek Gereksinimleri . . . . .	485
14.3	Biriktirme Katmanları . . . . .	486
14.3.1	TensorFlow Uygulaması . . . . .	488

---

14.4	CNN Mimarileri . . . . .	490
14.4.1	LeNet-5 . . . . .	493
14.4.2	AlexNet . . . . .	494
14.4.3	GoogLeNet . . . . .	497
14.4.4	VGGNet . . . . .	501
14.4.5	ResNet . . . . .	501
14.4.6	Xception . . . . .	504
14.4.7	SENet . . . . .	506
14.5	Keras Kullanarak Bir ResNet-34 CNN’i Uygulama . . . . .	508
14.6	Keras’taki Önceden Eğitilmiş Modelleri Kullanmak . . . . .	510
14.7	Aktarmalı Öğrenme için Önceden Eğitilmiş Modeller . . . . .	512
14.8	Sınıflandırma ve Konumlandırma . . . . .	515
14.9	Nesne Tespiti . . . . .	517
14.9.1	Tamamen Evrişimsel Ağlar . . . . .	518
14.9.2	YOLO (You Look Only Once) . . . . .	521
14.9.3	Anlamsal Bölütleme . . . . .	524
14.10	Alıştırmalar . . . . .	528
<b>15</b>	<b>RNN’ler ve CNN’ler Kullanarak Dizileri İşlemek</b>	<b>530</b>
15.1	Yinelemeli Sinir Hücreleri ve Katmanlar . . . . .	531
15.1.1	Bellek Hücreleri . . . . .	533
15.1.2	Girdi ve Çıktı Dizileri . . . . .	534
15.2	RNN’leri Eğitmek . . . . .	535
15.3	Bir Zaman Serisini Tahmin Etmek . . . . .	536
15.3.1	Temel Ölçütler . . . . .	538
15.3.2	Basit bir RNN’yi Uygulamak . . . . .	538
15.3.3	Derin RNN’ler . . . . .	540
15.3.4	Birkaç Zaman Adımı Sonrasını Tahmin Etmek . . . . .	542
15.4	Uzun Dizileri Ele Almak . . . . .	546

15.4.1	Kararsız Gradyanlar Problemi ile Mücadele Etmek . . . .	546
15.4.2	Kısa Dönem Bellek Problemiyle Mücadele Etmek . . . .	549
15.5	Alıştırmalar . . . . .	558
<b>16</b>	<b>RNN'lerle Doğal Dil İşleme ve İlgi</b>	<b>559</b>
16.1	Karakter RNN'si Kullanarak Shakespeare Tarzı Metin Üretmek	560
16.1.1	Eğitim Veri Setini Oluşturmak . . . . .	561
16.1.2	Sıralı Bir Veri Setini Bölmek . . . . .	562
16.1.3	Sıralı Veri Setini Çoklu Pencerelelere Ayırmak . . . . .	563
16.1.4	Karakter RNN Modelini Oluşturmak ve Eğitmek . . . .	565
16.1.5	Karakter RNN Modelini Kullanmak . . . . .	566
16.1.6	Shakespeare Tarzı Sahte Metin Üretmek . . . . .	566
16.1.7	Durum Bilgisi Taşıyan RNN . . . . .	568
16.2	Duygu Analizi . . . . .	570
16.2.1	Maskeleye . . . . .	575
16.2.2	Önceden Eğitilmiş Gömülmeleri Tekrar Kullanmak . . .	577
16.3	Sinirsel Makine Çevirisi için Bir Kodlayıcı-Kodçözücü Ağı . . .	579
16.3.1	İki Yönlü RNN'ler . . . . .	582
16.3.2	Demet Arama . . . . .	583
16.4	İlgi Mekanizmaları . . . . .	585
16.4.1	Görsel İlgi . . . . .	589
16.4.2	Tek İhtiyacınız Olan İlgidir: Dönüştürücü Mimarisi . . .	590
16.5	Dil Modellerinde Yakın Zamandaki Yenilikler . . . . .	599
16.6	Alıştırmalar . . . . .	603
<b>17</b>	<b>Otokodlayıcılar ve GAN'lar Kullanarak Gösterim ve Üretken Öğrenme</b>	<b>604</b>
17.1	Verimli Veri Gösterimleri . . . . .	605
17.2	Tamamlanmamış Doğrusal Otokodlayıcı ile PCA (Principal Component Analysis: Ana Bileşenler Analizi) Gerçekleştirmek .	607



17.3	Üst Üste Yığılmış Otokodlayıcılar . . . . .	609
17.3.1	Keras Kullanarak Üst Üste Yığılmış Bir Otokodlayıcıyı Gerçeklemek . . . . .	609
17.3.2	Yeniden Oluşturmaları Görselleştirme . . . . .	611
17.3.3	Fashion MNIST Veri Setini Görselleştirmek . . . . .	612
17.3.4	Üst Üste Yığılmış Otokodlayıcılar Kullanan Denetimsiz Öneğitim . . . . .	612
17.3.5	Ağırlıkları Bağlamak . . . . .	614
17.3.6	Her Seferinde Bir Otokodlayıcı Eğitmek . . . . .	615
17.4	Evrişimsel Otokodlayıcılar . . . . .	617
17.5	Yinelemeli Otokodlayıcılar . . . . .	618
17.6	Arıtan Otokodlayıcılar . . . . .	619
17.7	Seyrek Otokodlayıcılar . . . . .	620
17.8	Değişimsel Otokodlayıcılar . . . . .	624
17.8.1	Fashion MNIST Görüntülerini Üretmek . . . . .	628
17.9	Üretken Çekişmeli Ağlar . . . . .	630
17.9.1	GAN'ları Eğitmenin Zorlukları . . . . .	634
17.9.2	Derin Evrişimsel GAN'lar . . . . .	636
17.9.3	Sürekli Gelişen GAN'lar . . . . .	639
17.9.4	StyleGAN'lar . . . . .	642
17.10	Alıştırmalar . . . . .	644
<b>18</b>	<b>Pekiştirmeli Öğrenme</b>	<b>646</b>
18.1	Ödülleri Eniyilemeyi Öğrenmek . . . . .	647
18.2	Politika Araması . . . . .	649
18.3	OpenAI Gym'e Giriş . . . . .	650
18.4	Sinir Ağı Politikaları . . . . .	655
18.5	Hareketleri Değerlendirmek: Kredi Atama Problemi . . . . .	656
18.6	Politika Gradyanları . . . . .	658
18.7	Markov Karar Süreçleri . . . . .	663

18.8	Zamansal Fark Öğrenimi . . . . .	668
18.9	Q-Öğrenmesi . . . . .	669
18.9.1	Keşif Politikaları . . . . .	672
18.9.2	Yaklaşık Q-Öğrenmesi ve Derin Q-Öğrenmesi . . . . .	672
18.10	Derin Q-Öğrenmesini Gerçeklemek . . . . .	674
18.11	Derin Q-Öğrenmesinin Başka Biçimleri . . . . .	679
18.11.1	Sabit Q-Değer Hedefleri . . . . .	679
18.11.2	Çift DQN . . . . .	680
18.11.3	Öncelikli Tecrübe Yeniden Oynatması . . . . .	681
18.11.4	Çekişen DQN . . . . .	682
18.12	TF-Agents Kütüphanesi . . . . .	683
18.12.1	TF-Agents'ı Kurmak . . . . .	684
18.12.2	TF-Agents Ortamları . . . . .	685
18.12.3	Ortam Özellikleri . . . . .	686
18.12.4	Ortam Sarmalayıcıları ve Atari Önışleme . . . . .	687
18.12.5	Eğitim Mimarisi . . . . .	690
18.12.6	Derin Q-Ağını Oluşturmak . . . . .	692
18.12.7	DQN Ajanını Oluşturmak . . . . .	694
18.12.8	Yeniden Oynatma Tamponunu ve Karşılık Gelen Göz- lemciyi Oluşturmak . . . . .	696
18.12.9	Eğitim Ölçütlerini Oluşturmak . . . . .	698
18.12.10	Toplama Sürücüsünü Oluşturmak . . . . .	699
18.12.11	Veri Setini Oluşturmak . . . . .	701
18.12.12	Eğitim Döngüsünü Oluşturmak . . . . .	704
18.13	Bazı Popüler RL Algoritmalarına Genel Bakış . . . . .	705
18.14	Alıştırmalar . . . . .	708
<b>19</b>	<b>TensorFlow Modellerini Ölçekleyerek Eğitime ve Dağıtma</b>	<b>709</b>
19.1	TensorFlow Modellerini Servis Etmek . . . . .	710
19.1.1	TensorFlow Serving Kullanmak . . . . .	710

19.1.2	GCP AI Platformunda Tahmin Servisi Oluşturmak . . . . .	720
19.1.3	Tahmin Servisini Kullanmak . . . . .	725
19.2	Modeli Bir Mobil veya Gömülü Cihaza Dağıtmak . . . . .	728
19.3	Hesaplamaları Hızlandırmak için GPU Kullanmak . . . . .	733
19.3.1	Kendi GPU'nuzu Alma . . . . .	734
19.3.2	GPU'lu Sanal Makine Kullanımı . . . . .	736
19.3.3	Colaboratory . . . . .	737
19.3.4	GPU RAM'ini Yönetmek . . . . .	739
19.3.5	Aygıt Üzerine İşlemleri ve Değişkenleri Yerleştirmek . . . . .	741
19.3.6	Birden Çok Aygıt Üzerinde Paralel Çalıştırma . . . . .	743
19.4	Modelleri Birden Çok Aygıtta Eğitim . . . . .	746
19.4.1	Model Paralleleştirme . . . . .	747
19.4.2	Veri Paralleleştirme . . . . .	749
19.4.3	Distribution Strategies API Kullanarak Eğitimi Ölçekle- mek . . . . .	754
19.4.4	Modeli TensorFlow Kümesinde Eğitim . . . . .	756
19.4.5	Google Cloud AI Platform'u Üzerinde Büyük Eğitim İşlerini Çalıştırmak . . . . .	759
19.4.6	AI Platform Üzerinde Kara Kutu Hiperparametre Ayar- lama . . . . .	761
19.5	Alıştırmalar . . . . .	763
19.6	Teşekkür . . . . .	764
<b>A</b>	<b>Alıştırma Çözümleri</b>	<b>765</b>
A.1	<b>Bölüm 1:</b> Makine Öğrenmesi Dünyası . . . . .	765
A.2	<b>Bölüm 2:</b> Uçtan Uca Makine Öğrenmesi Projesi . . . . .	767
A.3	<b>Bölüm 3:</b> Sınıflandırma . . . . .	768
A.4	<b>Bölüm 4:</b> Modelleri Eğitim . . . . .	768
A.5	<b>Bölüm 5:</b> Destek Vektör Makineleri . . . . .	770
A.6	<b>Bölüm 6:</b> Karar Ağaçları . . . . .	772

A.7	<b>Bölüm 7:</b> Topluluk Öğrenme ve Rastgele Ormanlar . . . . .	773
A.8	<b>Bölüm 8:</b> Boyut Azaltma . . . . .	774
A.9	<b>Bölüm 9:</b> Denetimsiz Öğrenme . . . . .	776
A.10	<b>Bölüm 10:</b> Keras ile Sinir Ağlarına Giriş . . . . .	778
A.11	<b>Bölüm 11:</b> Derin Sinir Ağlarını Eğitmek . . . . .	781
A.12	<b>Bölüm 12:</b> Kullanıcı Tanımlı Modeller ve TensorFlow ile Eğitim	782
A.13	<b>Bölüm 13:</b> TensorFlow ile Veriyi Yüklemek ve Önışlemeden Geçirmek . . . . .	785
A.14	<b>Bölüm 14:</b> Evrişimsel Sinir Ağları Kullanarak Derin Bilgisayarlı Görü . . . . .	788
A.15	<b>Bölüm 15:</b> RNN'ler ve CNN'ler Kullanarak Dizileri İşlemek . .	792
A.16	<b>Bölüm 16:</b> RNN'lerle Doğal Dil İşleme ve İlgili . . . . .	794
A.17	<b>Bölüm 17:</b> Otokodlayıcılar ve GAN'lar Kullanarak Gösterim ve Üretken Öğrenme . . . . .	796
A.18	<b>Bölüm 18:</b> Pekiştirmeli Öğrenme . . . . .	798
A.19	<b>Bölüm 19:</b> TensorFlow Modellerini Ölçekleyerek Eğitime ve Dağıtma . . . . .	801
<b>B</b>	<b>Makine Öğrenmesi Projesi Kontrol Listesi</b>	<b>804</b>
<b>C</b>	<b>SVM Dual Problemi</b>	<b>810</b>
<b>D</b>	<b>Autodiff</b>	<b>813</b>
<b>E</b>	<b>Diğer Popüler ANN Mimarileri</b>	<b>820</b>
<b>F</b>	<b>Özel Veri Yapıları</b>	<b>828</b>
<b>G</b>	<b>TensorFlow Çizgeleri</b>	<b>835</b>
<b>Dizin</b>		<b>844</b>

# 1 Makine Öğrenmesi Dünyası

İnsanlar “Makine Öğrenmesi”ni duyduklarında akıllarına bir robot geliyor: Kime sorduğunuza bağlı olarak değişen, güvenilir bir uşak ya da ölümcül bir Terminator. Ama makine öğrenmesi geleceğe ait bir hayalden ibaret değil: Hâlihazırda elimizde bulunuyor. Aslında on yıllardan beri Optik Karakter Tanıma (OCR) gibi bazı özel uygulamalarıyla hayatımızdadır. Ana akım hâline gelen ilk makine öğrenmesi uygulaması 1990’ların başından beri yüz milyonlarca insanın hayatını iyileştiren *gereksiz posta filtresidir*. Tam olarak kendini tanıyan bir Skynet değil ancak teknik olarak makine öğrenmesi olarak nitelendirilmektedir (aslında bir e-postayı artık gereksiz posta olarak işaretlemeniz gerektiğini çok da iyi öğrendi). Bunu daha iyi önerilerden, sesli aramaya kadar düzenli olarak kullandığınız yüzlerce ürün ve özellikle sessizce güç veren yüzlerce makine öğrenmesi uygulaması izledi.

Makine öğrenmesi nerede başlar ve nerede biter? Bir makinenin bir şeyi *öğrenmesi* tam olarak ne anlama gelmektedir? Wikipedia’nın bir kopyasını indirirsem bilgisayarım gerçekten bir şey öğrenecek mi? Aniden daha akıllı mı oldu? Bu bölüme makine öğrenmesinin ne olduğunu ve neden kullanmak isteyebileceğinizi açıklayarak başlayacağız.

Ardından, makine öğrenmesi kıtasını keşfetmeye başlamadan önce haritaya bir göz atacağız ve ana bölgeler ve en önemli yerler hakkında bilgi edineceğiz: Denetimli ve denetimsiz öğrenmeyi, çevrim içi ve yığın öğrenmeyi, örnek tabanlı ve model tabanlı öğrenmeyi göreceğiz. Ardından, tipik bir makine öğrenmesi projesinin iş akışına bakacağız, karşılaşabileceğiniz ana zorlukları tartışacağız ve bir Makine Öğrenim sisteminin nasıl değerlendirileceğini ve hassas ayarın nasıl yapılacağını ele alacağız.

Bu bölüm, her veri bilimcinin ezbere bilmesi gereken birçok temel kavramı (ve jargonu) içermektedir. Bu, oldukça basit bir tanıtım olacak ancak kitabın geri kalanına devam etmeden önce her şeyin sizin için net olduğundan emin olmalıyız. Öyleyse bir kahve alın ve başlayalım!



Eğer makine öğrenmesi temellerini zaten biliyorsanız doğrudan **Bölüm 2**'ye atlayabilirsiniz. Ama emin değilseniz devam etmeden önce bölümün sonunda listelenen tüm soruları yanıtlamaya çalışın.

## 1.1 Makine Öğrenmesi Nedir?

Makine öğrenmesi, verilerden öğrenen bilgisayarları programlama bilimidir (ve sanatı).

İşte biraz daha genel bir tanım:

[Makine Öğrenmesi], bilgisayarlara açıkça programlama yapmadan öğrenme yeteneği kazandıran bir çalışma alanıdır.

— *Arthur Samuel, 1959*

Ve daha mühendislik odaklı olarıysa:

Belli bir  $T$  görevi ve  $P$  performans ölçütü için eğer  $P$  performans ölçütü,  $T$  görevinde  $E$  tecrübeleriyle artıyorsa bu  $E$  tecrübelerinden öğrendiği söylenen bilgisayar programıdır.

— *Tom Mitchell, 1997*

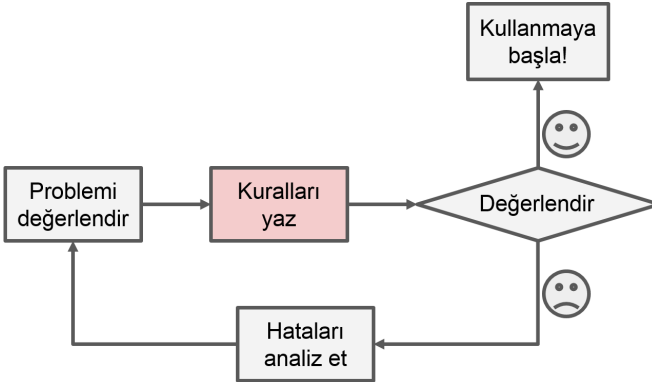
Gereksiz posta filtreniz, gereksiz posta (örneğin kullanıcılar tarafından işaretlenmiş) örnekleri ve normal (“gereksiz olmayan”) posta örnekleri verilip gereksiz posta işaretlemeyi öğrenebilen bir makine öğrenmesi programıdır. Sistemin öğrenmek için kullandığı örnekler *eğitim veri seti* denir. Her bir eğitim örneğine *örnekleme* adı da verilir. Bu durumda  $T$  görevi yeni postaları gerekli/gereksiz olarak işaretlemek,  $E$  tecrübesi *eğitim veri seti* ve  $P$  performans ölçütüyse, örneğin doğru sınıflandırılan posta oranı olarak tanımlanabilir. Bu performans ölçütü *doğruluk oranı* olarak adlandırılır ve genellikle sınıflandırma görevlerinde kullanılır.

Wikipedia'nın bir kopyasını indirirseniz artık bilgisayarınızda çok daha fazla veri bulunacaktır. Ama yaptığı herhangi bir görevde daha iyi değildir. Bu nedenle, Wikipedia'nın bir kopyasını indirmek makine öğrenmesi değildir.

## 1.2 Neden Makine Öğrenmesi Kullanılmalı?

Geleneksel programlama tekniklerini kullanarak nasıl gereksiz posta filtresi yazacağımızı düşünün (Şekil-1.1):

1. Öncelikle gereksiz postanın tipik olarak nasıl görüldüğünü düşünürdünüz. Bazı kelimelerin veya kelime öbeklerinin (“4U”, “kredi kartı”, “ücretsiz” ve “şaşırtıcı” gibi) konu satırında çok fazla ortaya çıkma eğiliminde olduğunu fark edebilirsiniz. Belki de gönderenin adında, postanın gövdesinde ve/veya postanın diğer bölümlerinde başka örüntüler de fark edebilirsiniz.
2. Fark ettiğiniz her bir örüntü için bir tespit algoritması yazarsınız ve bu örüntülerden birkaçı tespit edilirse programınız postaları gereksiz olarak işaretler.
3. Programınızı test eder ve üretim ortamında kullanmaya başlamadan önce yeterince iyi olana kadar 1. ve 2. adımları tekrarlıyorsunuz.



Şekil 1.1: Geleneksel Yaklaşım

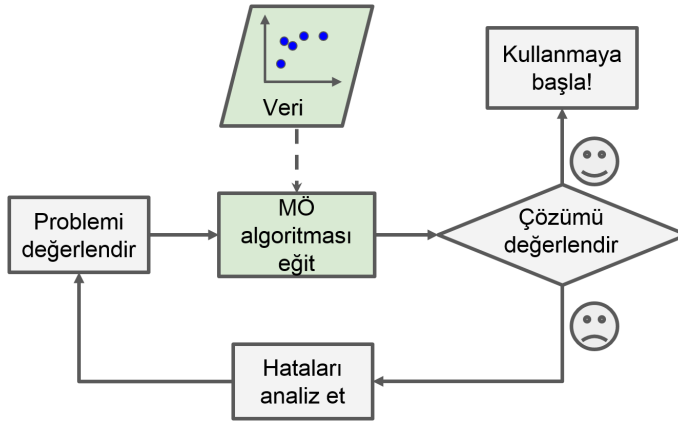
Problem zor olduğundan programınız muhtemelen karmaşık kuralların uzun bir listesi hâline gelecektir ve bakımı oldukça zorlaşacaktır.

Aksine, makine öğrenmesi tekniklerine dayanan bir gereksiz posta filtresi, gerekli posta örneklerine kıyasla gereksiz posta örneklerinde olağan dışı sık kullanılan kelime örüntülerini tespit ederek hangi kelimelerin ve ifadelerin gereksiz posta için iyi tahminci olduğunu otomatik olarak öğrenir (Şekil-1.2). Program çok daha kısa, bakımı daha kolay ve büyük olasılıkla daha doğru sonuçlar getirecektir.

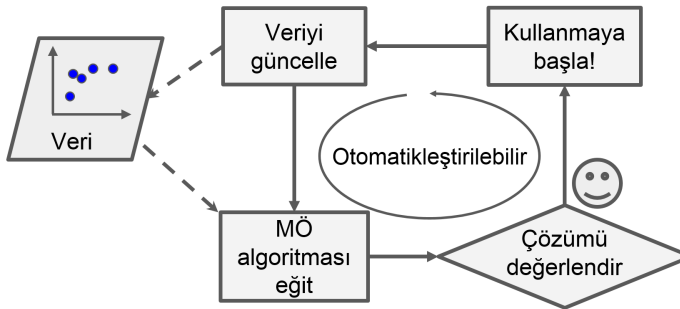
Gereksiz posta gönderenler “4U” içeren tüm postalarının engellendiğini fark ederse ne olur? Bunun yerine “For U” yazmaya başlayabilirler. Geleneksel

programlama tekniklerini kullanan bir gereksiz posta filtresinin “For U” içeren postaları işaretlemek üzere güncellenmesi gerekir. Gereksiz posta gönderenler gereksiz posta filtresi üzerinde çalışmaya devam ederse sonsuza kadar yeni kurallar yazmaya devam etmeniz gerekir.

Buna karşılık makine öğrenmesi tekniklerine dayanan bir gereksiz posta filtresi kullanıcılar için işaretlenen gereksiz postada “For U” nun olağan dışı bir şekilde sık sık geldiğini otomatik olarak fark eder ve sizin müdahaleniz olmadan bunları gereksiz posta olarak işaretlemeye başlar (Şekil-1.3).



