Derin Sinir Ağları
Hafta 2

Doç. Dr. Kazım HANBAY

Temel Kavramlar

Amaç-Kayıp Fonksiyonları

Sınıflandırma

Softmax Regresyon

Bir Önceki Ders >>>

Yapay sinir ağları

Katman yapıları

İleri-Geri Yayılım Algoritması

Stokastic Gradient Descent Algoritması

Notasyon

Bu kitap boyunca, aşağıdaki gösterim kurallarına bağlı kalacağız. Bu sembollerden bazılarının göstermelik değişken olduğunu, bazılarının ise belirli nesnelere atıfta bulunduğunu unutmayın. Genel bir kural olarak, belirsiz “a” nesnesi, sembolün bir göstermelik değişken olduğunu ve benzer şekilde biçimlendirilmiş sembollerin aynı tipteki diğer nesneleri gösterebileceğini belirtir. Örneğin, “x: bir sayıl”, küçük harflerin genellikle sayıl değerleri temsil ettiği anlamına gelir.

Sayısal Nesneler

- x: skalar (sayıl)
- $\mathbf{x}$: vektör (Yöney)
- $\mathbf{X}$: matris (Dizey)
- X: bir genel tensör (Gerey)
- $\mathbf{I}$: birim dizeyi – köşegen girdileri 1 köşegen-olmayan girdileri 0 olan kare dizey
- x_i, $[\mathbf{x}]_i$: $\mathbf{x}$ dizeyinin i. elemanı
- x_{ij}, $x_{i,j}$, $[\mathbf{X}]_{ij}$, $[\mathbf{X}]_{i,j}$: $\mathbf{X}$ dizeyinin i. satır j. sütundaki elemanı

Küme Kuramı

- $\mathcal{X}$: küme
- $\mathbb{Z}$: tam sayılar kümesi
- $\mathbb{Z}^+$: pozitif tam sayılar kümesi
- $\mathbb{R}$: gerçel sayılar kümesi
- $\mathbb{R}^n$: n boyutlu gerçel sayılı yöneyler kümesi
- $\mathbb{R}^{a\times b}$: a satır ve b sütunlu gerçek sayılı matrisler kümesi
- $|\mathcal{X}|$: $\mathcal{X}$ kümesinin kardinalitesi (eleman sayısı)
- $\mathcal{A} \cup \mathcal{B}$: $\mathcal{A}$ ve $\mathcal{B}$ kümelerinin bileşkesi
- $\mathcal{A} \cap \mathcal{B}$: $\mathcal{A}$ ve $\mathcal{B}$ kümelerinin kesişimi
- $\mathcal{A} \setminus \mathcal{B}$: $\mathcal{B}$ kümesinin $\mathcal{A}$ kümesinden çıkarılması (sadece $\mathcal{A}$ kümesinde olup $\mathcal{B}$ kümesinde olmayan

Fonksiyonlar ve Operatörler

- $f(\cdot)$: işlev (fonksiyon)
- $\log(\cdot)$: doğal logaritma
- $\log_2(\cdot)$: 2lik tabanda logaritma
- $\exp(\cdot)$: üstel fonksiyon
- $\mathbf{1}(\cdot)$: gösterge fonksiyonu, mantık argümanı doğru ise 1 ve değil ise 0
- $\mathbf{1}_{\mathcal{X}}(z)$: küme-üyeliği gösterge işlevi, eğer z elemanı $\mathcal{X}$ kümesine ait ise 1 ve değil ise 0
- $(\cdot)^\top$: bir vektörün veya matrisin devriği
- $\mathbf{X}^{-1}$: $\mathbf{X}$ matrisinin tersi
- $\odot$: Hadamard (eleman-yönlü) çarpımı
- $[\cdot, \cdot]$: bitiştirme
- $\|\cdot\|_p$: L_p büyüklüğü (Norm)
- $\|\cdot\|$: L_2 büyüklüğü (Norm)
- $\langle \mathbf{x}, \mathbf{y} \rangle$: $\mathbf{x}$ ve $\mathbf{y}$ vektörlerinin iç (nokta) çarpımı
- $\sum$: bir elemanlar topluluğu üzerinde toplam
- $\prod$: bir elemanlar topluluğu üzerinde toplam çarpımı

1.2. Temel Bileşenler

- İlk olarak, ne tür bir Makine Öğrenmesi (MÖ) problemi olursa olsun, bizi takip edecek temel bileşenlere daha fazla ışık tutmak istiyoruz:

1. Öğrenebileceğimiz *veri*.
2. Verinin nasıl dönüştürüleceğine dair bir *model*.
3. Modelin ne kadar iyi veya kötü iş çıkardığını ölçümleyen bir *amaç* işlevi (objective function).
4. Kaybı en aza indirmede modelin parametrelerini ayarlamak için bir *algoritma*.

1.2.1. Veri

- Veri bilimini veri olmadan yapamayız.
- Veriyle yararlı bir şekilde çalışmak için, genellikle uygun bir sayısal temsil (gösterim) bulmamız gerekir. Her *örnek* (*veri noktası*, *veri örnekleri* veya *örneklem* olarak da adlandırılır) tipik olarak onlardan modelimizin tahminlemelerini yaptığı *öznitelikler* (veya *ortak değişkenler*) adı verilen sayısal özelliklerden oluşur.

1.2.1. Veri

Eğer görüntü verileriyle çalışıyorsak, bir fotoğraf için, her bir pikselin parlaklığına karşılık gelen sıralı bir sayısal değerler listesi ile temsil edilen bir örnek oluşturabilir. 200×200 bir renkli fotoğraf, her bir uzamsal konum için kırmızı, yeşil ve mavi kanalların parlaklığına karşılık gelen 200×200×3=120000 tane sayısal değerden oluşur. Başka bir geleneksel görevde ise yaş, yaşamsal belirtiler ve teşhisler gibi standart bir dizi özellik göz önüne alındığında, bir hastanın hayatta kalıp kalmayacağını tahmin etmeye çalışabiliriz.

1.2.1. Veri

Genel olarak, ne kadar fazla veriye sahip olursak işimiz o kadar kolay olur. Daha fazla veriye sahip olduğumuzda, daha güçlü modeller eğitebilir ve önceden tasarlanmış varsayımlara daha az bel bağlayabiliriz. (Nispeten) küçük veriden büyük veriye düzen değişikliği, modern derin öğrenmenin başarısına önemli bir katkıda bulunur. İşin özü, derin öğrenmedeki en heyecan verici modellerin çoğu, büyük veri kümeleri olmadan çalışmaz. Bazıları küçük veri rejiminde çalışır, ancak geleneksel yaklaşımlardan daha iyi değildir.

1.2.1. Veri

Son olarak, çok fazla veriye sahip olmak ve onu akıllıca işlemek yeterli değildir. *Doğru* veriye ihtiyacımız vardır. Veri hatalarla doluysa veya seçilen öznitelikler hedefle ilgisizse, öğrenme başarısız olacaktır. Durum şu klişe ile iyi betimlenebilir: *Çöp girerse çöp çıkar*. Ayrıca, kötü tahmin performansı tek olası sonuç değildir. Tahminli polislik, özgeçmiş taraması ve borç verme için kullanılan risk modelleri gibi makine öğrenmesinin hassas uygulamalarında, özellikle çöp verinin olası sonuçlarına karşı dikkatli olmalıyız.

1.2.1. Veri

Yaygın bir hata durumu bazı insan gruplarının eğitim verilerinde temsil edilmediği veri kümelerinde gerçekleşir. Gerçek hayatta, daha önce hiç koyu ten görmemiş bir cilt kanseri tanıma sistemi uyguladığınızı düşünün. Nasıl karar verir??

1.2.2. Modeller

Çoğu makine öğrenmesi, veriyi bir anlamda dönüştürmeyi içerir. Fotoğrafları alarak güleryüzlülük tahmin eden bir sistem kurmak isteyebiliriz.. *Model* ile, bir tipteki veriyi alan ve muhtemel farklı tipteki tahminleri veren hesaplama makinelerini belirtiriz. Özellikle veriden tahmin yapabilecek istatistiksel modellerle ilgileniyoruz. Derin öğrenme, klasik yaklaşımlardan esas olarak odaklandığı güçlü modeller kümesi ile ayrılır. Bu modeller, yukarıdan aşağıya zincirlenmiş verinin art arda dönüşümlerinden oluşur, bu nedenle adları *derin öğrenme*dir.