Şekil 1.2: Makine öğrenmesi yaklaşımı



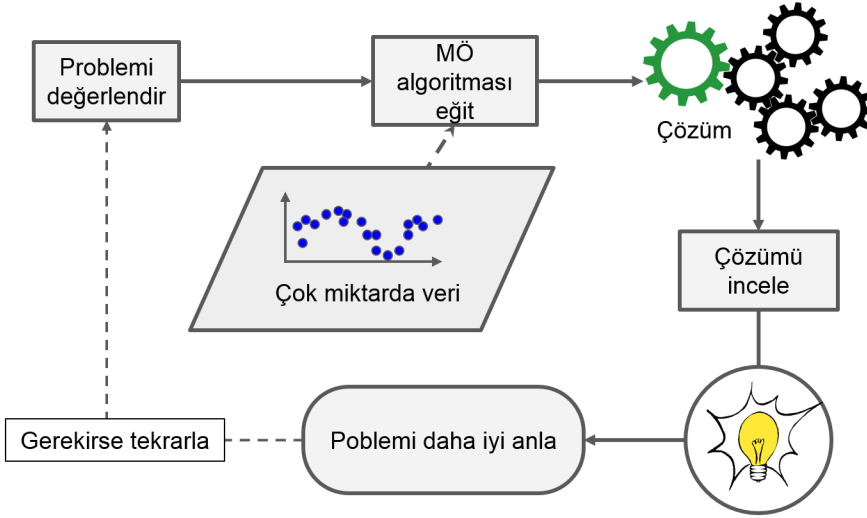
Şekil 1.3: Otomatik olarak değişime uyum sağlama

Makine öğrenmesinin yıldızlaştığı bir başka alan geleneksel yaklaşımlar için çok karmaşık olan ya da bilinen bir çözüm algoritması bulunmayan problemlerdir. Örneğin, bir konuşma tanıma programı düşünün. Basit bir başlangıç yapmak ve “bir” ve “iki” kelimelerini ayırt edebilecek bir program yazmak istediğinizi varsayalım. “İki” kelimesinin yüksek perdeli bir sesle (“İ”) başladığını fark edebilirsiniz, böylece yüksek perdeli ses yoğunluğunu ölçen bir algoritmayı kodlayabilir, birleri ve ikileri ayırt etmek için kullanabilirsiniz. Ama açıkçası



bu teknik gürültülü ortamlarda ve düzinelerce dilde milyonlarca farklı insanın konuştuğu binlerce kelimeye ölçeklenemez. En iyi çözüm (en azından bugün) her kelime için birçok örnek kayıt verildiğinde kendi kendine öğrenen bir algoritma yazmaktır.

Son olarak, makine öğrenmesi insanların öğrenmesine yardımcı olabilir (Şekil 1.4). Makine öğrenmesi algoritmaları ne öğrendiklerini görmek için incelenebilir (gerçi bu bazı algoritmalar için biraz zor olabilir). Örneğin, bir gereksiz posta filtresi yeteri kadar gereksiz posta üzerinde eğitildikten sonra gereksiz postaları en iyi tahmin ettiğine inandığı kelime listesi ve kelime kombinasyonları incelenebilir. Bazen bu; şüphe edilmeyen korelasyonlar ya da yeni eğilimler ortaya çıkartır. Böylece problemin daha iyi anlaşılmasına yol açar. Makine öğrenmesi tekniklerinin büyük miktarda veriye uygulanması kolayca görülmeyen örüntülerin keşfedilmesine yardımcı olabilir. Buna *veri madenciliği* denir.



Şekil 1.4: Makine öğrenmesi, insanların öğrenmesine yardımcı olabilir

Özetlemek gerekirse makine öğrenmesi aşağıda sıralanan konular için harika bir seçimdir:

- Mevcut çözümlerin çok sayıda hassas ayar veya uzun kural listesi gerektirdiği sorunlar: Bir makine öğrenmesi algoritması genellikle kodu basitleştirebilir ve geleneksel yaklaşımdan daha iyi performans gösterebilir.
- Geleneksel bir yaklaşım kullanmanın iyi bir çözüm getirmediği karmaşık sorunlar: En iyi makine öğrenmesi teknikleri belki bir çözüm bulabilir.

## 2.2.2 Performans Ölçütü Seçin

Sonraki adımınız bir performans ölçütü seçmektir. Bağlanım problemleri için tipik bir performans ölçütü Ortalama Karesel Hata Kareköküdür (RMSE). Sistem tahminlerini yaptığında büyük hatalara daha büyük ağırlıklar vererek ne kadar hata yaptığı hakkında bir fikir verir. **Denklem-2.1** matematiksel olarak RMSE'nin nasıl hesaplanacağını göstermektedir.

Ortalama Karesel Hata Karekökü (RMSE)

$$\text{RMSE}(\mathbf{X}, h) = \sqrt{\frac{1}{m} \sum_{i=1}^m (h(\mathbf{x}^{(i)}) - y^{(i)})^2} \quad (2.1)$$

### Notasyon

Bu denklem bu kitap boyunca kullanacağımız birçok makine öğrenmesi notasyonunu içermektedir.

- $m$ , üzerinde RMSE hesabını yapacağınız veri setindeki örnek sayısıdır.
  - Örneğin eğer doğrulama veri setinde 2000 bölge için RMSE hesaplıyorsanız  $m = 2000$ 'dir.
- $\mathbf{x}^{(i)}$  veri setindeki  $i^{\text{inci}}$  örneğin tüm niteliklerinin (etiketi hariç) değerlerini içeren bir vektördür ve  $y^{(i)}$  ise o örneğin etiketidir (o örnek için istenen çıktı değeri).
  - Örneğin ilk bölge  $-118.29^\circ$ enleminde,  $33.91^\circ$ boylamında ve gelirlerinin medyanı  $38372\$$  olan  $1416$  nüfusa ve medyan konut fiyatı  $156400\$$  (şimdilik diğer nitelikleri atlıyoruz) olsun:

$$\mathbf{x}^{(1)} = \begin{pmatrix} -118.29 \\ 33.91 \\ 1416 \\ 38372 \end{pmatrix}$$

ve:

$$y^{(1)} = 156400$$

- $\mathbf{X}$  veri setindeki tüm örneklerin tüm niteliklerini (etiketleri hariç) içeren bir matristir. Her satır bir örneğe ait ve  $i^{\text{inci}}$  satır  $\mathbf{x}^{(i)}$

vektörünün transpozuna, yani  $(\mathbf{x}^{(i)})^\top$  ifadesine eşittir.<sup>4</sup>

- Örneğin ilk bölgenin az önce tanımladığımız gibi olduğunu düşünürsek  $\mathbf{X}$  matrisi şuna benzer:

$$\mathbf{X} = \begin{pmatrix} (\mathbf{x}^{(1)})^\top \\ (\mathbf{x}^{(2)})^\top \\ \vdots \\ (\mathbf{x}^{(1999)})^\top \\ (\mathbf{x}^{(2000)})^\top \end{pmatrix} = \begin{pmatrix} -118.29 & 33.91 & 1416 & 38372 \\ \vdots & \vdots & \vdots & \vdots \end{pmatrix}$$

- $h$ , hipotez de denilen sisteminizin tahmin fonksiyonudur. Sisteminize bir örneğe ait nitelik vektörü  $\mathbf{x}^{(i)}$  verildiğinde, çıktısı o örneğe ait tahmin  $\hat{y}^{(i)} = h(\mathbf{x}^{(i)})$  edilen değerdir  $\hat{y}$ .
  - Örneğin eğer sisteminiz ilk bölge için median konut fiyatını 158400\$ olarak tahmin ederse  $\hat{y}^{(1)} = h(\mathbf{x}^{(1)}) = 158400\$$ 'dir. O bölge için tahmin hatası ise  $\hat{y}^{(1)} - y^{(1)} = 2000$ 'dir.
- RMSE( $\mathbf{X}, h$ )  $h$  hipotezinin bir örnekler seti için ölçülen maliyet fonksiyonudur.

Skaler değerleri ( $m$  ya da  $y^{(i)}$  gibi) ve fonksiyonları isimlendirirken ( $h$  gibi) küçük yatık harfler, vektörleri ( $\mathbf{x}^{(i)}$  gibi) küçük kalın harfler ve matrisleri ( $\mathbf{X}$  gibi) büyük kalın harflerle gösteriyoruz.

Her ne kadar RMSE bağlanım görevlerinde genellikle tercih edilen bir performans ölçütü olsa da bazen farklı fonksiyonlar da tercih edebilirsiniz. Örneğin çok fazla bölge için aykırı değerlerin olduğunu düşünün. Bu durumda *ortalama mutlak hata* kullanmayı düşünebilirsiniz (MAE, ortalama mutlak değişim olarak da adlandırılır; [Denklem-2.2](#) bakınız):

Ortalama mutlak hata (MAE)

$$\text{MAE}(\mathbf{X}, h) = \frac{1}{m} \sum_{i=1}^m \left| h(\mathbf{x}^{(i)}) - y^{(i)} \right| \quad (2.2)$$

Hem RMSE hem de MAE iki vektör arasındaki mesafeyi ölçmenin bir yoludur: Tahmin vektörü ve hedef etiket vektörü. Farklı mesafe ölçütleri ya da *normları* kullanmak mümkündür:

- Karelerin toplamının karekökünü (RMSE) hesaplamak şu anda aşına olduğunuz *Öklid normu* (ÇN: Euclidean norm) hesaplamaya karşılık gelir.

<sup>4</sup>Transpoz operatörü sütun vektörünü satır vektörüne dönüştürür (ve tam tersine).

Aynı zamanda  $\ell_2$  norm olarak adlandırılır ve  $\|\cdot\|_2$  ile de gösterilir. (Ya da sadece  $\|\cdot\|$ ).

- Mutlak değerlerin toplamını (MAE) hesaplamak  $\ell_1$  normuna karşılık gelir ve  $\|\cdot\|_1$  ile gösterilir. Bazen Manhattan normu olarak da adlandırılır. Çünkü bir şehirdeki iki nokta arasındaki mesafeyi sadece ortogonal bloklar kullanarak ölçer.
- Daha genel olarak  $n$  tane eleman içeren  $\mathbf{v}$ 'nin  $\ell_k$  normu vektörü  $\|\mathbf{v}\|_k = \left(|v_0|^k + |v_1|^k + \dots + |v_n|^k\right)^{1/k}$  olarak tanımlanabilir.  $\ell_0$  vektörde sıfırdan farklı elemanları gösterirken,  $\ell_\infty$  en büyük mutlak değeri gösterir.
- Büyük norm indeksleri küçük değerleri yok sayarken büyük değerlere odaklanır. Bu yüzden, RMSE aykırı değerlerden MAE'ye göre daha fazla etkilenir. Ama aykırı değerler üssel olarak seyrek olduğunda (çan eğrisinde olduğu gibi) RMSE çok başarılı olur ve tercih edilir.

### 2.2.3 Varsayımları Kontrol Edin

Son olarak; şu ana kadar yapılan varsayımları (sizin tarafınızdan ya da başkaları tarafından) listelemek ve doğrulamak iyi olacaktır ki bu size yapacağınız ciddi hataları önceden görmeyi yardımcı olacaktır. Örneğin, sisteminizin çıktısı olacak ve başka bir makine öğrenmesi sistemine girdi olacak olan bölge fiyatlarının sizin ürettiğiniz şekilde kullanılacağını varsaydınız. Ama diğer sistem ürettiğiniz fiyatları kullanmak yerine kategorilere (mesela “ucuz”, “ortalama” ya da “pahalı”) dönüştürüp kullanacaksa ne olacak? Bu durumda fiyatları mükemmel bir şekilde tahmin etmek çok da önemli olmayacak onun yerine sadece doğru kategoriye tahmin etmesi gerekecek. Eğer öyleyse problem bağlanım görevi yerine sınıflandırma görevi olarak düşünülmeliydi. Bunu bir bağlanım görevi olarak düşünüp aylarca uğraştıktan sonra fark etmek istemezsiniz.

Neyse ki diğer sistemden sorumlu olan takımla yaptığımız görüşme sonucunda sadece kategorilere değil gerçek fiyatlara da ihtiyaçları olduğuna emin oldunuz. Çok güzel! Her şeyi hazırladınız, yeşil ışık yandı ve kodlamaya başlayabilirsiniz!

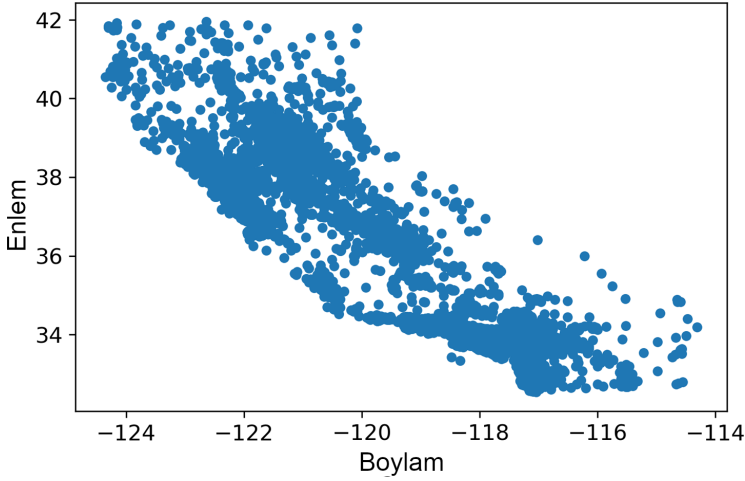
## 2.3 Verileri Toplayın

Ellerinizi kirletme zamamı geldi. Dizüstü bilgisayarınızı alıp Jupyter not defterinde bulunan kod örneklerinin üzerinden geçmekten çekinmeyin. Tüm Jupyter not defteri örneklerini <https://github.com/ageron/handson-ml2> bağlantısında bulabilirsiniz.

## Coğrafi Veriyi Görselleştirme

Enlem ve boylam gibi coğrafi bilgiler bulunduğundan bölge verilerini dağılım grafiği kullanarak görselleştirmek iyi bir fikir olacaktır (Şekil-2.11'e bakınız):

```
housing.plot(kind="scatter", x="enlem", y="boylam")
```



Şekil 2.11: Verinin coğrafi dağılım grafiği

Evet grafik Kaliforniya'ya benziyor ama ondan başka bir örüntü görülemiyor. `alpha` opsiyonunu 0.1 yapmak verinin yoğun olduğu noktaları daha iyi görselleştirir (Şekil-2.12):

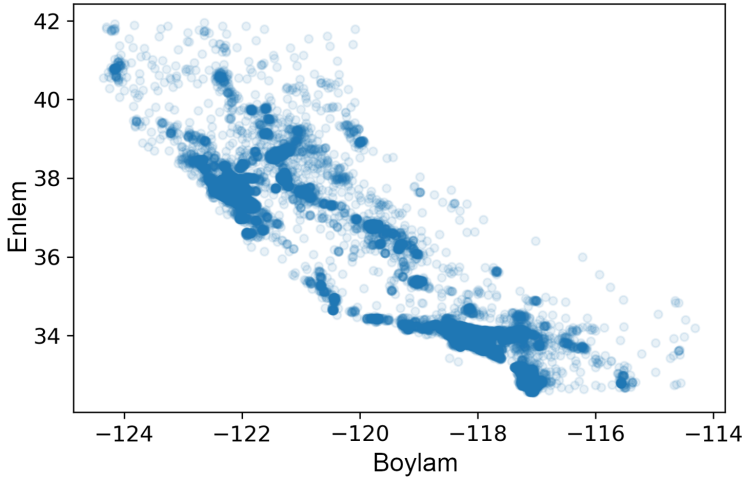
```
housing.plot(kind="scatter", x="enlem", y="boylam", alpha=0.1)
```

Şimdi daha iyi oldu: Yüksek yoğunluklu alanları görüyorsunuz. Los Angeles ve San Diego yakınlarında bulunan Bay Area, Sacramento ve Fresno yakınlarında uzanan Central Valley alanlarında yoğunluk göze çarıyor.

Beynimiz resimlerdeki örüntüleri bulma konusunda çok iyidir ama görselleştirme parametreleriyle biraz daha oynayarak örüntüleri ortaya çıkarmanız gerekecektir.

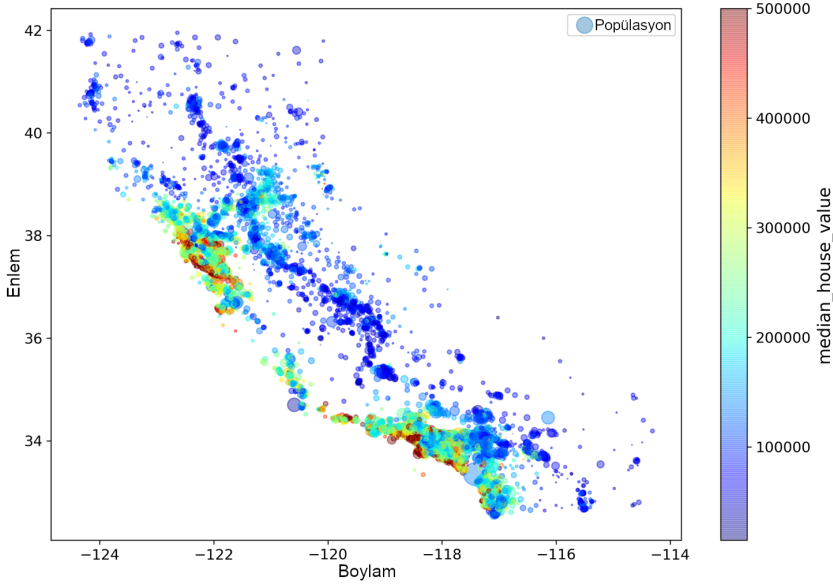
Şimdi de ev fiyatlarına bakalım (Şekil-2.13). Her dairenin yarıçapı nüfusu (`s` opsiyonu) renklerle fiyatları (`c` opsiyonu) gösteriyor. Maviden (düşük fiyatlar) kırmızıya (yüksek fiyatlar) değişen jet isimli, önceden tanımlı bir renk haritası (`cmap` opsiyonu) kullanacağız.<sup>16</sup>

<sup>16</sup>Bunu gri tonlamalı olarak okuyorsanız, kırmızı bir kalem alın ve Körfez Bölgesi'nden San Diego'ya kadar sahil şeridinin büyük bir kısmını karalayın (beklediğiniz gibi). Sacramento'nun etrafına da sarı bir yama ekleyebilirsiniz.



Şekil 2.12: Yüksek yoğunluklu bölgeleri daha iyi görselleştirme

```
housing.plot(kind="scatter", x="enlem", y="boylam", alpha=0.4,
             s=housing["population"]/100, label="nüfus", figsize=(10,7),
             c="median_house_value", cmap=plt.get_cmap("jet"), colorbar=True,
             )
plt.legend()
```



Şekil 2.13: Kaliforniya ev fiyatları: Kırmızı pahalı, mavi ucuz fiyatları, geniş daireler nüfus yoğun bölgeleri belirtmektedir.

öznitelik sayısına göre doğrusaldır. Diğer bir deyişle iki katı örnek üzerinde (ya da iki katı öznitelik üzerinde) tahmin yapmak iki katı zaman alacaktır.

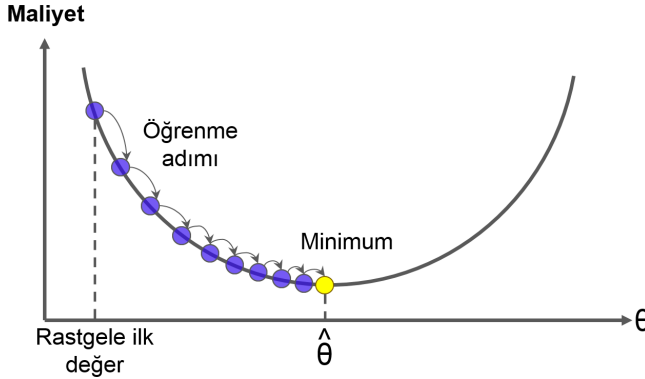
Şimdi doğrusal bağlanım modelini eğitmenin çok fazla sayıda öznitelik bulunduğu ya da hafızaya sığmayacak kadar fazla örnek bulunduğu çok kullanışlı olan oldukça farklı bir yöntemine bakacağız.

## 4.2 Gradyan İnişi

Gradyan inişi çok farklı problemler için optimum sonucu bulabilme yeteneğine sahip bir eniyileme algoritmasıdır. Gradyan inişinin ana fikri parametreleri her iterasyonda ayarlayarak maliyet fonksiyonunu enküçültmektir.

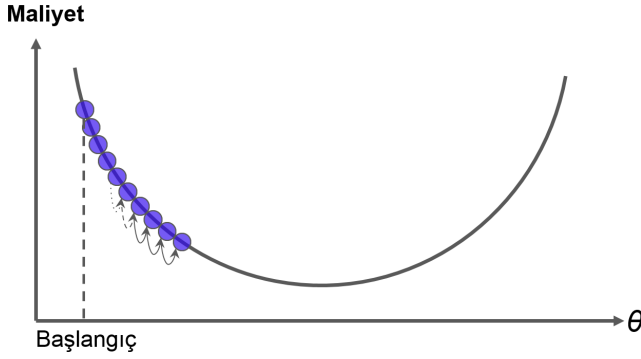
Bir dağda yoğun siste kaybolduğunuzu ve sadece ayağınızı bastığınız yerin eğimini hissettiğinizi düşünün. Vadiye hızla ulaşmanın iyi bir yolu en dik eğimi aşağı doğru takip etmektir. Bu tam olarak gradyan inişinin yaptığı şeydir: Hata fonksiyonunun  $\theta$  parametre vektörüne göre gradyanını hesaplar ve azalan gradyan yönünde ilerler. Gradyan sıfır olduğunda, minimum seviyeye ulaştınız!

Somut olarak  $\theta$  vektörünü rastgele değerlerle doldurarak (buna rastgele *ilk değer atama* denir) başlarız. Sonra yavaş yavaş onu geliştirirsiniz, her seferince küçük bir adım atıp, her adımda maliyet fonksiyonu (örneğin MSE) değerini azaltıp, bir minimuma yakınsayınca kadar ilerlersiniz (Şekil-4.3'e bakınız).



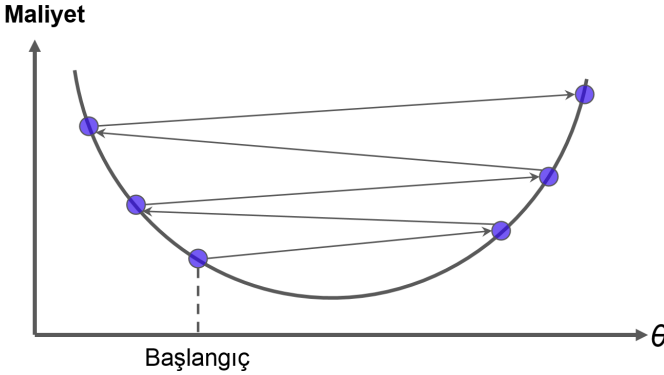
**Şekil 4.3:** Bu gradyan inişi tasvirinde, model parametrelerinin rastgele olarak ilk ataması yapılır ve maliyet fonksiyonunu enküçültmek için defalarca ayarlanır; öğrenme adımının büyüklüğü, maliyet fonksiyonunun eğimiyle doğru orantılıdır. Dolayısıyla adımlar, parametreler minimuma yaklaştıkça yavaş yavaş küçülür.

Gradyan inişinin önemli bir parametresi öğrenme oranı hiperparametresiyle kontrol edilen adım büyüklüğüdür. Eğer öğrenme oranı çok küçükse algoritma yakınsayana kadar çok fazla iterasyon yapacak ve bu çok zaman alacaktır (Şekil-4.4'e bakınız).