1.2.3. Amaç Fonksiyonları

- Daha önce, makine öğrenmesini deneyimden öğrenme olarak tanıttık. Burada *öğrenme* ile zamanla bazı görevlerde iyileştirmeyi kastediyoruz. Peki kim neyin bir iyileştirme oluşturduğunu söyleyecek? Modeli güncellemeyi önerdiğiniz zaman önerilen güncellemenin bir iyileştirme mi yoksa bir düşüş mü oluşturacağı konusunda görüş ayrılıkları olabileceğini tahmin edebilirsiniz.
- Biçimsel bir matematiksel öğrenen makineler sistemi geliştirmek için modellerimizin ne kadar iyi (ya da kötü) olduğuna dair kurallı ölçümlere ihtiyacımız var. Makine öğrenmesinde ve daha genel olarak optimizasyonda (eniyilemede), bunları amaç fonsiyonları olarak adlandırıyoruz.

1.2.3. Amaç Fonksiyonları

- Yaygın yaklaşım olarak, genellikle amaç fonksiyonları tanımlarız, böylece daha düşük değer daha iyi anlamına gelir. Bu sadece yaygın bir kanıdır. Daha yüksekken daha iyi olan herhangi bir fonksiyonu alabilir ve onu, işaretini değiştirerek niteliksel olarak özdeş ama daha düşükken daha iyi yeni bir fonksiyona dönüştürebilirsiniz. Düşük daha iyi olduğu için, bu fonksiyona bazen *yitim fonksiyonları (loss function)* denir.

1.2.3. Amaç Fonksiyonları

- Sayısal değerleri tahmin etmeye çalışırken, en yaygın amaç fonksiyonu hata karesidir, yani gerçek referans değeri ile tahmin değeri arasındaki farkın karesi. Sınıflandırma için en yaygın amaç fonksiyonu, hata oranını, yani tahminlerimizin gerçek değere uymadığı örneklerin oranını, en aza indirmektir.
- Tipik olarak, yitim fonksiyonu modelin parametrelerine göre tanımlanır ve veri kümesine bağlıdır. Modelimizin parametrelerinin en iyi değerlerini, eğitim için toplanan örneklerden oluşan bir kümede meydana gelen kaybı en aza indirerek öğreniriz.

1.2.3. Amaç Fonksiyonları

Bununla birlikte, eğitim verisinde iyi performans göstermemiz, görülmemiş veri üzerinde iyi performans göstereceğimizi garanti etmez. Bu nedenle, mevcut veriyi iki parçaya ayırırız: Modelimizin ikisinin de üzerindeki performanslarını raporladığımız eğitim veri kümesi ve test veri kümesi. Eğitim hatasını, bir öğrencinin gerçek bir final sınava hazırlanmak için girdiği uygulama sınavlarındaki puanları gibi düşünebilirsiniz. Sonuçlar cesaret verici olsa bile, bu final sınavında başarıyı garanti etmez. Başka bir deyişle, test performansı eğitim performansından ciddi derecede sapabilir. Bir model eğitim kümesi üzerinde iyi performans gösterdiğinde, ancak bunu görünmeyen veriye genelleştiremediğinde, buna *aşırı öğrenme (overfitting)* diyoruz. Bu durumu uygulama sınavlarında başarılı olunmasına rağmen gerçek sınavda çakmaya benzetebiliriz.

1.2.4. Optimizasyon (Eniyileme) Algoritmaları

Bir kez bir veri kaynağına ve gösterime, bir modele ve iyi tanımlanmış bir amaç fonksiyonuna sahip olduktan sonra, yitim fonksiyonunu en aza indirmek için mümkün olan en iyi parametreleri arayabilme becerisine sahip bir algoritmaya ihtiyacımız var. Derin öğrenme için popüler optimizasyon algoritmaları, gradyan (eğim) inişi (gradient descent) olarak adlandırılan bir yaklaşıma bağlıdır. Kısacası, her adımda, her bir parametre için, bu parametreyi sadece küçük bir miktar bozarsanız eğitim kümesi kaybının nasıl hareket edeceğini (değişeceğini) kontrol ederler. Daha sonra parametreyi kaybı azaltabilecek yönde güncellerler.

1.3. Makine Öğrenmesi Problemleri Çeşitleri

- 1.3.1. Gözetimli Öğrenme

- Gözetimli öğrenme girdi öznitelikleri verildiğinde etiketleri tahmin etme görevini ele alır. Her öznitelik–etiket çiftine örnek denir. Bazen, bağlam açık olduğunda, bir girdi topluluğuna atıfta bulunmak için -karşılık gelen etiketler bilinmese bile- *örnekler* terimini kullanabiliriz. Hedefimiz, herhangi bir girdiyi bir etiket tahminiyle eşleyen bir model üretmektir.
- Bu açıklamayı somut bir örneğe oturtalım; sağlık hizmetlerinde çalışıyorsak, bir hastanın kalp krizi geçirip geçirmeyeceğini tahmin etmek isteyebiliriz. “Kalp krizi var” veya “kalp krizi yok” gözlemi, bizim etiketimiz olacaktır. Girdi öznitelikleri, kalp atış hızı, diyastolik ve sistolik kan basıncı gibi hayati belirtiler olabilir.