Şekil 4.4: Öğrenme oranı çok küçük

Diğer taraftan öğrenme oranı çok büyükse vadiden karşıya atlayıp, bulunduğunuz yerden daha yüksek bir noktaya ulaşabilirsiniz. Bu durum algoritmanın iraksamasına sebep olur ve daha büyük değerlere doğru yönlendirir ve iyi bir çözüm bulamayabilir (Şekil-4.5'e bakınız).

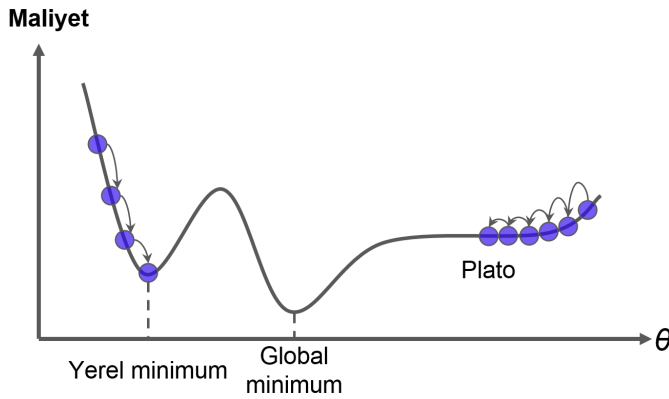


Şekil 4.5: Öğrenme oranı çok büyük

Son olarak, tüm maliyet fonksiyonları düzgün biçimli bir çanak şekline sahip değildir. Delikler, sırtlar, platolar ve diğer düzensiz yüzeyler bulunabilir ve minimuma yakınsamayı zorlaştırabilir. Şekil-4.6 gradyan inişinin karşılaştığı iki ana zorluğu göstermektedir. Eğer rastgele ilk değer atama algoritmayı solda başlatırsa yerel minimuma ulaşır ki global minimum kadar iyi değildir. Eğer sağda başlarsa düzlüğü geçmesi çok zaman alır. Çok erken durdurursanız, global minimuma asla ulaşamazsınız.

Neyse ki doğrusal bağlanım modeli için MSE maliyet fonksiyonu konveks fonksiyondur. Bu, eğri üzerinde seçeceğimiz iki noktadan geçen bir doğru parçasının eğriyi asla kesmeyeceği anlamına gelir. Bu da yerel minimum olmadığını, sadece

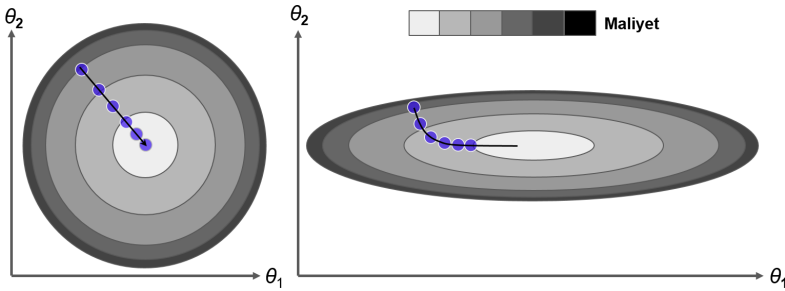




Şekil 4.6: Gradyan inişi tuzağı

tek olan global minimum olduğunu işaret eder. Aynı zamanda eğimi birden değişmeyen sürekli bir fonksiyondur.<sup>3</sup> Bu iki gerçeğin bir güzel sonucu vardır: Gradyan inişi global minimum değerine yeterince yaklaşacağını garanti eder (yeterince uzun beklerseniz ve öğrenme oranı çok büyük değilse).

Aslında maliyet fonksiyonunun şekli çanak şeklindedir ancak eğer nitelikler farklı ölçeklerdeyse yayvan bir çanak olabilir. Şekil-4.7 eğitim veri setindeki nitelik 1 ve 2'nin aynı ölçüğe geldiğindeki gradyan inişini (soldaki) ve nitelik 1'in nitelik 2'ye göre daha küçük değerler aldığı durumdaki gradyan inişini (sağdaki) gösteriyor.<sup>4</sup>



Şekil 4.7: Öznitelik ölçeklemesi olan (sol) ve olmayan (sağ) gradyan inişi

Gördüğünüz gibi soldaki gradyan inişi algoritması doğrudan minimuma gidiyor ve bu minimuma hızlıca ulaşıyor. Ama sağdaki önce global minimum istikametine hemen hemen ortogonal yönde gidiyor ve düz vadide uzun bir yol katediyor. Sonunda minimuma ulaşacaktır ancak bu uzun zaman alacaktır.

<sup>3</sup>Teknik olarak türevi *Lipschitz süreklisidir*.

<sup>4</sup>Nitelik 1 daha küçük değer aldığından, maliyet fonksiyonunu etkileyebilmek için  $\theta_1$ 'de daha büyük bir değişim gereklidir. Bu nedenle, çanak  $\theta_1$  eksenini boyunca uzamıştır.



Gradyan inişini kullanırken özniteliklerin aynı ölçekte olduğuna emin olmalısınız (örneğin Scikit-Learn'in `StandardScaler` sınıfını kullanarak) yoksa yakınsaması uzun zaman alacaktır.

Bu diyagram aynı zamanda model eğitmenin maliyet fonksiyonu enküçülten (eğitim veri setinde) parametre kombinasyonu arama anlamına geldiğini gösteriyor. Bu arama, modelin *parametre uzayında* yapılır: Model parametre sayısı arttıkça, bu uzayın boyutları çok artar ve arama zorlaşır: 300-boyutlu samanlıkta iğne aramak 3 boyutta aramaktan daha zordur. Neyse ki doğrusal bağlanımda maliyet fonksiyonu konvektir ve iğne de çanağın dibindedir.

### 4.2.1 Yığın Gradyan İnişi

Gradyan inişini gerçekleştirmek için modelin her parametresi  $\theta_j$  için maliyet fonksiyonunun gradyanını hesaplamalıyız. Diğer bir deyişle  $\theta_j$  parametresini bir miktar değiştirdiğinizde maliyet fonksiyonunun ne kadar değişeceğini hesaplamaya ihtiyacınız vardır. Bu, kısmi türev olarak adlandırılır. Şöyle sormaya benzer; “Eğer yüzümü batıya dönersem dağın ayaklığının altında kalan kısmının eğimi nedir?” ve aynı soruyu kuzey için de sorarsınız (ve bunu her yön için tekrar edersiniz). **Denklem 4.5** maliyet fonksiyonunun  $\theta_j$  parametresine göre kısmi türevini hesaplar ve  $\partial \text{MSE}(\theta) / \partial \theta_j$  ile gösterilir.

Maliyet fonksiyonunun kısmi türevi

$$\frac{\partial}{\partial \theta_j} \text{MSE}(\theta) = \frac{2}{m} \sum_{i=1}^m (\theta^\top \mathbf{x}^{(i)} - y^{(i)}) x_j^{(i)} \quad (4.5)$$

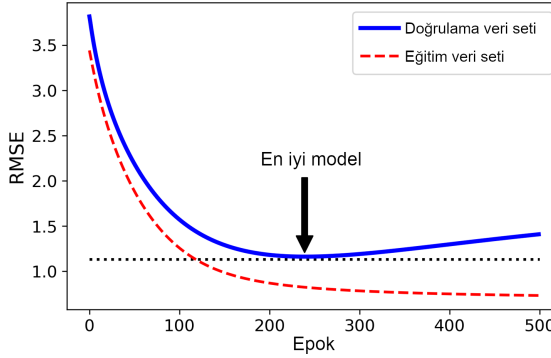
Tek tek hesaplamak yerine **Denklem 4.6**'daki vektörü kullanarak kısmi türevleri bir seferde de hesaplayabilirsiniz.  $\nabla_{\theta} \text{MSE}(\theta)$  ile gösterilen gradyan vektörü, maliyet fonksiyonunun tüm kısmi türevlerini içerir (her biri, model parametresi için olacak biçimde).

Maliyet fonksiyonunun gradyan vektörü

$$\nabla_{\theta} \text{MSE}(\theta) = \begin{pmatrix} \frac{\partial}{\partial \theta_0} \text{MSE}(\theta) \\ \frac{\partial}{\partial \theta_1} \text{MSE}(\theta) \\ \vdots \\ \frac{\partial}{\partial \theta_n} \text{MSE}(\theta) \end{pmatrix} = \frac{2}{m} \mathbf{X}^\top (\mathbf{X}\theta - \mathbf{y}) \quad (4.6)$$

#### 4.5.4 Erken Durdurma

Gradyan inişi gibi iterasyonla çalışan öğrenme algoritmalarını düzenleştirmenin çok farklı bir yoluysa doğrulama hatası en küçük değere ulaşır ulaşmaz eğitimi durdurmaktır. Bu, *erken durdurma* olarak adlandırılır. Şekil-4.20 yığın gradyan inişiyle eğitilen karmaşık bir modeli (yüksek dereceden polinom bağlanım modeli) göstermektedir. Epoklar ilerledikçe ve algoritma öğrendikçe eğitim ve doğrulama veri setlerinde tahmin hatası (RMSE) birlikte azalır. Bir süre sonra doğrulama hatasının azalması durur ve tekrar yükselişe geçer. Bu, modelin eğitim veri setine aşırı uydurmaya başladığına işaret eder. Erken durdurmada doğrulama hatası en küçük değere iner inmez eğitimi durdurursunuz. Bu, Geoffrey Hinton'ın "bedava öğlen yemeği" dediği, basit ve etkili bir düzenleştirme tekniğidir.



Şekil 4.20: Erken durdurma düzenleştirmesi



Stokastik ve mini-yığın gradyan inişi kullanırken eğriler bu kadar düzgün olmaz ve bazen minimuma ulaşır ulaşmadığımızı anlamak zor olabilir. Çözümlerden biri, doğrulama hatasının minimum civarında bir süre geçirmesi hâlinde durmak (modelin daha iyisini yapamayacağına emin olduğunuzda) ve model parametrelerini doğrulama hatasının en küçük olduğu duruma döndürmektir.

Aşağıda erken durdurmanın basit bir uygulamasını bulabilirsiniz:

```
from copy import deepcopy
# verileri hazırlama
poly_scaler = Pipeline([
    ("poly_features", PolynomialFeatures(degree=90, include_bias=False)),
    ("std_scaler", StandardScaler())
])
```

```

X_train_poly_scaled = poly_scaler.fit_transform(X_train)
X_val_poly_scaled = poly_scaler.transform(X_val)

sgd_reg = SGDRegressor(max_iter=1, tol=-np.infty, warm_start=True,
                       penalty=None, learning_rate="constant", eta0=0.0005)

minimum_val_error = float("inf")
best_epoch = None
best_model = None

for epoch in range(1000):
    sgd_reg.fit(X_train_poly_scaled, y_train) # kaldığı yerden devam ediyor
    y_val_predict = sgd_reg.predict(X_val_poly_scaled)
    val_error = mean_squared_error(y_val, y_val_predict)
    if val_error < minimum_val_error:
        minimum_val_error = val_error
        best_epoch = epoch
        best_model = deepcopy(sgd_reg)

```

`warm_start=True` seçilip `fit()` metodu çağrıldığında eğitimin sıfırdan başlamak yerine kaldığı yerden devam ettiğini unutmayın.

## 4.6 Lojistik Bağlanım

**Bölüm 1**'de aktardığımız gibi bazı bağlanım algoritmaları sınıflandırma için kullanılabilir (tersi de geçerlidir). *Lojistik bağlanım* genellikle örneklerin bir sınıfa ait olma olasılığını tahmin etmek için kullanılır (örneğin, bir e-postanın gereksiz posta olma olasılığı nedir?). Eğer tahmin edilen olasılık %50'den fazlaysa örnek o sınıfa aittir (*pozitif sınıf* denir ve etiketi "1"dir), değilse o sınıfa ait değildir (*negatif sınıf* aittir ve etiketi "0"dir). Bu, onu ikili sınıflandırıcı yapar.

### 4.6.1 Olasılık Tahmini

Lojistik bağlanım nasıl çalışır? Doğrusal bağlanım modelinde olduğu gibi lojistik bağlanım modeli de girdi niteliklerinin ağırlıklı toplamını (artı yanlılık terimi) hesaplar. Ama doğrusal bağlanım modelinin yaptığı gibi doğrudan bir çıktı vermek yerine sonucun *lojistiğini* verir (**Denklem 4.13**'e bakınız).

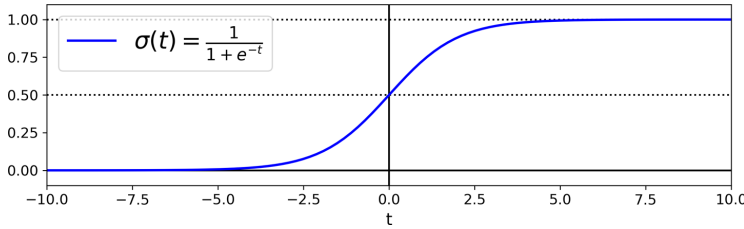
Lojistik — $\sigma(\cdot)$  ile gösterilen— sigmoid fonksiyonudur (*S*-şeklinde) ve 0 ile 1 arasında çıktılar verir. **Denklem 4.14** ve **Şekil-4.21**'de gösterildiği gibi tanımlanır.

Lojistik bağlanım modeli olasılık tahmini (vektörel form)

$$\hat{p} = h_{\theta}(\mathbf{x}) = \sigma(\mathbf{x}^T \theta) \quad (4.13)$$

Lojistik fonksiyon

$$\sigma(t) = \frac{1}{1 + \exp(-t)} \quad (4.14)$$



Şekil 4.21: Lojistik fonksiyon

Lojistik bağlanım modeli,  $\mathbf{x}$  örneğinin pozitif sınıfa ait olma olasılığı olan  $(\hat{p}) = h_{\theta}(\mathbf{x})$ 'i hesapladığında o örneğe ait  $\hat{y}$  tahminini kolaylıkla yapabilir (Denklem 4.15'e bakınız).

Lojistik bağlanım modelinin tahmini

$$\hat{y} = \begin{cases} 0, & \hat{p} < 0.5 \\ 1, & \hat{p} \geq 0.5 \end{cases} \quad (4.15)$$

$t < 0$  olduğunda  $\sigma(t) < 0.5$  olduğuna ve  $t \geq 0$  olduğunda  $\sigma(t) \geq 0.5$  olduğuna dikkat edin. Sonuç olarak lojistik bağlanım modeli  $\mathbf{x}^T \theta$  pozitif olduğunda 1 negatif olduğunda 0 tahmin eder.



Bu  $t$  skoru, lojit olarak da adlandırılır. İsmi, lojit( $p$ ) =  $\log(p / (1 - p))$  ile tanımlanan lojit fonksiyonundan gelir. Lojit fonksiyonu, lojistik fonksiyonun tersidir. Tahmin edilen olasılık  $p$ 'nin lojitini hesapladığımızda sonucun  $t$  olduğunu bulacaksınız. Pozitif sınıflar için tahmin edilen olasılık ile negatif sınıflar için tahmin edilen olasılık arasındaki oranın logaritması olduğu için lojit aynı zamanda *odds oranının logaritması* olarak da bilinir.

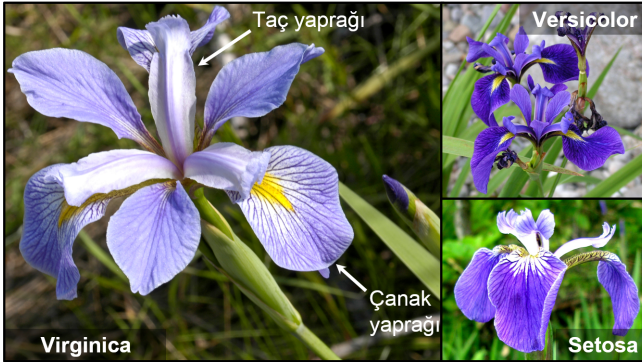
Lojistik maliyet fonksiyonunun kısmi türevleri

$$\frac{\partial}{\partial \theta_j} J(\theta) = \frac{1}{m} \sum_{i=1}^m \left( \sigma(\theta^\top \mathbf{x}^{(i)}) - y^{(i)} \right) x_j^{(i)} \quad (4.18)$$

Stokastik gradyan inişinde her seferinde bir örnek ve mini-yığın gradyan inişinde her seferinde bir mini-yığın alırsınız.

### 4.6.3 Karar Sınırları

Iris veri setini kullanarak lojistik bağlanımı görselleştirelim. Üç farklı türden 150 süsen çiçeğinin çanak ve taç yapraklarının genişliği ve uzunluğunu içeren çok ünlü bir veri setidir: *Iris setosa*, *Iris versicolor* ve *Iris virginica* (Şekil-4.22'ye bakınız<sup>14</sup>).



Şekil 4.22: Üç farklı süsen bitkisinin çiçekleri

Şimdi *Iris virginica* türünü sadece taç yaprağı genişliğine bakarak tespit eden bir sınıflandırıcı oluşturalım. Önce veri setini yükleyelim:

```
>>> from sklearn import datasets
>>> iris = datasets.load_iris()
>>> list(iris.keys())
['data', 'target', 'target_names', 'DESCR', 'feature_names', 'filename']
>>> X = iris["data"][:, 3:] # taç yaprağı genişliği
>>> y = (iris["target"] == 2).astype(np.int) # eğer Iris virginica ise 1 değilse 0
```

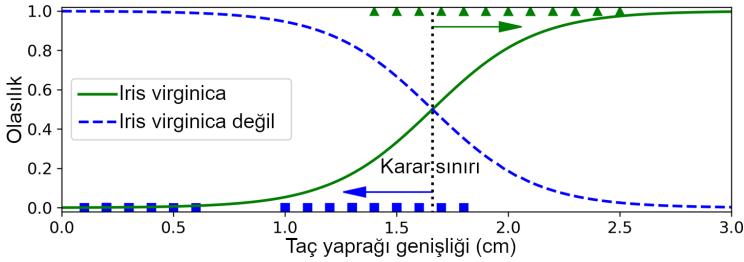
<sup>14</sup>Resimler ilgili Wikipedia sayfaları kullanılarak yeniden üretilmiştir. *Iris virginica* resmi, Frank Mayfield (Creative Commons BY-SA 2.0 (<https://creativecommons.org/licenses/by-sa/2.0/>)), *Iris versicolor* resmi D. Gordon E. Robertson (Creative Commons BY-SA 3.0 (<https://creativecommons.org/licenses/by-sa/3.0/>)), *Iris setosa* resmi açık kaynaklı.

Şimdi de bir lojistik bağlanım modeli eğitelim:

```
from sklearn.linear_model import LogisticRegression
log_reg = LogisticRegression()
log_reg.fit(X, y)
```

Genişliği 0 cm'den 3 cm'ye değişen taç yaprağa sahip çiçekler için modelin tahmin ettiği olasılıklara bakalım (Şekil-4.23):<sup>15</sup>

```
X_new = np.linspace(0, 3, 1000).reshape(-1, 1)
y_proba = log_reg.predict_proba(X_new)
plt.plot(X_new, y_proba[:, 1], "g-", label="Iris virginica")
plt.plot(X_new, y_proba[:, 0], "b--", label="Not Iris virginica")
# + güzel görkmesi için daha fazla Matplotlib kodu
```



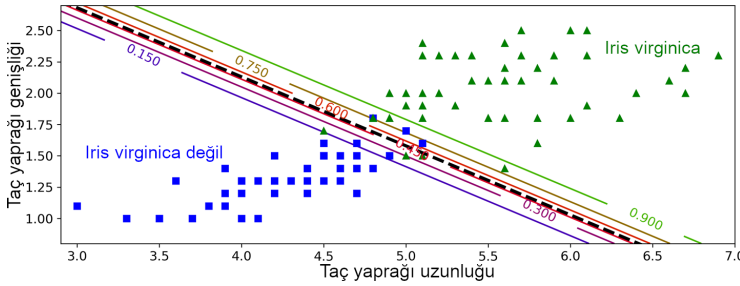
Şekil 4.23: Tahmin edilen olasılıklar ve karar sınırı

*Iris virginica* çiçeğinin taç yapraklarının genişliği (üçgenle gösterilen) 1.4 cm ile 2.5 cm aralığındadır, diğer süsen çiçeklerinin taç yaprağı genişliği ise (kareyle gösterilenler) daha küçüktür 0.1 cm ile 1.8 cm aralığındadır. Bir miktar üst üste geldiğine dikkat edin. Taç yaprağı genişliği 2 cm'nin üstüne çıktığında sınıflandırıcı çiçeğin *Iris virginica* (o sınıf için yüksek olasılık değerleri veriyor) olduğuna oldukça eminken, 1 cm'in altına indiğinde *Iris virginica* olmadığına oldukça emindir. Bu iki ucun haricinde sınıflandırıcı kararsız kalıyor ancak `predict` sınıfını (`predict_proba()` metodu yerine `predict()` metodu) kullanarak bir sınıf tahmin etmesini isterseniz, en olası sınıfı getirecektir. Dolayısıyla 1.6 cm civarında olasılık değerlerinin %50-%50 eşit olduğu bir karar sınırı bulunuyor. Eğer taç yaprağı genişliği 1.6 cm'den büyükse sınıflandırıcı çiçeği *Iris virginica* olarak sınıflandıracak aksi hâlde de *Iris virginica* olmadığını tahmin edecek (her ne kadar çok emin olmasa da):

```
>>> log_reg.predict([[1.7], [1.5]])
array([1, 0])
```

<sup>15</sup>NumPy'nin `reshape()` fonksiyonu bir boyutun -1 olmasına izin verir, bu "tanımlanmamış" anlamındadır: Bu değer dizinin uzunluğundan ve geri kalan boyutlardan elde edilir.

**Şekil-4.24** aynı veri setini ancak bu kez iki niteliğini gösteriyor: Taç yaprak uzunluğu ve genişliği. Lojistik bağlanım sınıflandırıcı eğitildikten sonra bu iki niteliği kullanarak bir çiçeğin *Iris virginica* olup olmadığını tahmin edebilir. Kesikli çizgi modelin %50 olasılığı tahmin ettiği noktaları göstermektedir: Bu da modelin karar sınırınıdır. Doğrusal bir sınır olduğuna dikkat edin.<sup>16</sup> Her paralel doğru modelin %15'ten (sol alt) %90'a (sağ üst) farklı olasılık değeri verdiği noktaları temsil etmektedir. Modele göre sağ üstteki doğrunun ötesinde bulunan bir çiçek %90'dan fazla ihtimalle *Iris virginica* olarak sınıflandırılacaktır.



**Şekil 4.24:** Doğrusal karar sınırı

Diğer doğrusal modeller gibi lojistik bağlanım modeli  $\ell_1$  ya da  $\ell_2$  kullanarak düzenlenebilir. Scikit-Learn varsayılan olarak  $\ell_2$  kullanır.



Scikit-Learn `LogisticRegression` modeli için düzenleme seviyesinin kontrolü `alpha` (diğer doğrusal modeller de olduğu gibi) parametresi yerine bu parametrelerin tersini kullanarak yapılır: `C`. `C`'nin değeri yükseldikçe model daha az düzenlenir.

#### 4.6.4 Softmax Bağlanım

Lojistik bağlanım modeli çok sınıflılığı destekleyecek şekilde birden fazla ikili sınıflandırıcı eğitmeye gerek olmaksızın doğrudan genelleştirilebilir. (Bölüm 3'te anlatıldığı gibi). Bu, *softmax bağlanım* ya da *çok terimli lojistik bağlanım* olarak adlandırılır.