1.3. Makine Öğrenmesi Problemleri Çeşitleri

- 1.3.1. Gözetimli Öğrenme

- Gözetimli öğrenme girdi öznitelikleri verildiğinde etiketleri tahmin etme görevini ele alır. Her öznitelik–etiket çiftine örnek denir. Bazen, bağlam açık olduğunda, bir girdi topluluğuna atıfta bulunmak için -karşılık gelen etiketler bilinmese bile- *örnekler* terimini kullanabiliriz. Hedefimiz, herhangi bir girdiyi bir etiket tahminiyle eşleyen bir model üretmektir.
- Bu açıklamayı somut bir örneğe oturtalım; sağlık hizmetlerinde çalışıyorsak, bir hastanın kalp krizi geçirip geçirmeyeceğini tahmin etmek isteyebiliriz. “Kalp krizi var” veya “kalp krizi yok” gözlemi, bizim etiketimiz olacaktır. Girdi öznitelikleri, kalp atış hızı, diyastolik ve sistolik kan basıncı gibi hayati belirtiler olabilir.

1.3.1. Gözetimli Öğrenme

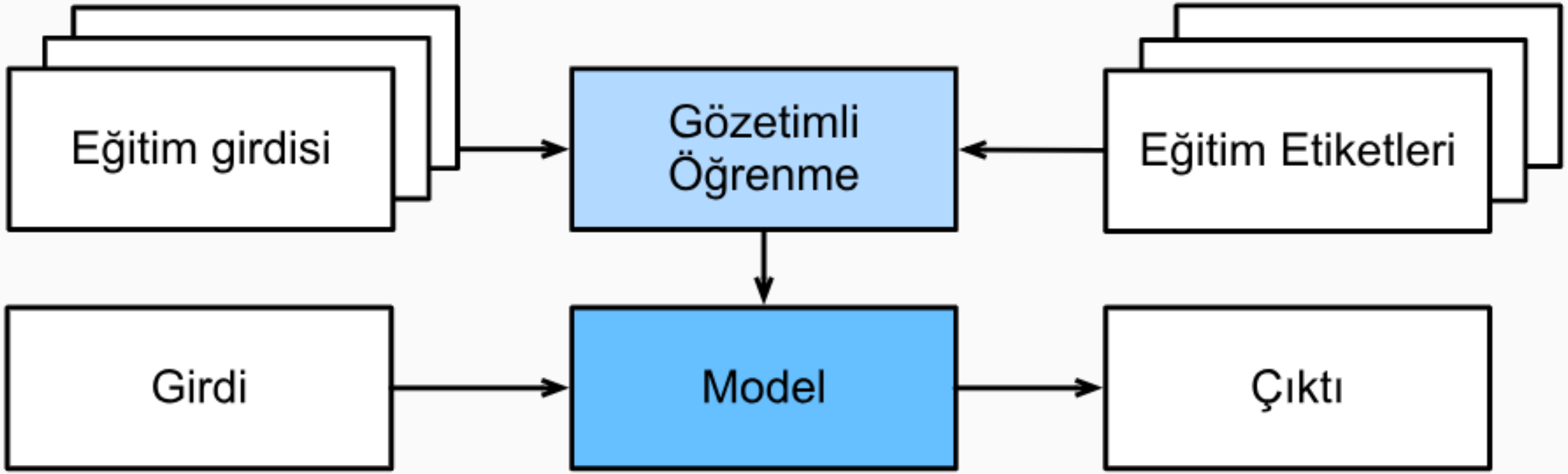


Fig. 1.3.1 Gözetimli öğrenme.

1.3.1.2. Sınıflandırma

*Sınıflandırma*da, modelimizin özniteliklere, mesela bir görüntüdeki piksel değerlerine, bakmasını ve ardından örneğimizin bazı ayrık seçenekler kümesi arasından hangi *kategori*ye (aslen *sınıf* olarak adlandırılırlar) ait olduğunu tahmin etmesini istiyoruz. Elle yazılmış rakamlar için, 0 ile 9 arasındaki rakamlara karşılık gelen on sınıfımız olabilir. Sınıflandırmanın en basit şekli, sadece iki sınıf olduğunda, *ikili sınıflandırma* dediğimiz bir problemdir. Örneğin, veri kümemiz hayvanların görüntülerinden oluşabilir ve *etiketlerimiz* ö{kedi,köpek} sınıfları olabilir. Bağlanımdayken, sayısal bir değer çıkarmak için bir *bağlanımcı* aradık, sınıflandırmada çıktısı tahminlenen sınıf ataması olan bir *sınıflandırıcı* ararız.