Fikir çok basit: Verilen bir  $\mathbf{x}$  örneği için softmax bağlanım modeli önce her  $k$  sınıfı için bir skor  $s_k(\mathbf{x})$  hesaplar, sonra da bu skorlara *softmax fonksiyonunu* (normalleştirilmiş üstel fonksiyon da denir) uygulayarak her sınıfın olasılığını tahmin eder.  $s_k(\mathbf{x})$ 'yı hesaplamak için kullanılan denklem doğrusal bağlanımın tahminlerini yaptığı denkleme benzetilebilir (Denklem 4.19'a bakınız).

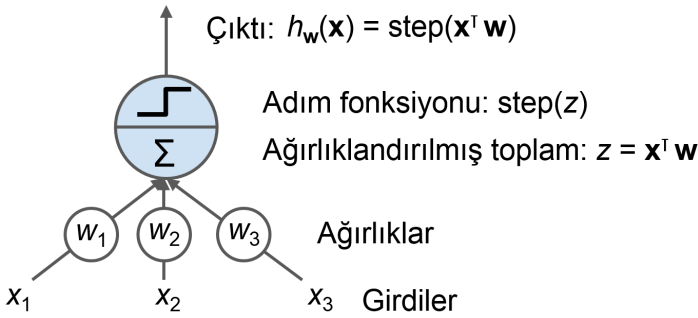
<sup>16</sup> $\mathbf{x}$  nokta kümesi  $\theta_0 + \theta_1 x_1 + \theta_2 x_2 = 0$  ile tanımlanır ve düz bir doğrudur.



Bu sinir ağlarının karmaşık mantıksal ifadeleri hesaplamak üzere nasıl bir araya getirilebileceğini hayal edebilirsiniz (örnek için bölüm sonundaki alıştırmalara bakınız).

### 10.1.3 Perseptron (Algılayıcı)

*Perseptron*, 1957'de Frank Rosenblatt tarafından icat edilmiş olan en basit ANN mimarilerinden biridir. Eşik mantık birimi (TLU: *Threshold Logic Unit*) veya bazen doğrusal eşik birimi de (LTU: *Linear Threshold Unit*) denilen biraz farklı bir yapay sinir hücresine dayanır (Şekil-10.4'e bakınız). Girdiler ve çıktılar sayılardır (ikili açık/kapalı değerleri yerine) ve her bir girdi bağlantısı bir ağırlık değeriyle ilişkilidir. TLU girdilerin ağırlıklandırılmış toplamını hesaplar ( $z = w_1 x_1 + w_2 x_2 + \dots + w_n x_n = \mathbf{x}^T \mathbf{w}$ ) ve daha sonra bu toplama adım fonksiyonu uygular ve sonucu çıkarır:  $h_{\mathbf{w}}(\mathbf{x}) = \text{step}(z)$ , burada  $z = \mathbf{x}^T \mathbf{w}$ 'dir.



Şekil 10.4: Eşik mantık birimi: girdilerinin ağırlıklandırılmış toplamını hesaplayan ve daha sonra adım fonksiyonu uygulayan yapay bir sinir hücresi

Perseptronlarda en yaygın kullanılan adım fonksiyonu Heaviside adım fonksiyonudur (Denklemler 10.1'e bakınız). Bunun yerine bazen işaret fonksiyonu ( $\text{sgn}$ ) da kullanılır.

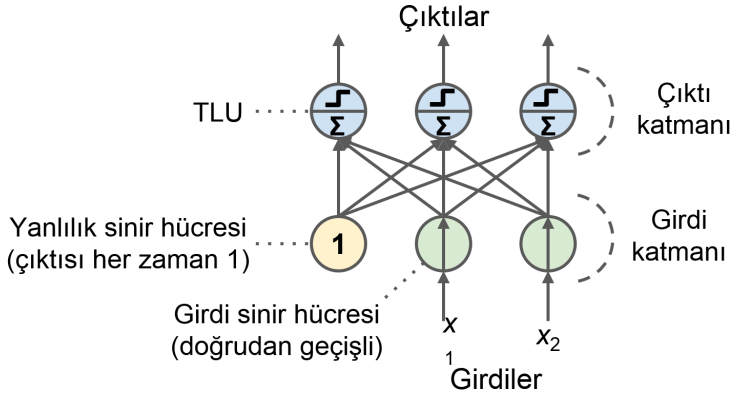
Perseptronlarda yaygın olarak kullanılan adım fonksiyonları (eşik=0 olduğunu varsayarak)

$$\text{heaviside}(z) = \begin{cases} 0, & z < 0 \\ 1, & z \geq 0 \end{cases} \quad \text{sgn}(z) = \begin{cases} -1, & z < 0 \\ 0, & z = 0 \\ +1, & z > 0 \end{cases} \quad (10.1)$$

Tek bir TLU basit doğrusal sınıflandırma için kullanılabilir. Girdilerin doğrusal bir bileşimini hesaplar ve sonuç belirli bir eşiği aştığında pozitif sınıfı sonucunu verir. Aksi takdirde negatif sınıf çıkarır (doğrusal bağlanım veya doğrusal

SVM sınıflandırıcısı gibi). Örneğin, taç yaprak uzunluğu ve genişliğine göre Süsen çiçeğini sınıflandırmak için tek bir TLU kullanabilirsiniz (daha önceki bölümlerde yaptığımız gibi ilave yanlılık özneliği olarak  $x_0 = 1$  ekleyerek). Bu durumda TLU'yu eğitmek  $w_0$ ,  $w_1$ , ve  $w_2$  için doğru değerleri bulmak anlamına gelir (eğitim algoritmasından birazdan bahsedeceğiz).

Bir perseptron her bir TLU bütün girdilere bağlanacak şekilde TLU'ların<sup>7</sup> tek bir katmanından oluşur. Bütün sinir hücreleri bir önceki katmandaki (yani girdi sinir hücreleri) her bir sinir hücresine bağlandığında bu katmana *tam bağlantılı katman* veya *yoğun katman* (ÇN: Dense layer) denir. Perseptronun girdileri *girdi sinir hücreleri* denilen doğrudan geçişli sinir hücrelerine beslenir: Hangi girdi ile beslenmişlerse onu çıktı olarak verirler. Bütün girdi sinir hücreleri *girdi katmanını* oluşturur. Dahası, genelde ilave bir yanlılık özneliği eklenir ( $x_0 = 1$ ): Tipik olarak *yanlılık sinir hücresi* denilen özel bir sinir hücresi çeşidiyle temsil edilirler. Bu sinir hücresinin çıktısı her zaman 1'dir. İki girdili ve üç çıktılı bir Perseptron Şekil-10.5'te gösterilmiştir. Bu Perseptron, örnekleri eş zamanlı olarak üç farklı ikili sınıfa sınıflandırabilir. Bu, Perseptronu çok etiketli bir sınıflandırıcı yapar.



Şekil 10.5: İki girdi sinir hücresi, bir yanlılık sinir hücresi ve üç çıktı sinir hücresi olan bir Perseptronun mimarisi

Doğrusal cebirin sihri sayesinde, [Denklem 10.2](#), yapay sinir hücreleri katmanının çıktılarını birkaç örnek için tek seferde etkin bir şekilde hesaplamayı mümkün kılar.

Yoğun katmanın çıktılarını hesaplamak

$$h_{\mathbf{W},\mathbf{b}}(\mathbf{X}) = \phi(\mathbf{X}\mathbf{W} + \mathbf{b}) \quad (10.2)$$

<sup>7</sup>Perseptron ismi bazen tek TLU'lu küçük bir ağ manasında da kullanılır.

Bu eşitlikte:

- Her zamanki gibi  $\mathbf{X}$ , girdi özneliklerinin matrisini temsil eder. Her bir örnek için bir satırı ve her bir öznelik için bir sütunu vardır.
- Ağırlık matrisi  $\mathbf{W}$ , yanlılık sinir hücresinden olanlar haricinde bütün bağlantı ağırlıklarını içerir. Katmandaki her bir girdi sinir hücresi için bir satırı ve her bir yapay sinir hücresi için bir sütunu vardır.
- Yanlılık vektörü  $\mathbf{b}$ , yanlılık sinir hücresi ve yapay sinir hücreleri arasındaki bütün bağlantı ağırlıklarını içerir. Her bir yapay sinir hücresi için bir yanlılık terimi vardır.
- $\phi$  fonksiyonuna *aktivasyon fonksiyonu* denir: Yapay sinir hücreleri TLU'lar olduğunda, bu aktivasyon fonksiyonu bir adım fonksiyonudur (aktivasyon fonksiyonlarından birazdan bahsedeceğiz).

Peki bir Perseptron nasıl eğitilir? Rosenblatt tarafından öne sürülen Perseptron eğitim algoritması çoğunlukla Hebb kuralından etkilenmişti. 1949 tarihli *The Organization of Behavior* isimli kitabında Donald Hebb, bir biyolojik sinir hücresinin diğer bir sinir hücresini sıkça tetiklediği zaman bu iki sinir hücresinin arasındaki bağlantının daha da güçlendiğini iddia etmiştir. Siegrid Löwel daha sonra Hebb'in fikrini hatırlaması kolay olan şu cümleyle özetledi: “Birlikte ateşlenen hücreler birbirine bağlanır”, yani, iki sinir hücresi arasındaki bağlantının ağırlığı hücreler eş zamanlı olarak ateşlendiğinde artma eğilimi gösterir. Bu kural daha sonra Hebb's kuralı (veya *Hebbian öğrenmesi*) olarak anıldı. Perseptronlar, tahmin yaparken, ağ tarafından yapılan hatayı hesaba katan bu kuralın bir çeşiti kullanılarak eğitilir. Perseptron öğrenme kuralı bağlantıları pekiştirir ve bu da hatayı azaltmaya yardımcı olur. Daha spesifik olarak, Perseptron her seferinde bir eğitim örneğiyle beslenir ve her bir örnek için tahminlerde bulunur. Yanlış tahminde bulunan her sinir hücresi için doğru tahmine katkıda bulunmuş olabilecek girdilerden gelen bağlantı ağırlıklarını pekiştirir. Kural, **Denklem 10.3**'te gösterilmiştir.

Perseptron öğrenme kuralı (ağırlık güncelleme)

$$w_{i,j}^{(\text{sonraki adım})} = w_{i,j} + \eta(y_j - \hat{y}_j)x_i \quad (10.3)$$

Bu eşitlikte:

- $w_{i,j}$ ,  $i^{\text{inci}}$  girdi sinir hücresi ve  $j^{\text{inci}}$  çıktı sinir hücresi arasındaki bağlantı ağırlığıdır.

- $x_i$ , mevcut eğitim örneğinin  $i^{\text{inci}}$  girdi değeridir.
- $(\hat{y}_j)$ , mevcut eğitim örneği için  $j^{\text{inci}}$  çıktı sinir hücresinin çıktısıdır.
- $y_j$ , mevcut eğitim örneği için  $j^{\text{inci}}$  çıktı sinir hücresinin hedef çıktısıdır.
- $\eta$  öğrenme oranıdır.

Her bir çıktı sinir hücresinin karar sınırı doğrusaldır dolayısıyla Perseptronlar karmaşık yapıları nedeniyle öğrenmede yetersizdirler (lojistik bağlanım sınıflandırıcıları gibi). Bununla birlikte, Rosenblatt, eğitim örneklerinin doğrusal ayrılabilir olması durumunda bu algoritmanın bir çözüme yakınsayacağını göstermiştir.<sup>8</sup> Buna *Perseptron yakınsama teoremi* denir.

Scikit-Learn tek bir TLU ağı gerçekleyen bir **Perceptron** sınıfı sağlamaktadır. Hemen hemen tahmin edeceğimiz şekilde kullanılabilir. Örneğin iris veri setinde (Bölüm 4'te gösterilmiştir).

```
import numpy as np
from sklearn.datasets import load_iris
from sklearn.linear_model import Perceptron
iris = load_iris()
X = iris.data[:, (2, 3)] # taç yaprak uzunluğu, taç yaprak genişliği
y = (iris.target == 0).astype(np.int) # Iris setosa?
per_clf = Perceptron()
per_clf.fit(X, y)
y_pred = per_clf.predict([[2, 0.5]])
```

Perseptron öğrenme algoritmasının stokastik gradyan inişine çok benzediği dikkatinizi çekmiştir. Aslında Scikit-Learn'nün **Perceptron** sınıfı şu hiperparametrelerle **SGDClassifier** kullanmaya denktir: `loss='perceptron'`, `learning_rate='constant'`, `eta0=1` (öğrenme oranı) ve `penalty=None` (düzenleştirme yok).

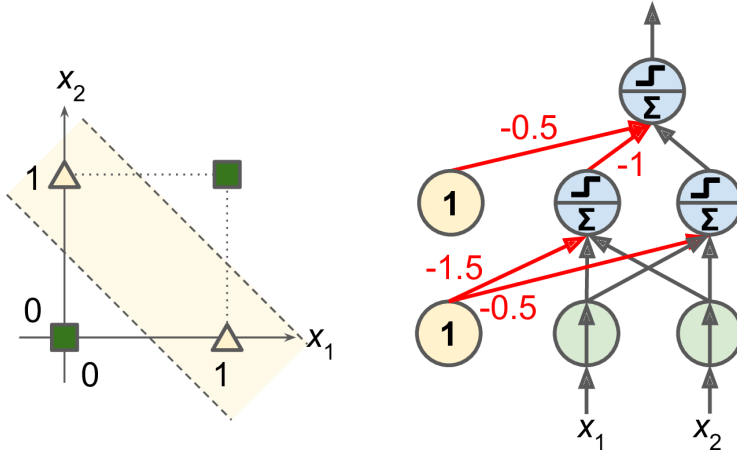
Lojistik bağlanım sınıflandırıcılarının aksine Perseptronların çıktı olarak sınıf olasılığı vermediğine dikkat edin; daha ziyade sert bir eşiğe dayalı olarak tahminde bulunurlar. Lojistik bağlanımı Perseptronlara nazaran tercih etme nedenlerinden biri budur.

1969 tarihli *Perseptronlar* adlı monografda Marvin Minsky ve Seymour Papert Perseptronların bir dizi ciddi zayıflığının altını çizdiler —özellikle bazı basit problemleri çözmede yetersiz olmaları (*örneğin dışlayıcı VEYA (XOR)*

<sup>8</sup>Bu çözümün emsalsiz olmadığına dikkat edin: Veri noktaları ayrılabilir olduğunda onları ayırabilecek sonsuz sayıda hiperdüzlem bulunur.

sınıflandırma problemi; Şekil-10.6'nın sol tarafına bakınız). Bu, diğer doğrusal sınıflandırma modelleri için de geçerlidir (Lojistik Bağlanım sınıflandırıcısı gibi) ancak araştırmacılar Perseptronlardan daha fazlasını ümit ettiler ve bazıları öyle büyük hayal kırıklığına uğramışlardı ki mantık, problem çözme ve arama gibi daha üst düzey sorunlar uğruna sinir ağından tümden vazgeçtiler.

Öyle görünüyor ki Perseptronların kısıtlarının bazıları çoklu Perseptronları üst üste yığarak giderilebilir. Oluşan ANN'ye *çok katmanlı Perseptron* (MLP: Multilayer Perceptron) denir. MLP, XOR problemini çözebilir. Bunu Şekil-10.6'nın sağ tarafında gösterilen MLP'nin çıktısını hesaplayarak doğrulayabilirsiniz: (0,0) veya (1,1) girdileri ile ağ 0 çıktısını verirken (0,1) ve (1,0) girdileri ile 1 çıktısını verir. Ağırlığı gösterilen dört bağlantı haricinde bütün bağlantıların ağırlığı 1'dir. Bu ağın XOR problemini gerçekten çözdüğünü doğrulamaya çalışınız!



Şekil 10.6: XOR sınıflandırma problemi ve onu çözen bir MLP

#### 10.1.4 Çok Katmanlı Perseptron ve Geri Yayılım

Bir MLP, bir adet (doğrudan geçişli) *girdi katmanından*, *gizli katmanlar* adı da verilen bir veya daha fazla TLU'lar içeren katmanlardan ve *çıkış katmanı* (Şekil-10.7'ye bakınız) denilen son bir TLU'lar katmanından oluşur. Girdi katmanına yakın olan katmanlar genellikle *alt katmanlar* ve çıkış katmanına yakın olanlar ise genellikle *üst katmanlar* olarak adlandırılır. Çıkış katmanı hariç her katman bir yanlılık sinir hücresi içerir ve bu yanlılık sinir hücresi bir sonraki katmana tam bağlıdır.

### 16.4.1 Görsel İlgi

İlgi mekanizmaları şimdilerde çeşitli amaçlar için kullanılmaktadır. NMT'nin ötesindeki ilk uygulamalarından biri **görsel ilgi** (<https://homl.info/visualattention>)<sup>17</sup> kullanarak görüntü başlığı üretmekteydi: Evrimsel bir sinir ağı önce görüntüyü işler ve bazı öznitelik haritalarını çıktı olarak verir. Daha sonra ilgi mekanizmasıyla donatılmış bir kodçözücü RNN her seferinde bir kelime olmak üzere başlığı üretir. Her bir kodçözücü zaman adımında (her bir kelime), kodçözücü, görüntünün doğru kısmına odaklanmak için ilgi modelini kullanır. Örneğin **Şekil-16.7**'de<sup>18</sup> model “A woman is throwing a frisbee in a park” (ÇN: Bir kadın parkta frizbi atıyor) başlığını üretmiştir. “frisbee” (ÇN: Frizbi) kelimesini çıktı olarak vermek üzereyken kodçözücünün görüntünün hangi kısmına ilgisini odakladığını görebilmektesiniz: Açık bir şekilde ilgisinin çoğu frizbi üzerine odaklanmıştır.



**Şekil 16.7:** Görsel ilgi: Bir resim girdisi (solda) ve modelin “frisbee” kelimesini üretmeden önce odaklanması (sağda)

### Açıklanabilirlik

İlgi mekanizmalarının ilave bir faydası, modelin çıktısının üretilmesine neyin yol açtığını anlamayı kolaylaştırmalarıdır. Buna *açıklanabilirlik* denir. Özellikle de model bir hata yaptığında kullanışlıdır: Örneğin, karda yürüyen bir köpeğin görüntüsü için “a wolf walking in the snow” (ÇN: Karda yürüyen bir kurt) şeklinde bir başlık verilmişse geri gidip

<sup>17</sup>Kelvin Xu et al., “Show, Attend and Tell: Neural Image Caption Generation with Visual Attention,” *Proceedings of the 32nd International Conference on Machine Learning* (2015): 2048–2057.

<sup>18</sup>Bu, makaledeki Şekil 3'ün bir kısmıdır. Yazarların kibar izniyle yeniden üretilmiştir.

modelin “wolf” (ÇN: Kurt) kelimesini çıktı olarak verdiğinde neye odaklandığını inceleyebilirsiniz. Sadece köpeğe ilgi göstermediğini aynı zamanda kara da ilgi gösterdiğini görebilirsiniz. Bu ise olası bir açıklamayı işaret eder: Belki de modelin köpekleri kurtlardan ayırt etmeyi öğrenme yöntemi etrafta çok kar olup olmadığını kontrol etmekle olmaktadır. Böylece bu hatayı modeli etrafta kar olmayan daha fazla kurt görüntüleri ve etrafta kar olan köpek görüntüleriyle eğiterek giderebilirsiniz. Bu örnek, Marco Tulio Riberio vd. tarafından yazılan, açıklanabilirliğe yönelik daha farklı yaklaşım kullanan **2016 yılına ait harika bir makaleden** (<https://homl.info/explainclass>)<sup>19</sup> gelmektedir: Bir sınıflandırıcının tahmini etrafında yerel olarak yorumlanabilir bir modeli öğrenmek.

Bazı uygulamalarda açıklanabilirlik sadece bir modelde hata ayıklama aracı değildir; aynı zamanda yasal bir gereklilik de olabilir (size kredi verip vermeyeceğine karar veren bir sistemi düşünün).

İlgi mekanizmaları o kadar güçlüdür ki sadece ilgi mekanizmaları kullanarak çok gelişmiş modeller inşa edebilirsiniz.

## 16.4.2 Tek İhtiyacınız Olan İlgidir: Dönüştürücü Mimarisi

**2017 yılına ait çığır açan bir makalede** (<https://homl.info/transformer>)<sup>20</sup> Google araştırmacıları “Tek ihtiyacımız olanın ilgi” olduğunu öne sürmüşlerdir. *Dönüştürücü* (ÇN: Transformes) denilen bir mimariyi oluşturmayı başarmışlardır. Bu mimari yinelemeli veya evrimsel katmanlar<sup>21</sup> kullanmadan sadece ilgi mekanizmaları (artı gömülme katmanları, yoğun katmanlar, normalleştirme katmanları ve birkaç ufak tefek şey) kullanarak NMT’deki teknolojiyi önemli ölçüde iyileştirmiştir. İlave bir bonus olarak, bu mimariyi eğitmesi çok daha hızlıydı ve paralelleştirmesi çok daha basitti, dolayısıyla daha önceki son teknoloji modellerin zaman ve maliyetinin çok altında bu mimariyi eğitmeyi başardılar.

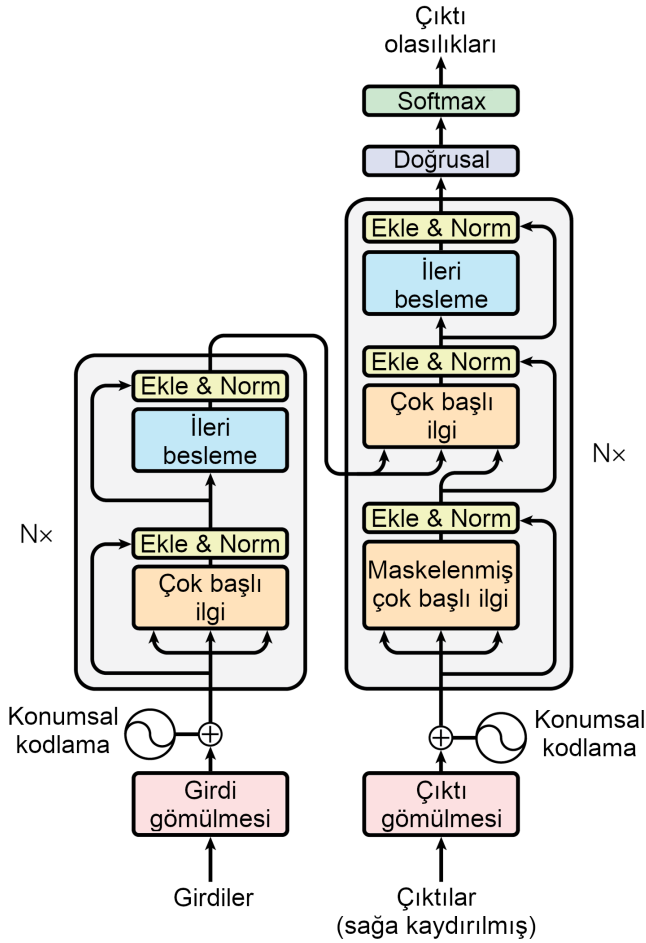
Dönüştürücü mimarisi **Şekil-16.8**’de<sup>22</sup> gösterilmiştir.

<sup>19</sup>Marco Tulio Ribeiro et al., “‘Why Should I Trust You?’: Explaining the Predictions of Any Classifier,” *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2016): 1135–1144.

<sup>20</sup>Ashish Vaswani et al., “Attention Is All You Need,” *Proceedings of the 31st International Conference on Neural Information Processing Systems* (2017): 6000–6010.

<sup>21</sup>Dönüştürücü, zaman dağılımlı Dense katmanları kullandığından, kernel büyüklüğü 1 olan bir boyutlu evrimsel katmanlar kullandığını iddia edebilirsiniz.

<sup>22</sup>Bu, makaledeki Şekil 1’dir. Yazarların kibar izniyle yeniden üretilmiştir.



Şekil 16.8: Dönüştürücü mimarisi

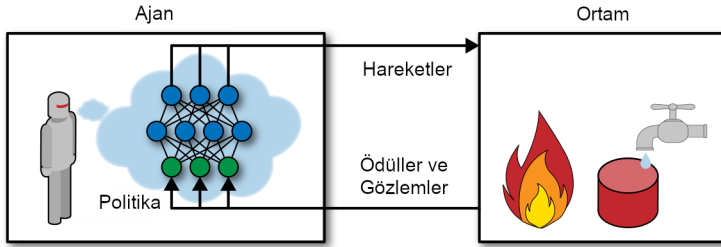
Bu şekili inceleyelim:

- Sol taraf kodlayıcıdır. Daha önce olduğu gibi kelime numaralarının bir dizisi olarak temsil edilen cümlelerin bir yığını girdi olarak alır (girdi şekli [*yıgın büyüklüğü, en büyük cümle girdisi uzunluğu*]’dur) ve her bir kelimeyi 512 boyutlu bir gösterime kodlar (dolayısıyla kodlayıcının çıktı şekli [*yıgın büyüklüğü, en büyük cümle girdisi uzunluğu, 512*]’dir). Kodlayıcının üst kısmının  $N$  kere üst üste yığıldığına dikkat ediniz (makalede  $N=6$ ’dır).
- Sağ taraf kodçözücüdür. Eğitim esnasında hedef cümleyi bir zaman adımı kadar sağa kaydırılmış olarak (en başa dizi başlangıcı andıcı eklenir) girdi alır (kelime numaralarının bir dizisi olarak da temsil edilir). Aynı



## 18.2 Politika Araması

Bir yazılım ajanının hareketlerini belirlemek için kullandığı algoritmaya o ajanın politikası denir. İlke, gözlemleri girdi olarak alan ve yapılması gereken hareketleri çıktı olarak veren bir sinir ağı olabilir (bakınız Şekil-18.2).



Şekil 18.2: Bir sinir ağı politikası kullanan pekiştirmeli öğrenme

İlke düşünebileceğiniz herhangi bir algoritma olabilir ve deterministik olmak zorunda değildir. Aslında bazı durumlarda ortamı gözlemlemek zorunda bile değildir! Örneğin, ödülü 30 dakikada topladığı toz miktarı olan robotik bir elektrikli süpürge düşünün. Bu süpürge politikası, her saniye  $p$  olasılıkla ilerlemek veya  $1-p$  olasılıkla rastgele sola sağa dönmek olabilir. Dönme açısı  $-r$  ve  $+r$  arasında rastgele bir açı olur. Bu politika bir miktar rastgelelik içerdiğinden *stokastik politika* olarak adlandırılır. Robot düzensiz bir yol takip edecektir. Bu ise ulaşabileceği her yere nihayetinde gideceğini ve bütün tozu toplayacağını garanti eder. Soru ise 30 dakikada ne kadar toz toplayacağıdır.

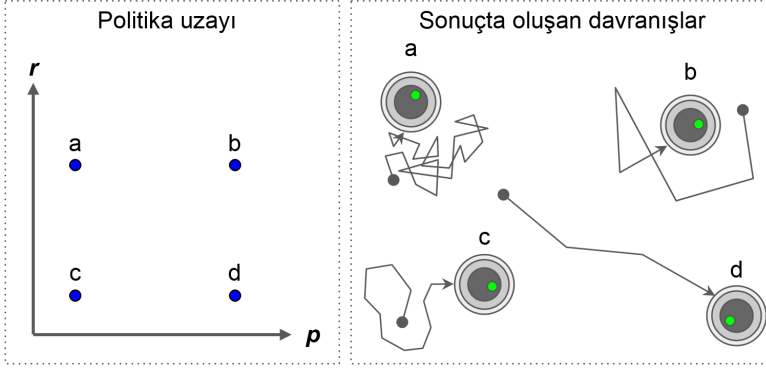
Böyle bir robotu nasıl eğitirsiniz? Ayarlayabileceğiniz sadece iki politika parametresi vardır:  $p$  olasılığı ve  $r$  açı aralığı. Muhtemel bir öğrenme algoritması bu parametreler için pek çok farklı değer denemek ve en iyi performansı veren kombinasyonu seçmektir (bakınız Şekil-18.3). Bu, *politika aramasının* bir örneğidir. Bu durumda brüt kuvvet yaklaşımı kullanılmaktadır. *Politika uzayı* çok geniş olduğunda (genelde böyledir), bu şekilde iyi bir parametre kümesi bulmak devasa bir saman yığnında iğne aramaya benzer.

Politika uzayını keşfetmenin bir başka yolu *genetik algoritmalar* kullanmaktır. Örneğin, 100 politikanın rastgele ilk neslini oluşturabilir ve bunları deneyebilir, daha sonra en kötü 80 politikayı<sup>6</sup> öldürüp, kurtulan 20 politikanın her birinin 4 yavru vermesini sağlayabilirsiniz. Yavru, ebeveyninin<sup>7</sup> bir kopyası artı bir

<sup>6</sup>“gen havuzunda” bir miktar çeşitliliği muhafaza etmek için zayıf performans gösterenlere azıcık kurtulma şansı vermek sıklıkla iyidir.

<sup>7</sup>Tek bir ebeveyn varsa buna *eşeysiz üreme* denir. İki (veya daha fazla) ebeveyn varsa buna *eşeyli üreme* denir. Yavrunun genomu (bu durumda politika parametreleri kümesi) ebeveynlerinin genomlarının rastgele kısımlarından oluşur.

miktar rastgele değişimdir. Kurtulan politikalar artı yavruları birlikte ikinci nesli oluşturur. İyi bir politika<sup>8</sup> bulana kadar bu şekilde nesiller boyunca döngü kurabilirsiniz.



Şekil 18.3: Politika uzayında dört nokta (solda) ve ajanın ilgili davranışı (sağda)

Bir başka yaklaşım, politika parametrelerine göre ödüllerin gradyanlarını hesaplamak ve daha sonra bu parametreleri daha yüksek ödüllere<sup>9</sup> doğru olan gradyanları takip ederek ayarlamak suretiyle eniyileme teknikleri kullanmaktır. *Politika gradyanları* (PG: Policy Gradients) denilen bu yaklaşımdan bu bölümde daha sonra daha ayrıntılı olarak bahsedeceğiz. Elektrikli süpürge robotuna geri dönersek,  $p$ 'yi az miktarda artırabilir ve bunu yapmanın 30 dakika içinde robot tarafından toplanan toz miktarını artırıp artırmadığını hesaplayabilirsiniz. Eğer artırırsa  $p$ 'yi biraz daha artırmak, aksi takdirde  $p$ 'yi azaltın. TensorFlow kullanarak popüler bir PG algoritması gerçekleyeceğiz ama bunu yapmadan önce ajanın yaşayacağı bir ortam oluşturmamız gerekiyor —dolayısıyla OpenAI Gym'i tanıtmaya vakti gelmiştir.

## 18.3 OpenAI Gym'e Giriş

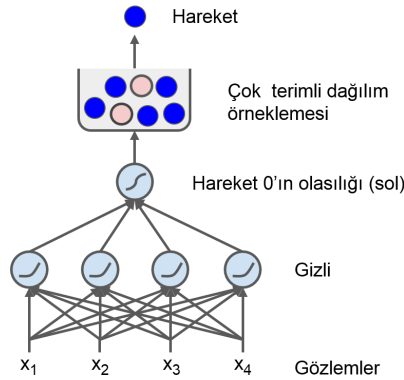
Pekiştirmeli Öğrenmenin zorluklarından biri bir ajanı eğitmek için öncelikle çalşan bir ortamın gerekli olmasıdır. Atari oyununu oynamasını öğrenecek olan bir ajan programlamak istiyorsanız Atari oyun simülatörüne ihtiyacınız olacaktır. Yürüyen bir robot programlamak isterseniz, ortam gerçek dünyadır ve robotunuzu doğrudan bu ortamda eğitebilirsiniz ancak bunun kendi sınırları

<sup>8</sup>Pekiştirmeli Öğrenme için kullanılan ilginç bir genetik algoritma örneği *NeuroEvolution of Augmenting Topologies* (<https://homl.info/neat>) (NEAT) algoritmasıdır.

<sup>9</sup>Buna gradyan çıkışı denir. Gradyan inişi gibidir ama tersi yönde: Enküçültmek yerine enbüyütmek.

## 18.4 Sinir Ağı Politikaları

Bir sinir ağı politikası oluşturalım. Daha önce gömülü olarak kodladığımız politika gibi bu sinir ağı bir gözlemi girdi olarak alacak ve yerine getirilecek hareketi çıktı olarak verecektir. Daha kesin olmak gerekirse bu sinir ağı her bir hareket için bir olasılık kestirecek ve daha sonra kestirilen olasılıklara göre rastgele bir hareket seçeceğiz (bakınız Şekil-18.5). CartPole ortamı durumunda sadece iki muhtemel hareket vardır (sol veya sağ), dolayısıyla sadece bir tane çıktı sinir hücresine ihtiyacımız vardır. Bu çıktı hareket 0'ın (sol)  $p$  olasılığını çıktı olarak verecektir ve elbette ki hareket 1'in (sağ) olasılığı  $1-p$  olacaktır. Örneğin çıktı olarak 0.7'yi verirse hareket 0'ı %70 olasılıkla veya hareket 1'i %30 olasılıkla seçeceğiz demektir.



Şekil 18.5: 5 Sinir ağı politikası

En yüksek skorlu hareketi seçmekten ziyade neden sinir ağı tarafından verilen olasılıklara dayalı olarak rastgele bir hareket seçtiğimizi merak ediyor olabilirsiniz. Bu yaklaşım ajanın yeni hareketleri *keşfetmek* ile iyi çalıştığı bilinen hareketlerden  *faydalanmak* arasındaki doğru dengeyi bulmasını sağlar. İşte bir benzerlik: Bir restorana ilk defa gittiğinizi varsayın. Bütün yemekler aynı güzellikte görünsün. Dolayısıyla rastgele bir tanesini seçersiniz. İyi çıkarsa, bir sonraki sefer bu yemeği isteme olasılığınızı artırabilirsiniz. Ancak bu olasılığı %100'e kadar artırmamalısınız, aksi takdirde diğer yemekleri denemeyebilirsiniz ve bu yemeklerden bazıları denediğinizden daha güzel olabilir.

Her bir gözlem ortamın tam durumunu içerdiğinden, bu özel ortamda geçmiş hareketlerin ve gözlemlerin güvenli bir şekilde ihmal edilebileceğine dikkat ediniz. Birtakım gizli durumlar varsa o zaman geçmiş hareketleri ve gözlemleri göz önüne alabilirsiniz. Örneğin, ortam el arabasının sadece konumunu açığa vurmuşsa ama hızını açığa vurmamışsa o anki hızı kestirmek için sadece o anki gözlemi değil aynı zamanda bir önceki gözlemi de göz önünde bulundurmak

zorunda kalırsınız. Bir başka örnek ise gözlemlerin gürültülü olduğu durumdur. Bu durumda genellikle en muhtemel o anki durumu kestirmek için geçmiş birkaç gözlemleri kullanmak istersiniz. CartPole problemi olabileceği kadar basittir; gözlemler gürültüsüz ve ortamın tam durumunu içermektedir.

Aşağıda tf.keras kullanarak bu sinir ağı politikasını inşa etmek için gerekli kod bulunmaktadır:

```
import tensorflow as tf
from tensorflow import keras

n_inputs = 4 # == env.observation_space.shape[0]

model = keras.models.Sequential([
    keras.layers.Dense(5, activation="elu", input_shape=[n_inputs]),
    keras.layers.Dense(1, activation="sigmoid"),
])
```

import'lardan sonra politika ağını tanımlamak için basit bir `Sequential` model kullanıyoruz. Girdi sayısı gözlem uzayının boyudur (CartPole için bu sayı 4'tür) ve sadece beş gizli birimimiz vardır çünkü bu basit bir problemdir. Son olarak, tek bir olasılığı (sola gitme olasılığı) çıktı olarak vermek istiyoruz, dolayısıyla sigmoid aktivasyon fonksiyonu kullanan tek bir çıktı sinir hücremiz vardır. İki muhtemel hareketten daha fazlası olsaydı hareket başına bir çıktı sinir hücresi olurdu ve bu durumda softmax aktivasyon fonksiyonunu kullanırdık.

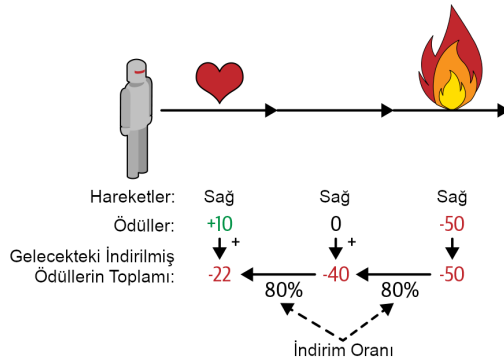
Tamam, şimdi gözlemleri alıp hareket olasılıklarını çıktı olarak verecek bir sinir ağı politikamız var. Ancak bunu nasıl eğitiriz?

## 18.5 Hareketleri Değerlendirmek: Kredi Atama Problemi

Her adımda en iyi hareketin ne olduğunu bilseydik, kestirilmiş olasılık dağılımı ve hedef olasılık dağılımı arasındaki çapraz entropiyi enküçülterek sinir ağını her zamanki gibi eğitebilirdik. Bu, sıradan denetimli öğrenme olurdu. Ancak pekiştirmeli öğrenmede ajanın aldığı tek kılavuzluk ödüller yoluyla ve ödüller tipik olarak seyrek ve gecikmelidir. Örneğin ajan, direği 100 adım boyunca dengelemeyi başarır, yaptığı 100 hareketin hangilerinin iyi hangilerinin kötü olduğunu nasıl bilecektir? Tek bildiği son hareketten sonra direğin düştüğüdür fakat bu son hareket elbette tamamen sorumlu değildir. Bu, kredi atama problemi olarak bilinir: Ajan bir ödül aldığı anda hangi hareketlerinin kredilen-

dirildiğini (veya suçlanacağını) bilmesi zordur. İyi davrandıktan saatler sonra ödüllendirilen bir köpek düşünün; niçin ödüllendirildiğini anlayacak mıdır?

Bu problemin üstesinden gelmek için sıkça kullanılan bir strateji, genellikle her adımda bir  $\gamma$  (gama) *indirim çarpanı* uygulamak suretiyle, söz konusu hareketten sonra gelen bütün ödüllerin toplamına dayanan bir hareketi hesaplamaktır. Bu indirilmiş ödüllerin toplamına hareketin getirisi denir. Şekil-18.6'daki örneği düşünün. Ajan art arda üç kez sağa gitmeye karar verirse ve ilk adımdan sonra +10 ödül, ikinci adımdan sonra 0 ve son olarak üçüncü adımdan sonra -50 alırsa,  $\gamma = 0.8$  olacak şekilde bir indirim çarpanı kullandığımızı varsayarak, ilk hareketin getirisi  $10 + \gamma \times 0 + \gamma^2 \times (-50) = -22$  olur. İndirim çarpanı 0'a yakın ise gelecekteki ödüller anlık ödüllerle kıyaslandığında çok fazla sayılmaz. Aksine, indirim çarpanı 1'e yakınsa uzak gelecekteki indirimler hemen hemen dolaysız ödüller kadar sayılacaktır. Tipik indirim çarpanları 0.9'dan 0.99'a kadar değişir. 0.99'luk bir indirim çarpanı ile 69 adım gelecekteki ödüller anlık ödüllerin yarısı kadar sayılırken, 0.95'lik bir indirim çarpanı ile 13 adım gelecekteki ödüller anlık ödüllerin kabaca yarısı kadar sayılır ( $0.95^{13} \approx 0.5$  olduğundan dolayı). CartPole ortamında hareketler oldukça kısa vadeli etkilere sahiptir. Dolayısıyla 0.95'lik bir indirim çarpanı kullanmak makul görünmektedir.



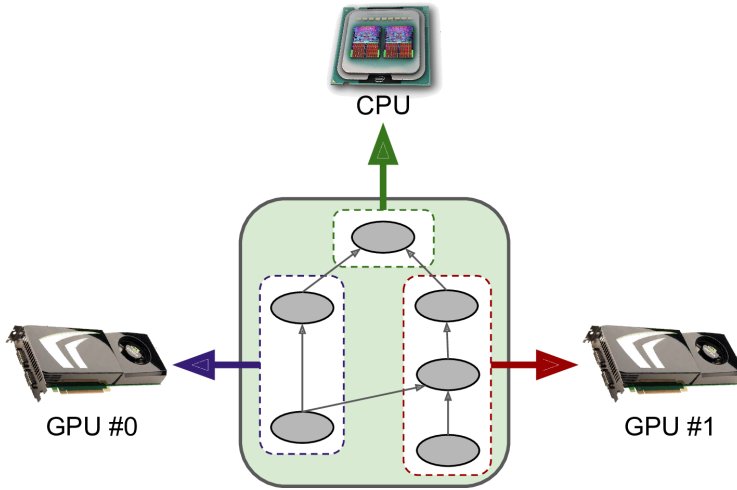
Şekil 18.6: Bir hareketin getirisini hesaplamak: Gelecekteki indirilmiş ödüllerin toplamı

Elbette ki iyi bir hareketi direğin çabucak düşmesine neden olan birkaç kötü hareket takip edebilir. Bu ise iyi hareketin düşük bir getiri elde etmesini sağlar (benzer şekilde iyi bir aktörün kötü bir filmde yıldızı parlayabilir). Bununla birlikte, oyunu yeterince oynarsak ortalamada iyi hareketler kötü hareketlerden daha yüksek bir getiri elde edecektir. Ortalamada, bir hareketin diğer muhtemel hareketlerle karşılaştırıldığında ne kadar iyi veya ne kadar kötü olduğunu kestirmek isteriz. Buna *hareket avantajı* denir. Bunun için pek çok epizod çalıştırmalı, bütün hareket getirilerini normalleştirmeliyiz (ortalamayı çıkarıp). Bundan sonra, makul bir şekilde negatif avantajlı hareketlerin kötü,

## 19.3 Hesaplamaları Hızlandırmak için GPU Kullanmak

**Bölüm 11**'de eğitimi oldukça hızlandırabilen farklı teknikler gördük: Ağırlıklara daha iyi ilk değer atama, Yığın Normalleştirme, daha karmaşık eniyileyciler vb. Bütün bu teknikleri kullansanız bile büyük bir sinir ağını bir CPU'ya sahip bir makine de eğitmek günler hatta haftalar alabilir.

Bu kısımda modelleri hızlandırmada GPU'ların nasıl kullanılacağına bakacağız. Aynı zamanda bir CPU ve birden çok GPU'ya sahip birden çok makineye hesaplamaların dağıtılmasını inceleyeceğiz (**Şekil-19.9**'a bakınız). Şimdilik her şeyi tek bir makinede çalıştıracamız ve bölüm sonuna doğru birden çok sunucuya hesaplamaların nasıl dağıtılacağını tartışacağız.



**Şekil 19.9:** Birden çok cihazda paralel olarak TensorFlow çizgisini çalıştırma

GPU'lar sayesinde eğitim algoritmasının bitmesini günlerce ya da haftalarca beklemek yerine sadece birkaç dakika veya saat bekleyebilirsiniz. Bu sadece çok fazla zaman kazanma anlamına gelmez, aynı zamanda farklı modelleri deneyebilmeniz ve sıklıkla yeni veriler üzerinde modelinizi yeniden eğitebilmeniz anlamına gelir.



Genellikle tek bir makineye GPU kartları ekleyerek büyük bir performans artışı elde edebilirsiniz. Aslında çoğu durumda bu yeterlidir ve birden çok makine kullanmaya ihtiyacınız olmayacaktır. Örneğin, dağıtılmış bir kurulumdaki ağ iletişiminin gerektirdiği ekstra gecikme nedeniyle, tipik olarak bir sinir ağını, birden çok makine boyunca dağıtılmış 8 GPU yerine tek bir makinede dört GPU kullanarak aynı hızda eğitilebilirsiniz. Benzer şekilde, tek bir güçlü GPU kullanmak, genellikle, birden çok yavaş GPU kullanmaya tercih edilir

İlk adım olarak bir GPU kullanımıyla başlayalım. Bunun için iki seçenek var: Kendi GPU ya da GPU'larınızı satın almak ya da bulut üzerinde GPU ile donatılmış sanal makineler kullanmak. Şimdi ilk seçenikle başlayalım.

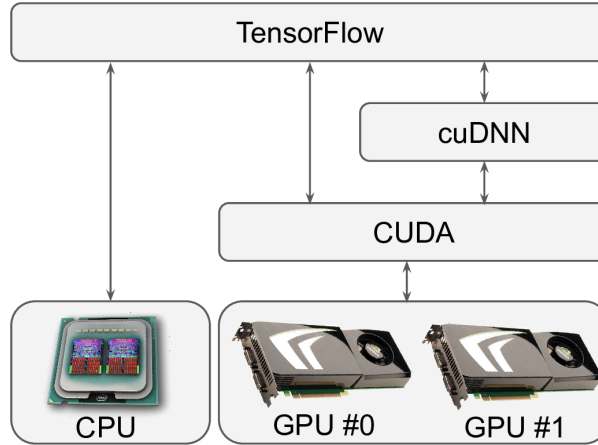
### 19.3.1 Kendi GPU'nuzu Alma

Eğer bir GPU kartı almayı seçerseniz doğru seçimi yapmak biraz zaman alacaktır. Tim Dettmers'in seçim yapmanıza yardımcı olacak ve düzenli olarak güncellediği [harika bir internet günlüğü bulunmaktadır \(https://homl.info/66\)](https://homl.info/66): Dikkatlice okumanızı tavsiye ederim. Bu kitabın yazıldığı sırada TensorFlow sadece **CUDA Compute Capability 3.5 ve daha büyük Nvidia kartlara (https://homl.info/-cudagpus)** destek vermektedir (elbette aynı zamanda Google'ın TPU'larına) ama ileride diğer üreticileri de destekleyebilir. Her ne kadar TPU'lar şimdilik sadece GCP üzerinden ulaşılabilir olsalar da yakın gelecekte TPU benzeri kartlar satışa çıkabilir ve TensorFlow bu kartları destekleyebilir. Kısacası, bu noktada hangi cihazların desteklendiğini görmek için **TensorFlow'un dokümantasyonunu (https://tensorflow.org/install)** kontrol ettiğinizden emin olun.