1.3.1.2. Sınıflandırma

Kitap daha teknik hale geldikçe gireceğimiz nedenlerden ötürü, yalnızca kategorik bir atama yapabilen, örneğin “kedi” veya “köpek” çıktısı, bir modeli optimize etmek zor olabilir. Bu tür durumlarda, modelimizi olasılıklar dilinde ifade etmek genellikle daha kolaydır. Bir örneğin öznitelikleri verildiğinde, modelimiz her olası sınıfa bir olasılık atar. Hayvan sınıflandırma örneğimize dönersek, ki burada sınıflar {kedi,köpek}’tir, bir sınıflandırıcı bir görüntü görebilir ve görüntünün bir kedi olma olasılığını 0.9 çıkarabilir. Bu sayıyı, sınıflandırıcının görüntünün bir kediyi gösterdiğinden %90 emin olduğunu söyleyerek yorumlayabiliriz. Öngörülen sınıf için olasılığın büyüklüğü bir çeşit belirsizlik taşır.

1.3.1.2. Sınıflandırma

İkiden fazla olası sınıfımız olduğunda, soruna *çok sınıflı sınıflandırma* diyoruz. Yaygın örnekler arasında elle yazılmış karakteri, {0,1,2,...9,a,b,c,...}, tanıma yer alır. Bağlanım problemleriyle uğraşırken kare hata kayıp fonksiyonunu en aza indirmeye çalışırız; sınıflandırma problemleri için ortak kayıp fonksiyonu, sonraki bölümlerdeki bilgi teorisine giriş ile adını açıklığa kavuşturacağımız *çapraz düzensizlik* (entropi) diye adlandırılır.

1.3.1.2. Sınıflandırma

Arka bahçenizde aşağıda gösterildiği gibi güzel bir mantar bulduğunuzu varsayın.

Fig. 1.3.2 Ölüm tehlikesi – yemeyin!

1.3.1.2. Sınıflandırma

Şimdi, bir sınıflandırıcı oluşturduğunuzu ve bir mantarın bir fotoğrafına göre zehirli olup olmadığını tahmin etmek için eğittiğinizi varsayın. Zehir tespit sınıflandırıcısının Fig. 1.3.2'nin zehirli olma olasılığında 0.2 sonucunu verdiğini varsayalım. Başka bir deyişle, sınıflandırıcı, mantarımızın zehirli *olmadığından* %80 emindir. Yine de, yemek akıllıca değil!! Çünkü lezzetli bir akşam yemeğinin belirli bir yararı, ondan ölme riski olan %20 değerine değmez. Başka bir deyişle, belirsiz riskin etkisi faydadan çok daha fazladır. Bu nedenle, zarar fonksiyonu olarak maruz kaldığımız beklenen riski hesaplamamız gerekir, yani sonucun olasılığını onunla ilişkili fayda (veya zarar) ile çarpmamız gerekir.

1.3.1.2. Sınıflandırma

Temel olarak, maruz kaldığımız beklenen riski kayıp fonksiyonu olarak hesaplamamız gerekir, yani sonucun olasılığını, bununla ilişkili fayda (veya zarar) ile çarpmamız gerekir. Bu durumda, mantarı yemekten kaynaklanan kayıp $0.2 \times \infty + 0.8 \times 0 = \infty$ olurken, onu çöpe atmanın kaybı $0.2 \times 0 + 0.8 \times 1 = 0.8$ olacaktır. Dikkatimiz haklıydı: Herhangi bir mantar bilimcinin bize söyleyeceği gibi, Fig. 1.3.2'deki mantar aslında zehirlidir.