Eğer bir Nvidia kartı seçerseniz Nvidia donanım sürücülerini ve bazı diğer Nvidia kütüphanelerini<sup>10</sup> kurmalısınız. Bunlar geliştiricilere CUDA destekli GPU'larda her türlü hesaplamayı (sadece grafik hızlandırma değil) yapma imkânı sağlayan *Compute Unified Device Architecture kütüphanesi* (CUDA) ve derin sinir ağlarındaki işlemleri için GPU hızlandırmaları sağlayan *CUDA Deep Neural Network* kütüphanesidir (cuDNN). cuDNN aktivasyon katmanları, normalizasyon, ileriye ve geriye evrişim ve biriktirme gibi çok sık kullanılan derin sinir ağı hesaplamaları için eniyilenmiş gerçekleştirmeler sağlar (**Bölüm 14'e** bakınız). Nvidia'nın Deep Learning SDK'sının bir parçasıdır (indirmek için Nvidia geliştirici hesabı oluşturmanız gerekecektir). TensorFlow, CUDA ve

<sup>10</sup>Güncel kurulum adımları ve detaylar için lütfen dokümantasyonu sıkça kontrol edin.

cuDNN kütüphanelerini GPU'ları kontrol etmek ve hesaplamaları hızlandırmak için kullanır (Şekil-19.10'a bakınız).



**Şekil 19.10:** TensorFlow, GPU'ları kontrol etmek ve DNN'leri hızlandırmak için CUDA ve cuDNN kullanır

GPU kartlarını (veya kartını) ve gerekli tüm sürücülerini ve kütüphanelerini yükledikten sonra, CUDA'nın doğru şekilde kurulup kurulmadığını kontrol etmek için `nvidia-smi` komutunu kullanabilirsiniz. Mevcut GPU kartlarını ve her kartta çalışan işlemleri listeler:

```

$ nvidia-smi
Sun Jun  2 10:05:22 2019

+-----+
| NVIDIA-SMI 418.67 Driver Version: 410.79 CUDA Version: 10.0 |
+-----+-----+
| GPU Name Persistence-M| Bus-Id Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap| Memory-Usage | GPU-Util Compute M. |
+-----+-----+-----+
| 0 Tesla T4 Off | 00000000:00:04.0 Off | 0 |
| N/A  61C  P8  17W /  70W | 0MiB / 15079MiB | 0% Default |
+-----+-----+-----+

+-----+
| Processes: GPU Memory |
| GPU PID  Type  Process name  Usage |
+-----+-----+
| No running processes found |
+-----+
  
```



Bu kitabın yazıldığı zaman için TensorFlow'un GPU versiyonunu da (örneğin `tensorflow-gpu` kütüphanesi) kurmaya ihtiyacınız vardı. Ancak sadece CPU'lu ve GPU'lu makinelere kurulumu birleştiren bir çalışma devam ediyordu. Dolayısıyla lütfen kurulum dokümantasyonunu inceleyerek hangi kütüphaneyi kurmanız gerektiğini kontrol edin. Her durumda tüm gerekli kütüphanelerin doğru şekilde kurulumu biraz zahmetli ve zaman alıcı olduğundan (eğer doğru versiyonlarını kurmamanız hâlinde içinden çıkılmaz hâle gelecektir) TensorFlow ihtiyacınız olan her şeyin içerisinde olduğu bir Docker görüntüsü sağlamaktadır. Ancak Docker konteyneri içerisinde GPU'ya erişmek için hâlâ ana makinenizde Nvidia donanım sürücülerinin kurulu olmasına ihtiyacınız olacaktır.

TensorFlow'un GPU'ları gerçekten gördüğünü kontrol etmek için aşağıdaki testleri çalıştırın:

```
>>> import tensorflow as tf
>>> tf.test.is_gpu_available()
True
>>> tf.test.gpu_device_name()
'/device:GPU:0'
>>> tf.config.experimental.list_physical_devices(device_type='GPU')
[PhysicalDevice(name='/physical_device:GPU:0', device_type='GPU')]
```

`is_gpu_available()` fonksiyonu en azından bir GPU'nun mevcut olup olmadığını kontrol eder. `gpu_device_name()` fonksiyonu ilk GPU'nun adını döndürür: Varsayılan olarak işlemler bu GPU üzerinde çalışacaktır. `list_physical_devices()` fonksiyonu tüm mevcut GPU cihazlarının listesini (bu örnekte bir tane var) döndürür.<sup>11</sup>

Şimdi eğer zamanınızı ve paranızı kendi GPU'nuzaya yatırmayı istemezseniz ne yapabileceğinize bakalım. Bulutta GPU'lu sanal makine kullanırsınız!

### 19.3.2 GPU'lu Sanal Makine Kullanımı

Tüm büyük bulut platformları önceden tüm donanım sürücülerini konfigüre edilmiş ve ihtiyacınız olan kütüphaneleri (TensorFlow gibi) barındıran GPU'lu sanal makineler sunmaktadır. Google Cloud Platform dünya çapında ve bölge bazında GPU kotaları uygular: Google'dan izin almadan binlerce GPU'lu

<sup>11</sup>Bu bölümdeki kod örneklerinin çoğu deneysel API'ları kullanmaktadır. İlerideki versiyonlarda temel API içine taşınma ihtimali bulunmaktadır. Sonuç olarak eğer deneysel bir metod hata verirse `experimental` kelimesi iç aktarmadan silip tekrar denerseniz muhtemelen çalışacaktır. Eğer bu da olmazsa API değişmiş olabilir. Her zaman doğru kodun olmasını sağlayacağımdan lütfen Jupyter not defterini kontrol edin.

# Dizin

- 1B evrimsel katmanlar, 555
- 1cycle çizelgeleme, 381
- A/B testleri, 709
- Actor-Critic algoritması, 663, 706
- açıklanabilirlik, 589
- AdaBoost, 211
- AdaGrad, 374
- Adam ve Nadam eniyilemesi, 376
- adaptif örnek normalleştirme (AdaIN), 642
- Adaptive Boosting, 211
- adım aralığı, 479
- adım fonksiyonu, 294
- afın dönüşüm, 642
- ağırlıkları bağlamak, 614
- ağırlıklı hareketli ortalama modeli, 539
- AI Platform, 722
- Akaike bilgi kriteri (AIC), 278
- akışlı metrik, 413
- aktarmalı öğrenme, 339, 364, 512
- aktif kısıt, 811
- aktif öğrenme, 265
- aktivasyon fonksiyonları
  - doyuma ulaşmama, 351
  - hiperbolik tanjant fonksiyonu (tanh), 301
  - lojistik, 153, 303, 315, 348
  - Rectified Linear Unit function (ReLU), 302–303
  - Scaled Exponential Linear Unit (SELU), 350, 354–355, 389
  - softmax, 305, 311, 500, 513, 519, 580
  - softplus, 303
  - üstel doğrusal birim (ELU: Exponential Linear Unit), 353–355
- AlexNet, 494
- alfa kanalı, 260
- algoritmalar
  - Actor-Critic algoritması, 663, 706
  - AllReduce algoritması, 749
  - anomali tespiti için, 284
  - asen kron avantaj actor-critic (A3C), 706
  - avantaj actor-critic (A2C), 706
  - Beklenen Değeri Enbüyütme (EM) algoritması, 273
  - BIRCH algoritması, 269
  - CART eğitim algoritması, 189, 191
  - çekişen DQN algoritması, 682
  - değer yineleme algoritması, 666
  - denetimli öğrenme, 9
  - denetimsiz öğrenme, 11
  - dinamik yerleştirici algoritması, 742
  - genetik algoritmalar, 649
  - görselleştirme algoritmaları, 13
  - hırslı algoritma, 191
  - hiyerarşik kümeleme algoritması, 12
  - isomap algoritması, 243
  - izole ormanlar algoritması, 285
  - K-ortalamalar algoritması, 248
  - kümeleme algoritmaları, 12
  - Lloyd-Forgy algoritması, 248
  - Mean-Shift algoritması, 269
  - politika içi algoritmalar, 671
  - politika-dışı algoritma, 671
  - proksimal politika eniyilemesi (PPO), 707
  - rastgele PCA algoritması, 236
  - REINFORCE algoritmaları, 658
  - tek-sınıflı SVM algoritması, 285
  - veriye karşı önemi, 27
  - yumuşak actor-critic algoritması, 706

- alıştırma çözümleri, 765–803
- AllReduce algoritması, 749
- alt gradyan vektörü, 150
- alt türevler, 184
- alt uzay, 226
- altörnekleme, 486
- Altınıflama API (Subclassing API), 326
- Ana Bileşenler Analizi (PCA)
  - ana bileşen eksenleri, 231
  - anomali ve yenilik tespiti, 285
  - artırmalı PCA, 236
  - d boyuta projeksiyon , 232
  - değişirliği korumak, 230
  - değişirlik oranı, 233
  - doğru boyutun seçimi, 234
  - genel bakış, 229
  - kernel PCA (kPCA), 237–240
  - rastgele PCA, 236
  - tamamlanmamış doğrusal
    - otokodlayıcılar için PCA, 607
- andıçlama, 572
- anlamsal aradeğerleme, 629
- anlamsal bölütleme, 259, 488, 524
- anomali tespiti, 247, 276
  - amacı, 246
  - Gauss karışım kullanarak anomali
    - tespiti, 276
  - için diğer algoritmalar, 284
  - kümeleme kullanarak, 247
  - örnekleri, 14
- arama motorları, 248
- argmax operatörü, 159
- arıtan otokodlayıcılar, 619
- artık birimler, 502
- artık bloklar, 420
- artık hatası, 215
- artık öğrenme, 501
- artırmalı öğrenme, 18
- artırmalı PCA (IPCA), 236
- asen kron avantaj actor-critic (A3C), 706
- asen kron güncellemeler, 751
- asimetrik veri seti, 100
- aşırı büyüyen gradyanlar, 348
- aşırı rastgele ağaçlar topluluğu, 209
- aşırı uydurma
  - düzenleştirme ile önlemek, 384–392
  - riskini limitlemek, 486
  - tanım, 31
- Atari önışleme, 687
- atlamalı bağlantı, 354, 501
- AutoGraphs, 434
- AutoML, 338
- avantaj actor-critic (A2C), 706
- ayrılabilir evrişim, 504
- ayrıştırıcı, 605
- bağımsız ve özdeşçe dağılım (IID), 137
- bağlanım problemleri
  - bağlanım MLP'leri, 303
  - çok değişkenli bağlanım, 44
  - çoklu bağlanım, 44
  - doğrusal bağlanım, 123–129
  - k-en yakın komşu bağlanımı, 25
  - karar ağaçları, 194
  - Lasso bağlanımı, 148
  - lojistik bağlanım, 153–162
  - polinom bağlanımı, 139
  - Ridge bağlanımı, 145
  - Sequential API kullanarak MLP
    - bağlanımı, 320
  - Softmax bağlanımı, 158–162
  - SVM bağlanımı, 174
  - tanım, 10
  - tek değişkenli bağlanım, 44
- bağlantıcılık, 290
- Bahdanau ilgisi, 587
- bant genişliği doyması, 752
- basit öngörü, 538
- bayat gradyanlar, 751
- Bayes bilgi kriteri (BIC), 278
- Bayes çıkarımı, 624
- Bayesçi Gauss karışım modeli, 281
- bayt çifti kodlaması, 572
- beklenen değer adımı, 273
- Beklenen Değeri Enbüyütme (EM)
  - algoritması, 273
- belirginlik çıktısı, 517
- belirsizlik örnekleme, 265
- bellek bant genişliği, 449
- bellek hücreleri, 533
- Bellman optimallik eşitliği, 665
- benzerlik ölçüsü, 20
- benzeşim, 247
- benzeşim yayılımı, 270
- benzetimli tavlama, 136
- Better Life Index, 21
- bileşenler, 43
- bilgi teorisi, 192
- bir sonraki cümle tahmini (NSP), 601
- bir-elemanı-bir kodlama, 73

- bir-elemanı-bir vektör, 460  
 BIRCH algoritması, 269  
 bire-karşı-bir (OvO) stratejisi, 111  
 bire-karşı-diğerleri (OvR) stratejisi, 111  
 bire-karşı-hepsi (OvA) stratejisi, 111  
 biriktirme katmanı, 486  
 biriktirme kerneli, 487  
 birim matris, 147  
 birinci dereceden kısmi türevler  
 (Jakobiyenler), 378  
 birincil problem, 180  
 bitleştirici ilgi, 587  
 biyolojik sınır hücreleri, 290  
 biyolojik sınır hücreleri (BNN), 290  
 Boltzmann makineleri, 821  
 boyut azaltma  
 amacı, 13  
 Ana Bileşenler Analizi (PCA),  
 229–240  
 boyut laneti, 225  
 diğer boyut azaltma teknikleri, 242  
 genel bakış, 224  
 LLE (Locally Linear Embedding),  
 240  
 yaklaşımlar, 226–229  
 boyut laneti, 225  
 bölge öneri ağı (RPN), 524  
 budama, 193  
 büyük sayılar yasası, 202  
 büzüştürme, 216  
  
 California Housing Prices veri seti, 41  
 callbacks, 329  
 canary testi, 727  
 CART eğitim algoritması, 189, 191  
 Colab Runtime, 738  
 Colaboratory (Colab), 737  
 Compute Unified Device Architecture  
 library (CUDA), 734  
 contrastive divergence, 823  
 CUDA Deep Neural Network library  
 (cuDNN), 734  
 cümle kodlayıcılar, 577  
  
 çalışma alanı oluşturmak, 48  
 çapa kutuları, 521  
 çapraz doğrulama, 36, 81, 99  
 çapraz entropi kaybı, 160, 305  
 çarpımsal ilgi, 587  
 çekirdek örnek, 266  
 çekişen DQN algoritması, 682  
 çekişmeli öğrenme, 527, 605  
 çevrim içi model, 679  
 çevrim içi öğrenme, 18, 98  
 çevrim içi SVM'ler, 183  
 çıkarsama, 26  
 çıktı geçidi, 550  
 çıktı katmanları, 298  
 çift çekişen DQN, 683  
 çift uçlu kuyruk, 675  
 çizge modu, 434  
 çoğunluk-oylaması sınıflandırıcısı, 201  
 çoğunluk-oylaması tahminleri, 199  
 çok başlı ilgi katmanı, 592, 595  
 çok boyutlu ölçekleme (MDS), 243  
 çok çıktı sınıflandırma, 119  
 çok değişkenli bağlanım problemleri, 44  
 çok değişkenli zaman serileri, 536  
 çok etiketli sınıflandırma, 117  
 çok katmanlı Perseptron (MLPs)  
 bağlanım MLP'leri, 303  
 sınıflandırma MLP'leri, 304  
 ve geri yayılım, 298–302  
 çok sınıflı sınıflandırma, 111  
 çok terimli (multinomial) sınıflandırıcı,  
 111  
 çok terimli lojistik bağlanım, 158  
 çok-görevli sınıflandırma, 325  
 çoklu arka uçlu Keras, 306  
 çoklu bağlanım problemleri, 44  
 çoklu çıktılar, 324  
 darboğaz katmanları, 498  
 Data API (TensorFlow)  
 dönüştürücüleri zincirleme hale  
 getirmek, 441  
 genel bakış, 440  
 tf.keras ile veri setlerini kullanmak,  
 450  
 veriyi karıştırmak, 443  
 veriyi önceden getirme, 448  
 veriyi önışleme, 446  
 yardımcı fonksiyon oluşturma, 447  
 DBSCAN, 266  
 Decision Stumps, 214  
 Deep Learning VM Images, 737  
 değer yineleme algoritması, 666  
 değişimsel çıkarım, 283  
 değişimsel otokodlayıcılar, 624–629  
 değişimsel parametreler, 283  
 değişirlik

- değişirlik oranı, 233
- korumak, 230
- değişirlik oranı, 233
- değişkenler, 405
- değişmezlik, 487
- demet arama, 583
- demet genişliği, 584
- denetimli öğrenme
  - görevleri, 9
  - kapsanan algoritmalar, 11
  - tanım, 9
- denetimsiz öğrenme
  - Gauss karışımı modeli (GMM), 270–285
  - genel bakış, 245
  - görevleri, 12
  - kapsanan algoritmalar, 11
  - kümeleme, 246–270
  - tanım, 11
  - üst üste yığılmış otokodlayıcıları kullanarak öğretim, 612–616
- denetimsiz öğretim, 368
- dengeşizlik, 196
- dense katmanı, 295
- derin evrişimsel GAN'lar, 636
- derin inanç ağları (DBNs), 15, 824
- Derin Nöroevrim, 338
- derin otokodlayıcılar, 609
- derin Q-Öğrenmesi
  - biçimleri, 679
  - çekişen DQN, 682
  - Çift DQN, 680
  - genel bakış, 672
  - öncelikli tecrübe yeniden oynatması, 681
  - sabit Q-Değer hedefleri, 679
  - uygulama, 674
- derin Q-Öğrenmesi (DQNs), 673, 692
- Derin Sinir Ağları (DNNs)
  - aşırı uydurmayı önlemek, 384–392
  - daha hızlı eniyileyciler, 370–384
  - genel bakış, 347
  - önceden eğitilmiş katmanları tekrar kullanmak, 364–370
  - tanım, xvii, 299
  - varsayılan yapılandırma, 393
  - yok olan/aşırı büyüyen gradyan problemi, 348–364
- derinlemesine ayrılabilir evrişim katmanı, 504
- derinlik birleştirme katmanı, 498
- derinlik yarıçapı, 497
- Destek Vektör Makinesi (SVM)
  - çevrim içi SVM'ler, 183
  - doğrusal olmayan SVM sınıflandırması, 168–174
  - doğrusal SVM sınıflandırma, 164
  - dual problem, 180, 810
  - eğitim amacı, 177
  - faizleri, 164
  - karar fonksiyonu ve tahminler, 176
  - Kernelli SVM'ler, 181
  - SVM bağlantısı, 174
- destek vektörü, 165
- dil modelleri, 599
- dinamik modeller, 326
- dinamik programlama, 666
- dinamik yerleştirici algoritması, 742
- Distribution Strategies API, 710, 754
- dizgi alt indisi kerneli, 172
- dizgi kernelleri, 172
- dizgi tensörler, 406, 828
- dizi-sonu (EoS) andıcı, 579, 592
- diziden vektöre ağlar, 534
- diziden-diziye modeller, 544
- diziler
  - bir zaman serisini tahmin etmek, 536–546
  - girdi ve çıktı dizileri, 534
  - için RNN'ler, 530
  - uzun dizileri ele almak, 546–557
- dizinin başlangıcı andıcı (SoS token), 571
- doğal dil işleme (NLP)
  - duygu analizi, 570–578
  - genel bakış, 559
  - için CNN'ler, 475
  - için kodlayıcı-kodçözücü ağı, 579–585
  - için RNN'ler, 530
  - ilgi mekanizmaları, 585–599
  - karakter RNN'si kullanarak metin üretimi, 560–570
  - uygulamaları, 370
  - yakın zamandaki yenilikler, 599
- doğrulama veri seti, 35
- doğruluk oranı
  - çapraz-doğrulama kullanarak ölçmek, 99
  - örnekleri, 3
  - tanım, 100
- doğrusal bağlanım modeli

- eğitim yaklaşımı, 122, 124
- genel bakış, 123
- hesaplama karmaşıklığı, 128
- normal denklem, 125
- doğrusal cebir, 123
- doğrusal diskriminant analizi (LDA), 243
- doğrusal model, 21
- doğrusal olmayan boyut azaltma (NLDR), 240
- doğrusal olmayan SVM sınıflandırması, 168–174
- doğrusal otokodlayıcılar, 607
- doğrusal SVM sınıflandırma, 164
- doyuma ulaşmayan aktivasyon fonksiyonları, 351
- dönüştürücü mimarisi, 590
- dönüştürücüler
  - afın dönüşüm, 642
  - dönüşüm iletim hatları, 77
  - özel, 75
  - tersine dönüşüm, 235
  - zincirleme hale getirmek, 441
- dönüştürücüleri zincirleme hale getirmek, 441
- DQN ajanları, 694
- dual problem, 180, 810
- dual sayılar, 816
- durgun nokta, 810
- durum bilgisi taşıyan metrik, 413
- durum-hareket değerleri, 666
- duyarlılık, 102–107
- duygu analizi
  - genel bakış, 570
  - maskeleme, 575
  - önceden eğitilmiş gömülmeleri tekrar kullanmak, 577
  - tanım, 560
- düz veri setleri, 564
- düzenleştirilmiş doğrusal modeller
  - Elastic Net, 151
  - genel bakış, 145
  - Lasso bağlanımı, 148
  - Ridge bağlanımı, 145
- düzenleştirme
  - büzüştürme tekniği, 216
  - için çoklu çıktı, 325
  - ile aşırı uydurmayı önlemek, 384–392
  - karar ağaçları için hiperparametreler, 193
  - tanım, 32
- düzenleştirme terimleri, 145
- düzensiz tensörler, 829
- düzleştirme terimi, 357
- eğitim örnekleri, 226
- eğitim sonrası nicemleme, 730
- eğitim veri seti, 3, 34, 224
- eğitim verisi
  - alakasız öznelilik, 30
  - aşırı uydurma, 31
  - düşük kalite, 30
  - eğitim veri seti oluşturma, 561
  - tanım, 3
  - temsilsiz edememesi, 28
  - yetersiz oluşu, 26
  - yetersiz uydurma, 33
- eğitim-doğrulama veri seti, 36
- eğitim/servis çarpıklığı, 470
- eğrinin altında kalan alan (AUC), 108
- eksi kanıt alt sınırı (ELBO), 283
- eksik etiketli öğrenme (ZSL), 600
- Elastic Net, 151
- ELU (exponential linear unit), 353–355
- en büyük olabilirlik kestirimi (MLE), 279
- en büyükleri biriktirme katmanı, 486
- en küçük-en büyük ölçekleme, 76
- enbüyütme adımı, 273
- enerji fonksiyonu, 821
- eniyleyiciler
  - AdaGrad, 374
  - Adam ve Nadam eniyilemesi, 376
  - birinci ve ikinci dereceden kısmi türevler, 378
  - daha hızlısını oluşturmak, 370
  - momentum eniyilemesi, 371
  - Nesterov hızlanan gradyanı, 372
  - öğrenme oranı çizgelemesi, 379
  - stokastik gradyan inişi (SGD), 98, 135
- entropi safsızlığı ölçütü, 192
- epoklar, 136, 300
- erken durdurma, 152
- eşdeğerlilik, 488
- eşik mantık birimi (threshold logic unit (TLU)), 294
- eşiksiz oylama, 203
- eşitlenmiş öğrenme oranı, 641
- eşitlik kısıtları, 810
- etiket, 9, 44, 249
- etiket yayılımı, 264

- evrişim katmanı
  - bellek gereksinimleri, 485
  - çoklu öznitelik haritalarını üst üste yığmak, 480
  - filtreler, 479
  - genel bakış, 477
  - TensorFlow uygulaması, 482
- evrişim kerneli, 479
- evrişimsel otokodlayıcılar, 617
- Evrişimsel Sinir Ağları (CNNs)
  - aktarmalı öğrenme için önceden eğitilmiş modeller, 512
  - anlamsal bölütleme, 524
  - biriktirme katmanı, 486
  - CNN mimarileri, 490–508
  - evrişim katmanı, 477–486
  - genel bakış, 475
  - görsel korteksin mimarisi, 476
  - keras kullanılarak ResNet-34, 508
  - keras'taki önceden eğitilmiş modeller, 510
  - nesne tespiti, 517–524
  - sınıflandırma ve konumlandırma, 515
- eylemsizlik, 253
- F1 skoru, 103
- fark alma, 540
- Fashion MNIST veri seti, 308, 612, 628
- Fast-MCD (minimum covariance determinant), 285
- fazladan ağaçlar sınıflandırıcısı, 209
- filtreler, 479
- foldlar, 99
- fonksiyon çizgesi, 836
- fonksiyon tanımı, 836
- Functional API, 322–326
- Gauss Dairesel Baz Fonksiyonu (RBF), 171
- Gauss karışımı modeli (GMM)
  - anormali ve yenilik tespiti için diğer algoritmalar, 284
  - Bayeşçi Gauss karışım modeli, 281
  - çizgesel gösterimi, 271
  - genel bakış, 270
  - kullanarak anormali tespiti, 276
  - küme sayısını seçmek, 278
  - türleri, 270
- geçit denetçileri, 551
- geçit yinelemeli birim (GRU) hücresi, 553
- genelleştirilmiş Lagrange, 810
- genelleştirme hatası, 34
- genetik algoritmalar, 649
- geniş sınır sınıflandırması, 165
- Geoffrey Hinton, xvii
- gerçek negatif oranı (TNR), 108
- gerçek pozitif oranı (TPR), 102
- gereksiz posta filtresi, 2
- gerektirim, 600
- geri mod autodiff, 817
- geri modda autodiff, 300
- geri yayılım, 298–302
- gevşek değişken, 178
- Gini safsızlığı ölçütü, 192
- girdi geçidi, 550
- girdi imzası, 835
- girdi katmanları, 298
- girdi sinir hücreleri, 295
- girdi ve çıktı dizileri, 534
- giriş yelpazesi (fan-in) ve çıkış yelpazesi (fan-out), 349
- gizli birimler, 822
- gizli değişkenler, 272
- gizli katmanlar
  - gizli katman başına düşen sinir ağı, 340
  - MLPs, 298
  - sayısı, 339
- global minimum, 130
- global ortalama biriktirme katmanı, 490
- Glorot ve He ilk değer ataması, 349
- Google Cloud Platform (GCP)
  - kullanan tahmin servisleri, 725–728
  - tahmin servisi oluşturmak, 720–724
- Google Cloud Storage (GCS), 722
- Google News 7B külliyyatı, 577
- GoogLeNet, 497
- gömülme, 74, 439, 462
- gömülme matrisi, 464
- gömülü cihazlar, 728
- gömülü rehber dil bilgileri, 603
- görsel ilgi, 589
- görselleştirme algoritmaları, 13
- görüntü bölütleme, 248, 259
- görüntü sınıflandırma
  - çok-görevli sınıflandırma, 325
  - Sequential API kullanarak, 308–320
- görüntü üretme, 527
- görünür birimler, 822
- gösterim öğrenme, 74, 462, 604

- gözetleme deliği bağlantıları, 552  
gözlemciler, 696  
gözlemlenebilir değişken, 272  
GPU'lar (graphics processing units)  
  aygıt üzerine işlemleri ve değişkenleri  
  yerleştirmek, 741  
  birden çok aygıt üzerinde paralel  
  çalıştırma, 743  
  Colaboratory (Colab), 737  
  GPU RAM'ini yönetmek, 739  
  GPU seçimi, 734  
  GPU'lu sanal makineler, 736  
  ile hesaplamaları hızlandırmak, 733  
  tek makineye eklemek, 734  
gradyan ağacı hızlandırma, 215  
gradyan hızlandırılmış bağlanım ağacı  
  (GBRT), 215  
gradyan hızlandırma, 215  
gradyan inişi  
  genel bakış, 122, 129  
  mini-yığın gradyan inişi, 138  
  stokastik gradyan inişi, 135  
  yığın gradyan inişi, 132  
gradyan kırpma, 363  
güçlü öğrenici, 201  
gürültülü veri, 21
- hareket adımı, 699  
hareket avantajı, 657  
hareketler  
  değerlendirme, 656  
  faýdalanma ve keşfetme dengesi, 655  
harmanlayıcı, 220  
harmonik ortalama, 103  
hassasiyet, 102  
hata analizi, 114  
hata matrisi, 100  
HDF5 formatı, 328  
He ilk değer ataması, 349  
Heaviside adım fonksiyonu, 294  
Hebb's kuralı, 296  
Hebbian öğrenmesi, 296  
hedef model, 679  
hesaplama çizgeleri, 398  
hırslı algoritma, 191  
hızlandırılmış K-ortalamalar, 254  
hızlandırma  
  AdaBoost, 211  
  genel bakış, 211  
  gradyan hızlandırma, 215  
  hinge kayıp fonksiyonu, 166, 184  
  hiperbolik tanjant fonksiyonu (tanh), 301  
  hiperdüzlem, 176  
  hiperparametreler  
    düzenleştirme hiperparametreleri,  
    193  
    eniyeleme için Python kütüphaneleri,  
    337  
    hiperparametre ayarı, 35, 83  
    öğrenme oranı, 129  
    sinir ağları için ince ayar, 335–343  
    tanım, 32  
  hipotez hızlandırma, 211  
  histogramlar, 54  
  Hiyerarşik DBSCAN (HDBSCAN), 269  
  hiyerarşik kümeleme algoritması, 12  
  holdout doğrulama, 35  
  Hopfield ağları, 820  
  Huber kaybı, 303, 407  
  Hyperas, 337  
  Hyperband, 338  
  Hyperopt, 337
- ısınma aşaması, 752  
iç içe geçmiş veri setleri, 564  
iki yönlü RNN'ler, 582  
iki yönlü yinelemeli katman, 583  
ikili ağaçlar, 189  
ikili sınıflandırıcılar, 98  
ikinci dereceden kısmi türevler (Hesseler),  
  378  
ileri beslemeli sinir ağları (FNNs), 299  
ileri mod autodiff, 815  
ileri yayılım, 300  
iletim hattı, 43, 451  
ilgi mekanizmaları  
  dönüştürücü mimarisi, 590  
  genel bakış, 585  
  görsel ilgi, 589  
  tanım, 560  
  ve açıklanabilirlik, 589  
ilişkilendirme kuralı öğrenmesi, 14  
ilişkili bellek ağları, 820  
ilk değer atama  
  Glorot ve He ilk değer ataması, 349  
  kütle merkezi ilk değer atama  
  metotları, 252  
  LeCun ilk değer ataması, 350  
  rastgele ilk değer atama, 129  
  Xavier ilk değer ataması, 349



- ilk giren ilk çıkar (FIFO) kuyrukları, 406  
 indirim çarpanı, 657  
 indirimsiz ödüller, 699  
 Internet Movie Database, 570  
 iris veri seti, 156  
 isomap algoritması, 243  
 istatistiksel önem, 193  
 istekli yürütme/istekli mod, 434  
 işlem içi iş parçacığı havuzu, 744  
 işlemler arası iş parçacığı havuzu, 744  
 izole çalışma alanı, 48  
 izole ormanlar algoritması, 285
- JupyterLab, 737  
 just-in-time (JIT) derleyici, 398
- K-ortalamalar  
 genel bakış, 248  
 girdi özniteliklerini ölçekleme, 259  
 görüntü bölütleme, 259  
 hızlandırılmış ve mini yığın  
 K-ortalamalar, 254  
 ile önışleme, 261  
 K-ortalamalar algoritması, 251  
 kısıtları, 258  
 kütle merkezi ilk değer atama  
 metotları, 252  
 optimum küme sayısını bulma, 255  
 önerilen iyileştirmeler, 253  
 sert ve yumuşak kümeleme, 250
- k-en yakın komşu bağlanımı, 25  
 K-parça çapraz-doğrulama, 81, 99  
 kalabalığın bilgeligi, 200  
 kalkülüs, 123  
 kapalı form çözüm, 125  
 kara kutu modelleri, 190  
 kara kutu stokastik değişimsel çıkarım  
 (BBSVI), 284  
 karakter RNN'leri (Char RNN'leri)  
 eğitim veri seti oluşturma, 561  
 genel bakış, 560  
 kullanımı, 566  
 oluşturma ve eğitmek, 565  
 Shakespeare tarzı metin üretmek, 566  
 sıralı bir veri setini bölmek, 562  
 sıralı veri setlerini ayırmak, 563  
 ve durum bilgisi taşıyan RNN'ler, 568
- karar ağaçları  
 bağlanım görevleri, 194  
 CART eğitim algoritması, 191  
 değerlendirme, 81  
 dengesizlik sorunu, 196  
 düzenleme hiperparametreleri,  
 193  
 eğitmek ve görselleştirmek, 186  
 faydaları, 186  
 Gini safsızlığına karşı entropi, 192  
 hesaplama karmaşıklığı, 192  
 sınıf olasılıklarını kestirmek, 190  
 tahminler yapmak, 188  
 karar fonksiyonu, 104  
 karar sınırları, 156  
 kararsız gradyanlar problemi, 546  
 Karesel Programlama (QP) problemleri,  
 179  
 karıştıran düzenleme, 644  
 Karush–Kuhn–Tucker (KKT) çarpanları,  
 811  
 kategorik dağılım, 272  
 kategorik öznitelikler  
 bir-elemanı-bir vektörleri kullanarak  
 kodlamak, 460  
 gömülmeleri kullanarak kodlama, 462  
 katman normalleştirme, 547  
 katmanlar  
 1B evrişimsel katmanlar, 555  
 adaptif örnek normalleştirme  
 (AdaIN), 642  
 biriktirme katmanı, 486  
 çıktı katmanı, 298  
 çok başlı ilgi katmanı, 592, 595  
 evrişim katmanı, 477–486  
 girdi katmanı, 298  
 gizli katmanlar, 298  
 iki yönlü yinelemeli katman, 583  
 maskelenmiş çok başlı ilgi katmanı,  
 592  
 mini yığın standart sapma katmanı,  
 640  
 ölçeklenmiş nokta çarpımı ilgi  
 katmanı, 595  
 önceden eğitilmiş olanları kullanmak,  
 364–370  
 yinelemeli, 531–535  
 yoğun (tam bağlantılı) katman, 295  
 katmanların hırslı eğitimi, 615  
 katmanların hırslı öğretilmesi, 369  
 kelime ağacı (WordTrees), 522  
 kelime andıçlama, 572  
 kelime gömülmeleri, 463

- kelime hazinesi, 460  
kelime hazinesi dışı kovalar, 460  
kendi kendine ilgi mekanizması, 592  
kendi kendini normalleştirme, 354  
Keras  
  alt düzey API'si, 403  
  çoklu arka uçlu Keras, 306  
  faydaları, xix  
  gradyan kırpma, 363  
  ile aktarmalı öğrenme, 366  
  ile modelleri kaydetmek ve geri yüklemek, 328  
  ile ResNet-34 uygulama, 508  
  ile seyreltme uygulama, 388  
  ile üst üste yığılmış otokodlayıcıları kullanmak, 609  
  ile veri seti yükleme, 308  
  ile yığın normalleştirmeyi uygulamak, 359  
  karmaşık mimariler, 328  
  keras'taki önceden eğitilmiş modelleri kullanmak, 510  
  keras.callbacks paketi, 330  
  keras.io'dan kod örneklerini kullanmak, 311  
  MLP'leri uygulamak, 306–334  
  önışleme katmanları, 467  
Keras Tüner, 337  
kernel hilesi, 170, 239  
kernel PCA (kPCA), 237–240  
kerneller, 182, 237, 399  
Kernelli SVM'ler, 181  
kesinlik, 101–107  
keşif politikası, 668, 672  
keşif veri seti, 61  
ki-kare testi, 193  
kısa-dönem bellek problemi, 549–557  
kısayol bağlantıları, 501  
kısıtlı Boltzmann makineleri (RBMs), 16, 369, 823  
kısıtlı eniyileme, 178  
kısmi türev, 132  
kodçözücüler, 534, 606  
kodlamalar, 604  
kodlayıcı-kodçözücü model, 534, 579–585  
kodlayıcılar, 534, 606  
konumlandırma, 515  
konumsal gömülmeler, 593  
konveks fonksiyon, 130  
kopt kütüphanesi, 337  
koşullu olasılık, 584  
kök düğümler, 188  
kredi atama problemi, 656  
kukla nitelik, 73  
kullanıcı tanımlı modeller  
  aktivasyon fonksiyonları, başlatıcılar, düzenleştirciler ve kısıtlar, 410  
  Autodiff kullanarak gradyanları hesaplamak, 424, 813–819  
  eğitim döngüleri, 428  
  hakkında, 397  
  katmanlar, 415  
  kaydetme ve geri yükleme, 408  
  kayıp fonksiyonları, 407  
  kayıplar ve ölçütler, 421  
  modeller, 419  
  ölçütler, 410  
Kullback–Leibler ıraksaması, 161  
kuvvet çizelgelemesi, 380  
kuyruğu ağır histogramlar, 56  
kuyruklar, 406, 833  
küme tanımlaması, 756  
kümeleme algoritmaları  
  amacı, 245  
  DBSCAN, 266  
  diğer algoritmalar, 269  
  genel bakış, 246  
  K-ortalamlar, 248–259  
  uygulamaları, 12, 247  
kümeler, 406, 832  
kütle merkezi, 248  
Lagrange çarpanları, 810  
Lasso bağlanımı, 148  
LeCun ilk değer ataması, 350  
LeNet-5, 493  
Levenshtein mesafesi, 172  
liblinear kütüphanesi, 173  
libsvm kütüphanesi, 173  
LLE (Locally Linear Embedding), 240  
Lloyd–Forgy algoritması, 248  
Local Outlier Factor (LOF), 285  
log-kayıp, 155  
logical GPU devices, 739  
lojistik bağlanım  
  eğitim ve maliyet fonksiyonu, 155  
  genel bakış, 153  
  ile sınıflandırma, 10  
  karar sınırları, 156  
  olasılık tahmini, 153

- Softmax bağlanımı, 158
- lojistik fonksiyon, 153, 303–305, 315, 348
- lojit, 154
- Luong ilgisi, 587
- Makine Öğrenmesi (ML)
- ek kaynaklar, xxii
  - faydaları, 4
  - genel bakış, 33
  - hakkında makaleler, 401
  - kapsanan başlıklar, xx
  - notasyon, 45, 176
  - öğrenme yaklaşımı, xviii
  - öğrenmeniz için gerekli ön koşullar, xix
  - tanım, 2
  - tarihi, xvii
  - test ve doğrulama, 34–37
  - türleri, 9–26
  - uygulamaları, xviii, 7
  - zorlukları, 26–34
- makine öğrenmesi proje kontrol listesi, 42, 804
- maliyet fonksiyonu
- çapraz entropi kaybı, 160
  - görevi, 23
  - hinge kaybı, 166, 184
  - ortalama karesel hata, 130, 303, 321, 407, 607, 610, 621, 677
  - ortalama mutlak hata (MAE), 303
- Manhattan normu, 47
- manifold hipotezi, 229
- manifold öğrenme, 228
- manifold varsayımı, 229
- mantıksal hesaplamalar, 293
- Markov karar süreçleri (MDP), 663–668
- Markov zincirleri, 663
- Mask R-CNN, 527
- maske tensörleri, 575
- maskeleme, 575
- maskelenmiş çok başlı ilgi katmanı, 592
- maskelenmiş dil modeli, 601
- max-norm düzenleme, 392
- Mean-Shift algoritması, 269
- merak-tabanlı keşif, 707
- Mercer koşulları, 182
- Mercer teoremi, 182
- meta çizge, 713
- meta öğrenici, 220
- metin üretimi
- eğitim veri seti oluşturma, 561
  - genel bakış, 560
  - için kullanılan modeller, 566
  - için model oluşturmak ve eğitmek, 565
  - Shakespeare tarzı metin üretmek, 566
  - sıralı bir veri setini bölmek, 562
  - sıralı veri setlerini ayırmak, 563
  - ve durum bilgisi taşıyan RNN'ler, 568
- Microsoft Cognitive Toolkit (CNTK), 306
- mini yığın ayrıştırma, 636
- mini yığın standart sapma katmanı, 640
- mini-yığın gradyan inişi, 138
- mini-yığın K-ortalamar, 254
- mini-yığın, 18, 138
- ML Engine, 722
- MNIST veri seti, 95
- mobil cihazlar, 728
- mod çöküşü, 635
- model, 397
- aktarmalı öğrenme için önceden eğitilmiş modeller, 512
  - Altsınıflama API kullanarak dinamik model, 326
  - birden çok aygıt üzerinde eğitme, 746–763
  - dizin-diziye modeller, 544
  - eğitim, 23, 79, 122
  - Fonksiyonel API kullanarak karmaşık modeller, 322–326
  - geri çağırma kullanmak, 329
  - görselleştirme için TensorBoard kullanmak, 331
  - ince-ayar yapmak, 83–89
  - kaydetmek ve geri yüklemek, 328
  - keras'taki önceden eğitilmiş modeller, 510
  - nedensel modeller, 544
  - parametrik modele karşı parametrik olmayan model, 193
  - seyrek modelleri eğitmek, 378
  - tanım, 23
  - TensorFlow ile kullanıcı tanımlı modeller, 407–431
- model paralelleştirme, 746
- model parametreleri, 22
- model seçimi, 21, 35, 79
- model tabanlı öğrenme, 20
- modeli eğitmek
- doğrusal bağlanım, 123–129

- düzenleştirilmiş doğrusal modeller, 145–153
- genel bakış, 122
- gradyan inişi, 129–138
- lojistik bağlanım, 153–162
- öğrenme eğrileri, 141–145
- örnek proje, 79
- polinom bağlanımı, 139–141
- tanım, 23
- modelleri kaydetmek ve geri yüklemek, 328
- modüller, 577
- momentum eniyilemesi, 371
- momentum vektörü, 371
- Monte Carlo (MC) seyreltme, 389
- müşteri bölütleme, 247
- Nash dengesi, 634
- nedensel modeller, 544
- nesne tespiti
  - genel bakış, 517
  - tamamen evrimsel ağlar (FCNs), 518
  - You Only Look Once (YOLO), 521
- Nesterov hızlanan gradyanı, 372
- Nesterov momentum eniyilemesi, 372
- Newton fark katsayısı, 814
- nicemlemeye duyarlı eğitim, 731
- nitelik, 10
- niteliklerin benzerliği, 171
- nokta çarpımı, 587
- non-max bastırma, 517
- normal denklem, 125
- normalizasyon, 76, 356, 641
- normalleştirilmiş üstel fonksiyon, 158
- NP-Tam problem, 191
- NumPy
  - array\_split() fonksiyonu, 236
  - büyük dizileri serileştirmek, 83
  - inv() fonksiyonu, 126
  - kurulumu, 48
  - memmap sınıfı, 237
  - randint() fonksiyonu, 119
  - svd() fonksiyonu, 232
  - TensorFlow'u NumPy gibi kullanmak, 401–407
  - yoğun diziler, 74
- NVIDIA Collective Communications Library (NCCL), 755
- Nvidia GPU kartları, 734
- odds oranının logaritması, 154
- olabilirlik fonksiyonu, 278
- olasılık yoğunluk fonksiyonunu (PDF), 246, 274
- olasılıksal otokodlayıcılar, 624
- olay dosyaları, 331
- OpenAI Gym, 650–654
- Optik Karakter Tanıma (OCR), 2
- optimal durum değeri, 665
- orijinal uzay, 237
- orta alan değişimsel çıkarım, 283
- ortalama biriktirme katmanı, 489
- Ortalama KareSEL Hata Karekökü (RMSE), 45, 130
- ortalama kesinlik (AP), 523
- ortalama kodlama, 624
- Ortalama Mutlak Hata (MAE), 46
- ortalama ortalama kesinlik (mAP), 522
- otokodlayıcılar
  - arıtan, 619
  - değişimsel, 624–629
  - evrimsel, 617
  - genel bakış, 604
  - olasılıksal, 624
  - otokodlayıcılara karşı üretken çekişmeli ağlar, 604
  - parçaları, 606
  - seyrek, 620
  - tamamlanmamış, 607
  - tamamlanmamış doğrusal otokodlayıcı ile PCA, 607
  - üretken, 624
  - üst üste yığılmış otokodlayıcı kullanan denetimsiz öğretim, 612–616
  - üst üste yığılmış otokodlayıcılar, 609–612
  - verimli veri gösterimleri, 605
  - yinelemeli, 618
- otomatik türev alma (autodiff), 300, 424, 813–819
- otonom sürüş sistemleri, 530
- otoregresif tamamlanmış hareketli ortalama (ARIMA) modelleri, 539
- p (ardıl) dağılımı, 282
- p (öncül) dağılımı, 281
- p-değeri, 193
- öğrenme eğrileri, 141–145

- öğrenme oranı, 19, 129, 341, 641
- öğrenme oranı çizgelemesi, 379
- öğrenme zamanlayıcısı, 136, 380
- Öklid normu, 46
- ölçeklenmiş nokta çarpımı ilgi katmanı, 595
- ölçütler
  - doğruluk oranı, 410
  - duyarlılık, 102–107
  - eğrinin altında kalan alan (AUC), 108
  - F1 skoru, 103
  - hata matrisi, 100
  - kesinlik, 101–107
  - ortalama karesel hata, 195, 538
  - ortalama mutlak hata (MAE), 303
  - ortalama ortalama kesinlik, 522
  - RMSE, 45
  - ROC eğrisi, 108
- ölen ReLU problemi, 351
- ölümcül unutmama, 678
- ön-görüntü, 239
- ön-görüntüyü yeniden oluşturmak, 239
- öncelikli tecrübe yeniden oynatması (PER), 681
- öğretim
  - aktarmalı öğrenme için ön eğitim, 512
  - denetimsiz öğretim, 368
  - katmanların hırsh öğretilmesi, 369
  - keras'taki modeller, 510
  - önceden eğitilmiş gömülmeleri tekrar kullanmak, 577
  - önceden eğitilmiş katmanları tekrar kullanmak, 364–370
  - üst üste yığılmış otokodlayıcıları kullanmak, 612–616
  - yardımcı bir görev üzerine, 370
- önem örnekleme (IS), 681
- öneri sistemleri, 247
- önergeler mantığı, 290
- öngörü, 536
- önişleme, 261, 458–469
- örnek bölütleme, 259, 527
- örnek dışı hata, 34
- örnek proje
  - genel bakış, 40
  - gerçek veri ile çalışmak, 40
  - makine öğrenmesi proje kontrol listesi, 42, 804
  - model seçimi ve eğitimi, 79, 807
  - modele ince ayar yapmak, 83–89, 808
  - performans ölçütünü seçmek, 45
  - proje amaçları, 41
  - sisteminizi çalıştırmak, takip etmek ve bakımı yapmak, 89, 809
  - sorunu çerçevelemek, 42, 804
  - varsayımları doğrulamak, 47
  - veri görselleştirme, 61–68, 806
  - veriyi hazırlama, 68–79, 806
  - veriyi indirmek, 47–61, 805
- örnek tabanlı öğrenme, 20, 25
- örneklem gürültüsü, 29
- örneklem yanlılığı, 29
- örneklenmiş softmax tekniği, 581
- örüntü eşleştirme, 606
- özetler (TensorFlow), 331
- öznitelik, 10
- öznitelik çıkarma, 30
- öznitelik haritası, 239, 480
- öznitelik mühendisliği, 30
- öznitelik ölçekleme, 76
- öznitelik seçme, 30
- öznitelik uzayı, 237
- öznitelik vektörü, 124
- parametre matrisi, 159
- parametre sunucuları, 750
- parametre uzayı, 132
- parametre vektörü, 124
- parametre verimi, 339
- parametrik modeller, 193
- parametrik olmayan modeller, 193
- parametrik sızıntılı ReLU (PReLU), 352
- parçalı sabit çizgeleme, 381
- Pearson r korelasyon katsayısı, 64
- pekiştirmeli öğrenme (RL)
  - derin Q-Öğrenmesi, 672–679
  - genel bakış, 646
  - hareketleri değerlendirme, 656
  - için algoritmalar, 705
  - Markov karar süreçleri, 663–668
  - OpenAI Gym, 650–654
  - ödülleri eniyilemek, 647
  - politika araması, 649
  - politika gradyanları, 658–663
  - Q-Öğrenmesi, 669–673
  - sinir ağı politikaları, 655
  - TF-Agents kütüphanesi, 683–705
  - zamansal fark öğrenimi, 668
- performans çizgelemesi, 381

- performans ölçütleri, 99
- perseptron (algılayıcı), 294–298
- Perseptron yakınsama teoremi, 297
- piksel odaklı normleştirme katmanı, 641
- polinom bağlanımı, 122, 139
- polinom kerneli, 182
- polinomsal nitelik, 169
- politika araması, 649
- politika gradyanları (PG), 650, 658–663
- politika içi algoritmalar, 671
- politika parametreleri, 649
- politika uzayı, 649
- politika-dışı algoritma, 671
- politikalar, 649
- projeksiyon, 226
- proksimal politika eniyilemesi (PPO), 707
- protokol tamponları (protobufs), 452
- pürüzlü tensörler, 406
- PyTorch kütüphanesi, 307
- Q-Değer yineleme, 666
- Q-Değerleri, 666
- Q-Öğrenmesi
  - genel bakış, 669
  - keşif politikası, 672
  - uygulama, 670
  - yaklaşık Q-Öğrenmesi ve derin Q-Öğrenmesi, 672
- Rainbow ajamı, 683
- rastgele hâle getirilmiş sızıntılı ReLU (RReLU), 352
- rastgele ilk değer atama, 129
- rastgele ormanlar
  - faýdaları, 200
  - fazladan ağaçlar, 209
  - genel bakış, 208
  - nitelik önemi, 210
- rastgele parçalar ve rastgele alt uzaylar, 208
- rastgele PCA, 236
- rastgele projeksiyonlar, 242
- Rectified Linear Unit function (ReLU), 302–303
- REINFORCE algoritmaları, 658
- ReLU (Rectified Linear Unit function), 302–303
- renk bölütleme, 259
- renk kanalları, 481
- ResNet (Residual Network), 501
- ResNet-34 CNN, 508
- Ridge bağlanımı, 145
- RMSPProp, 375
- sabit Q-Değer hedefleri, 679
- safsızlık, 188, 192
- sahte nicemleme, 731
- sahtekârlık tespiti, 247
- saklı gösterimler, 604
- saklı kayıp, 625
- SAMME (Stagewise Additive Modeling using a Multiclass Exponential loss function), 214
- saniyedeki sorgu sayısı (QPS), 709
- SavedModel formatı, 711
- Scaled Exponential Linear Unit (SELU) fonksiyonu, 354–355, 389
- Scikit-Optimize, 338
- Scikit-Learn
  - anomalı ve yenilik tespiti, 284
  - artırımlı eğitim, 218
  - bir-elemanı-bir vektör, 73
  - CART eğitim algoritması, 189, 191
  - cross\_val\_score() fonksiyonu, 99
  - DecisionTreeRegressor sınıfı, 194
  - doğrusal bağlanım kullanımı, 127
  - doğrusal model kullanımı, 24
  - dönüştürücü dizileri, 77
  - dönüştürücüler, 75
  - eksik değerleri halletmek, 69
  - ExtraTreesClassifier sınıfı, 209
  - faýdaları, xviii
  - GBRT topluluk eğitimi, 216
  - GridSearchCV, 84
  - ile boyut azaltma, 242
  - ile veri merkezleme, 232
  - ile verileri önceden sıralama, 192
  - ile yeniden oluşturma, 239
  - IncrementalPCA sınıfı, 237
  - K-parça çapraz-doğrulama özelliği, 81
  - KernelPCA sınıfı, 237
  - kullanılan AdaBoost versiyonu, 214
  - kurulumu, 48
  - kümeleme algoritmaları, 269
  - LLE (Locally Linear Embedding), 240
  - max\_depth hiperparametresi, 194
  - mean\_squared\_error fonksiyonu, 80

- metinleri sayısal niteliklere çevirmek, 73
- modelleri kaydetmek, 83
- nitelik önem skoru, 210
- oylama sınıflandırıcısı, 202
- öznitelik ölçekleme, 165
- PCA kullanımı, 233
- Perceptron sınıfı, 297
- random\_state hiperparametresi, 197
- rastgele PCA algoritması, 236
- SGDClassifier sınıfı, 98
- sınıflandırıcı ölçütlerini hesaplamak, 102
- sisteminizi çalıştırmak, takip etmek ve bakımını yapmak, 89
- SVM modelleri, 166
- SVM sınıflandırma sınıfları, 174
- tabakalı örnekleme kullanımı, 60
- tam SVD yaklaşımı, 236
- tasarımı, 70
- tolerans hiperparametresi, 173
- torba-dışı değerlendirme, 206
- torbalama ve yapıştırma, 206
- veri seti sözlük yapısı, 95
- veri setini birkaç alt kümeyle bölmek, 59
- SE bloğu, 506
- SE-Inception, 506
- SE-ResNet, 506
- self-organizing maps (SOMs), 826
- SELU (Scaled Exponential Linear Unit) fonksiyonu, 350
- sembolik tensörler, 434, 836
- sembolik türev alma, 816
- SENet (Squeeze-and-Excitation Network), 506
- senkron güncellemeler, 750
- SequenceExample protobuf (TensorFlow), 457
- Sequential API
  - kullanılan görüntü sınıflandırıcıları, 308–320
  - MLP kullanarak bağlanım, 320
- sert kümeleme, 250
- sert sınır sınıflandırması, 165
- sert-oylama sınıflandırıcısı, 201
- servis hesabı, 725
- ses tanıma, 475
- seyrek matris, 73
- seyrek modeller, 378
- seyrek otokodlayıcılar, 620
- seyrek tensörler, 406, 830
- seyreklik, 620
- seyreklik kaybı, 621
- seyreltme, 386
- Shannon'un bilgi teorisi, 192
- sıcaklık
  - Boltzmann makinelerinde, 822
  - metin üretiminde sıcaklık, 567
- sıfır doldurma (zero padding), 478
- sıfır hipotezi, 193
- sıkıştırılan veriyi açmak, 235
- sıkıştırma, 235
- sınıflandırma problemleri
  - AdaBoost sınıflandırıcısı, 211
  - çok çıktılı sınıflandırma, 119
  - çok etiketli sınıflandırma, 117
  - çok sınıflı sınıflandırma, 111
  - çok-görevli sınıflandırma, 325
  - doğrusal olmayan SVM sınıflandırması, 168–174
  - doğrusal SVM sınıflandırma, 164
  - fazladan ağaçlar sınıflandırıcısı, 209
  - geniş sınır sınıflandırması, 165
  - hata analizi, 114
  - ikili sınıflandırıcılar, 98
  - MNIST veri seti, 95
  - oylama sınıflandırıcıları, 200
  - örnekler, 9
  - performans ölçütleri, 99–111
  - Sequential API kullanarak görüntü sınıflandırma, 308–320
  - sert sınır sınıflandırması, 165
  - sınıflandırma MLP'leri, 304
  - sınıflandırma ve konumlandırma, 515
  - yumuşak sınır sınıflandırması, 165
- Sınıflandırma ve Bağlanım Ağacı (CART), 189, 191
- sınır geçişleri, 703
- sınır ihlalleri, 165
- sınırlayıcı çerçeve öncülleri, 521
- sıradan ifadeler, 572
- sıralı olmayan sinir ağları, 322
- sızıntılı ReLU fonksiyonu, 352
- sigmoid aktivasyon fonksiyonu, 153, 303–305, 315, 348
- sigmoid kerneli, 182
- silhouette diyagramı, 257
- silhouette katsayısı, 257
- silhouette skoru, 257

- simetriyi kırmak, 301
- simülasyonu yapılmış ortam, 651
- sinir hücreleri
  - biyolojiktan yapaya, 290–305
  - girdi sinir hücreleri, 295
  - giriş yelpazesi (fan-in) ve çıkış yelpazesi (fan-out), 349
  - gizli katman başına düşen sinir hücresi, 340
  - sinir hücreleriyle mantıksal hesaplamalar, 293
  - stokastik nöronlar, 821
  - yanlılık sinir hücreleri, 295
  - yinelemeli sinir hücreleri, 531–535
- sinirsel makine çevirisi (NMT), 579–585, 599
- Sklearn-Deap, 338
- Softmax bağlantımı, 158
- Softmax fonksiyonu, 158, 305, 311, 500, 513, 519, 580
- softplus aktivasyon fonksiyonu, 303
- sohbet robotu, 559
- somut fonksiyon, 835
- sonlu farklar yaklaşımı, 814
- sonsal enbüyütme (MAP) kestirimi, 280
- sorular ve yorumlar, 764
- sözcük torbası, 468
- Spearment kütüphanesi, 338
- spektral kümeleme, 270
- standart korelasyon katsayısı, 64
- stil aktarımı, 642
- stil karıştırma, 644
- stokastik gradyan hızlandırma, 219
- stokastik gradyan inişi (SGD), 98, 135
- stokastik nöronlar, 821
- stokastik politika, 649
- StyleGANs, 604, 642
- sütun vektörü, 124
- t-dağıtılmış stokastik komşu gömülme (t-SNE), 243
- şeffaf kutu modelleri, 190
- tabakalı örnekleme, 59
- tahmin etmek, 536
- tahmin problemleri, 9, 19, 200
- tahmin servisi
  - GCP AI ile oluşturmak, 720–724
  - kullanımı, 725–728
- Talos kütüphanesi, 337
- tam bağlantılı katman, 295
- tam gradyan inişi, 133
- tamamen evrişimsel ağlar (FCNs), 518
- tamamen özelleşmiş model mimarisi, 23
- tamamlanmamış otokodlayıcı, 607
- tamamlanmışın üstünde otokodlayıcılar, 618
- tampon-karıştırma yaklaşımı, 444
- tanıma ağı, 606
- TD hatası, 669
- TD hedefi, 669
- tecrübe yeniden oynatması, 635
- tek değişkenli bağlantım problemleri, 44
- tek değişkenli zaman serileri, 536
- tek seferde öğrenme, 527
- tek-sınıflı SVM algoritması, 285
- Tekil Değer Ayrışımı (SVD), 128, 231
- temel hücreler, 533
- TensorBoard, 331
- TensorFlow Addons, 581
- TensorFlow Extended (TFX), 470
- TensorFlow Hub, 400, 577
- TensorFlow kümesi, 756
- TensorFlow Lite, 400
- TensorFlow Model İyileme Araç Takımı (TF-MOT), 379
- TensorFlow Playground, 305
- TensorFlow'un temelleri
  - faýdaları, xix, 397
  - işletim sistemi uyumluluğu, 400
  - kapsanan versiyonlar, 397
  - kurulumu, 307
  - kütüphane ekosistemi, 400
  - mimari, 399
  - özellikler, 398
  - topluluk desteği, 401
  - ve PyTorch kütüphanesi, 307
  - yardım almak, 401
- TensorFlow, CNN'ler
  - biriktirme katmanı, 488
  - evrişim katmanları, 482
  - evrişimsel işlemler, 526
- TensorFlow, fonksiyonlar ve çizgeler
  - AutoGraph ve izleme, 434, 835–843
  - genel bakış, 431
  - TF Fonksiyonu kuralları, 435
- TensorFlow, modelleri ölçekleyerek dağıtma
  - genel bakış, 709
  - hesaplamaları hızlandırmak için GPU'ları kullanmak, 733–746



- mobil ve gömülü cihazlara dağıtma, 728–732
  - modelleri birden çok aygıt üzerinde eğitime, 746–763
  - TensorFlow modellerini servis etmek, 710–728
- TensorFlow, NumPy benzeri işlemler
  - değişkenler, 405
  - diğer veri yapıları, 406
  - tensorler ve işlemler, 401
  - tensorler ve NumPy, 404
  - tip dönüşümü, 404
- TensorFlow, özel modeller ve eğitim
  - aktivasyon fonksiyonları, başlatıcılar, düzenleştirciler ve kısıtlar, 410
  - Autodiff kullanarak gradyanları hesaplamak, 424, 813–819
  - eğitim döngüleri, 428
  - hakkında, 397
  - katmanlar, 415
  - kaydetme ve geri yükleme, 408
  - kayıp fonksiyonları, 407
  - kayıplar ve ölçütler, 421
  - modeller, 419
  - öğrenme oranı çizgelemesini uygulamak, 383
  - ölçütler, 410
  - özel veri yapıları, 828–834
- TensorFlow, veriyi yüklemek ve önışlemek
  - Data API'si, 440–451
  - genel bakış, 439
  - girdi özniteliklerini önışlemek, 458–469
  - TensorFlow veri setleri (TFDS) projesi, 471
  - TF Dönüşümü, 469
  - TFRecord formatı, 451–458
- TensorFlow.js, 400
- tensor dizileri, 406, 831
- tensorler, 401
- teorik bilgi kriteri, 278
- termal denge, 822
- tersine dönüşüm, 235
- test ve doğrulama
  - hiperparametre ayarı, 35
  - model seçimi, 35
  - veri uyumsuzluğu, 36
- test veri seti, 34, 57
- TF Dönüşümü (tf.Transform), 439, 469
- TF Fonksiyonları
  - kurallar, 435
  - tarafından oluşturulan çizgeler, 835–843
- TF Veri Setleri (TFDS), 440, 471
- TF-Agents kütüphanesi
  - derin Q-ağı (DQNs), 692
  - DQN ajanları, 694
  - eğitim döngüleri, 704
  - eğitim mimarisi, 690
  - eğitim ölçütleri, 698
  - genel bakış, 683
  - kurulumu, 683
  - ortam özellikleri, 686
  - ortam sarmalayıcıları, 687
  - ortamları, 685
  - toplama sürücüsü, 699
  - veri setleri, 701
  - yeniden oynatma tamponu ve gözlemci, 696
- tf.keras, 307, 383, 450
- tf.summary paketi, 334
- TF.Text kütüphanesi, 572
- TFRecord formatı
  - Example'ları yüklemek ve ayrıştırmak, 456
  - genel bakış, 451
  - protokol tamponları (protobufs), 452
  - SequenceExample Protobuf'ını kullanan listelerin listeleri, 457
  - sıkıştırılmış TFRecord dosyaları, 452
  - TensorFlow protobufs, 454
- Theano, 306
- Tikhonov düzenleştirme, 145
- tip dönüşümü, 404
- tolerans, 134
- toplama politikası, 690
- toplama ilgi, 587
- topluluk metodu, 200
- topluluk öğrenme
  - en iyi kullanımı, 202
  - faýdaları, 82
  - hızlandırma, 211–220
  - oylama sınıflandırıcıları, 200
  - örnekleri, 200
  - rastgele ormanlar, 200, 208
  - rastgele parçalar ve rastgele alt uzaylar, 208
  - tanım, 200
  - torbalama ve yapıştırma, 204–207
  - yığıma, 220

- torbalama ve yapıştırma
  - genel bakış, 204
  - torba-dışı değerlendirme, 206
- TPU'lar (tensor processing units), 399
- transpozu alınmış evrişimsel katman, 524
- Turing testi, 559
- tutma olasılığı, 387
- tümler gevşeklik, 811
- türevleri elle almak, 813
  
- unutma geçidi, 550
- uyarlanır moment kestirimi, 376
- uyarlanır öğrenme oranı, 374, 375
- uydurma fonksiyonu, 23
- uzun diziler
  - genel bakış, 546
  - kararsız gradyanlar problemi, 546
  - kısa-dönem bellek problemi, 549–557
- uzun ömürlü kısa dönem bellek (LSTM) hücreleri, 549
  
- üretici modeller, 274, 604, 822
- üreticiler, 605
- üretken ağ, 606
- üretken çekişmeli ağlar (GANs)
  - derin evrişimsel GAN'lar, 636
  - eğitmenin zorlukları, 634
  - genel bakış, 630
  - otokodlayıcılara karşılaştırma, 604
  - StyleGANs, 642
  - sürekli gelişen GAN'lar, 639
  - uygulamaları, 604
- üretken otokodlayıcılar, 624
- üst üste yığılmış arıtan otokodlayıcılar, 619
- üst üste yığılmış otokodlayıcılar
  - Fashion MNIST veri setini görselleştirmek, 612
  - genel bakış, 609
  - keras ile kullanmak, 609
  - kullanılan denetimsiz öneğitim, 612–616
  - üst üste yığılmış arıtan otokodlayıcılar, 619
  - yeniden oluşturmayı görselleştirme, 611
- üstel çizelgeleme, 380
- üstel doğrusal birim (ELU: Exponential Linear Unit), 353–355
- vektörden diziye ağlar, 534
- vektörler
  - alt gradyan vektörü, 150
  - momentum vektörü, 371
  - öznitelik vektörü, 124
  - parametre vektörü, 124
  - sütun vektörü, 124
- veri, 40
  - algoritmalara karşı önemi, 27
  - asimetrik veri seti, 100
  - boyut azaltımı, 233
  - California Housing Prices veri seti, 41
  - coğrafi veri, 62
  - düz veri setleri, 564
  - eğitim veri seti oluşturma, 561
  - Fashion MNIST veri seti, 308, 612, 628
  - Google News 7B külliyatı, 577
  - gürültülü veri, 21
  - iç içe geçmiş veri setleri, 564
  - indirme, 51
  - Internet Movie Database, 570
  - iris veri seti, 156
  - karıştırmak, 443
  - kaynakları, 40
  - kümeleme ile analiz, 247
  - MNIST veri seti, 95
  - önceden getirme, 448
  - önışleme, 261, 446, 458–469
  - seyrek modelleri eğitmek, 378
  - sıkıştırılan veriyi açmak, 235
  - sıkıştırma, 235
  - sıralı bir veri setini bölmek, 562
  - sıralı veri setlerini ayırmak, 563
  - TensorFlow ile yüklemek ve önışlemek, 439–472
  - tf.keras ile veri setlerini kullanmak, 450
  - verimli veri gösterimleri, 605
  - yardımcı fonksiyon oluşturma, 447
  - yeniden oluşturma hatası, 235
- veri çeşitleme, 495
- veri görselleştirme
  - boyut azaltma, 224
  - coğrafi veri, 62
  - Fashion MNIST veri setini görselleştirmek, 612
  - korelasyonları hesaplamak, 64
  - nitelik kombinasyonları, 66
  - TensorBoard kullanarak, 331
  - test, eğitim ve keşif veri seti, 61

- yeniden oluşturmayı görselleştirme, 611
- veri gözetleme ön yargısı, 57
- veri hazırlama
  - dönüşüm iletim hatları, 77
  - fonksiyonlarla yapmanın faydaları, 68
  - metinsel ve kategorik nitelikler, 72
  - özel dönüştürücüler, 75
  - öznitelik ölçekleme, 76
  - veri temizleme, 69
- veri paralelleştirme, 746, 749
- veri setinin rotasyonu, 196
- veri setleri, 440
- verimsiz örnekleme, 663
- VGGNet, 501
- virtual GPU device, 739
  
- WaveNet, 530, 556
- Wide&Deep (Geniş ve Derin) sinir ağları, 322
  
- Xavier ilk değer ataması, 349
- Xception (Extreme Inception), 504
- XGBoost, 219
- XOR sınıflandırma problemi, 298
  
- yakınsama, 129
- yaklaşık Q-Öğrenmesi, 672
- yanıltıcı örüntüler, 821
- yanlılık sinir hücreleri, 295
- yanlılık terimi, 123
- yanlılık/değişirlik dengesi, 145
- yanlış pozitif oranı (FPR), 108
- yansıtılmış strateji, 749
- yapay sinir ağları (ANNs)
  - biyolojik sinir hücrelerinden yapay sinir hücrelerine, 290–305
  - Boltzmann makineleri, 821
  - genel bakış, 289
  - Hopfield ağları, 820
  - için hiperparametrelere ince ayar yapmak, 335–343
  - keras ile MLP'leri uygulamak, 306–334
  - kısıtlandırılmış Boltzmann makineleri (RBMs), 823
  - self-organizing maps (SOMs), 826
- yapay sinir hücreleri, 293
- yapıştırma, 204
- yaprak düğümler, 188
- yardımcı fonksiyon, 23
- yardımcı fonksiyonlar, 447
- yarı denetimli öğrenme, 370
  - için kümeleme algoritmaları, 247, 263
  - örnekleri, 15
  - tanım, 15
- yedek kopya, 751
- yeniden oluşturma hatası, 235
- yeniden oluşturma kaybı, 422, 607
- yeniden oluşturmalar, 607
- yeniden oynatma belleği, 674
- yeniden oynatma tamponu, 674, 691, 696
- yenilik tespiti, 14, 277, 284
- yerel minimum, 130
- yerel tepki normalleştirmesi, 496
- yetersiz uydurma, 33
- yetersiz-çekirdek-öğrenme, 18
- yığın büyüklüğü, 341
- yığın gradyan inişi, 132
- yığın normalleştirme (BN: Batch Normalization), 356
- yığın öğrenme, 17
- yığınlanmış hareket adımı, 700
- yığınlanmış yörünge, 700
- yığınlanmış zaman adımı, 700
- yığımsal kümeleme, 269
- yığma, 220
- yinelemeli otokodlayıcılar, 618
- yinelemeli sinir ağları (RNNs)
  - bir zaman serisini tahmin etmek, 536–545
  - durum bilgisi taşıyan ve taşımayan, 559, 568
  - eğitim, 535
  - genel bakış, 530
  - iki yönlü RNN'ler, 582
  - karakter RNN'leri kullanarak metin üretimi, 560–570
  - uzun dizileri ele almak, 546–557
  - yinelemeli sinir hücreleri ve katmanları, 531–535
- yinelemeli sinir hücreleri, 531
- yoğun vektörler, 593
- yoğunluk kestirimi, 246, 275
- yok olan/aşırı büyüyen gradyan problemi, 348–364
- yorumlar ve sorular, 764
- You Only Look Once (YOLO), 521
- yörünge, 691
- yörüngeler, 690

- yukarı örnekleme katmanı, 524
- yumuşak actor-critic algoritması, 706
- yumuşak kümeleme, 250
- yumuşak sınır sınıflandırması, 165
- yüksek boyutlu veri setleri, 224
  
- zaman adımı, 531
- zaman boyunca ağı açma, 531
- zaman boyunca geri yayılım (BPTT), 535
- zaman boyunca kesilmiş geri yayılım, 563
- zaman serisi verisi
  - basit RNN'ler, 538
  - birkaç zaman adımı sonrası tahmin etmek, 542
  - derin RNN'ler, 540
  - genel bakış, 536
  - için diğer modeller, 539
  - için RNN'ler, 530
  - temel ölçütler, 538
- zamansal fark öğrenimi (TD Learning), 668
- zayıf öğrenici, 201
- ZF Net, 497
- zincir kuralı, 300