1.5. Derin Öğrenme

- Basit bir şekilde açıklarsak derin öğrenme, çok katmanlı sinir ağlarına verilen addır.
- Fotoğraflar veya ses gibi gözlemsel verileri anlamlandırmak için sinir ağları, verileri birbirine bağlı düğüm katmanlarından geçirir. Bilgi bir katmandan geçtiğinde, o katmandaki her bir düğüm, veriler üzerinde basit işlemler gerçekleştirir ve sonuçları seçerek diğer düğümlere iletir. Sonraki her katman, ağ çıktıyı oluşturana kadar bir öncekinden daha yüksek düzeyde bir özelliğe odaklanır.
- Girdi katmanı ile çıktı katmanı arasındaki katman gizli katmanlardır. Sinir ağları ile derin öğrenme arasındaki fark burada ortaya çıkar: Temel bir sinir ağı bir veya iki gizli katmana sahip olabilirken, bir derin öğrenme ağı düzinelerce hatta yüzlerce katmana sahip olabilir. Farklı katman ve düğümlerin sayısını artırmak ağın doğruluğunu artırabilir. Ancak, daha fazla katman ayrıca bir modelin daha fazla parametre ve hesaplama kaynağı gerektireceği anlamına da gelebilir.

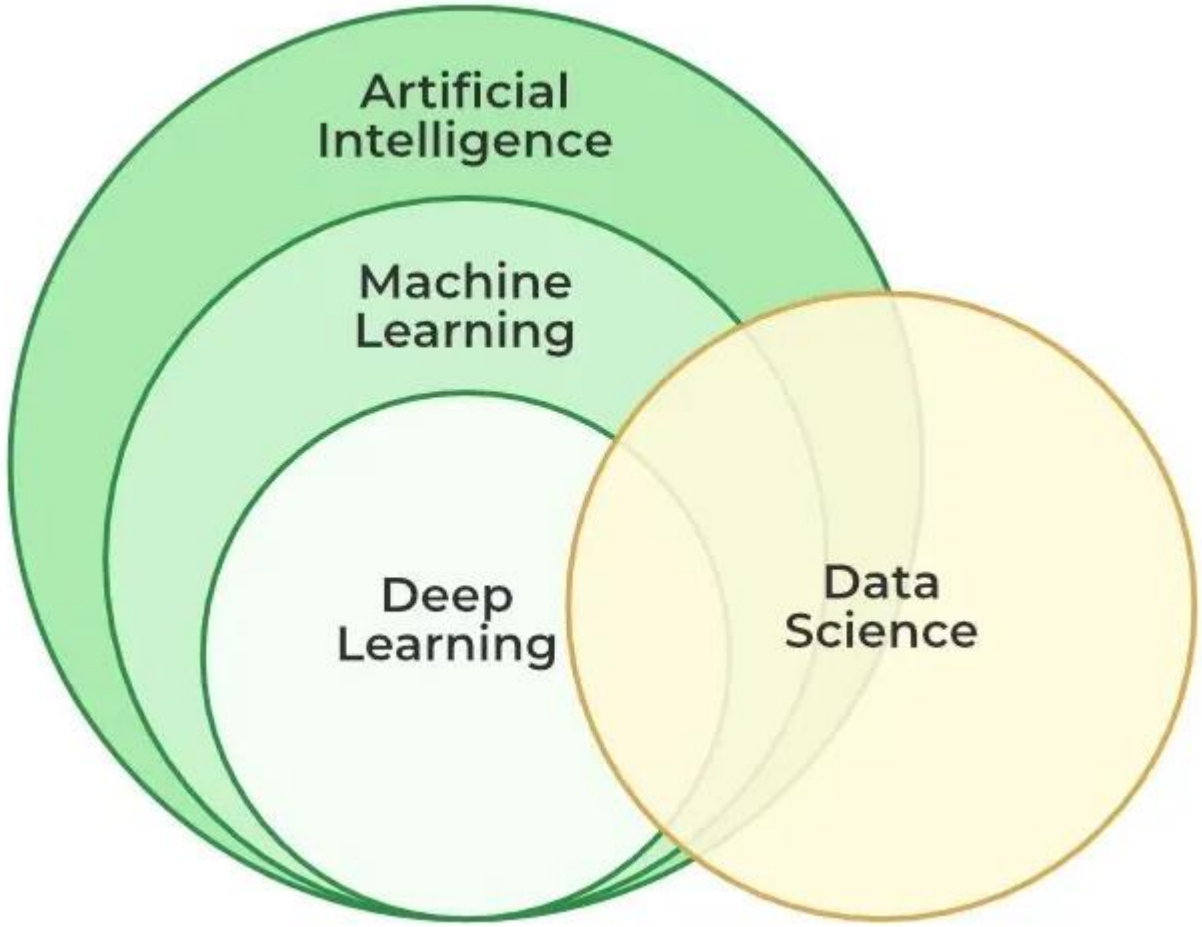
1.5. Derin Öğrenme

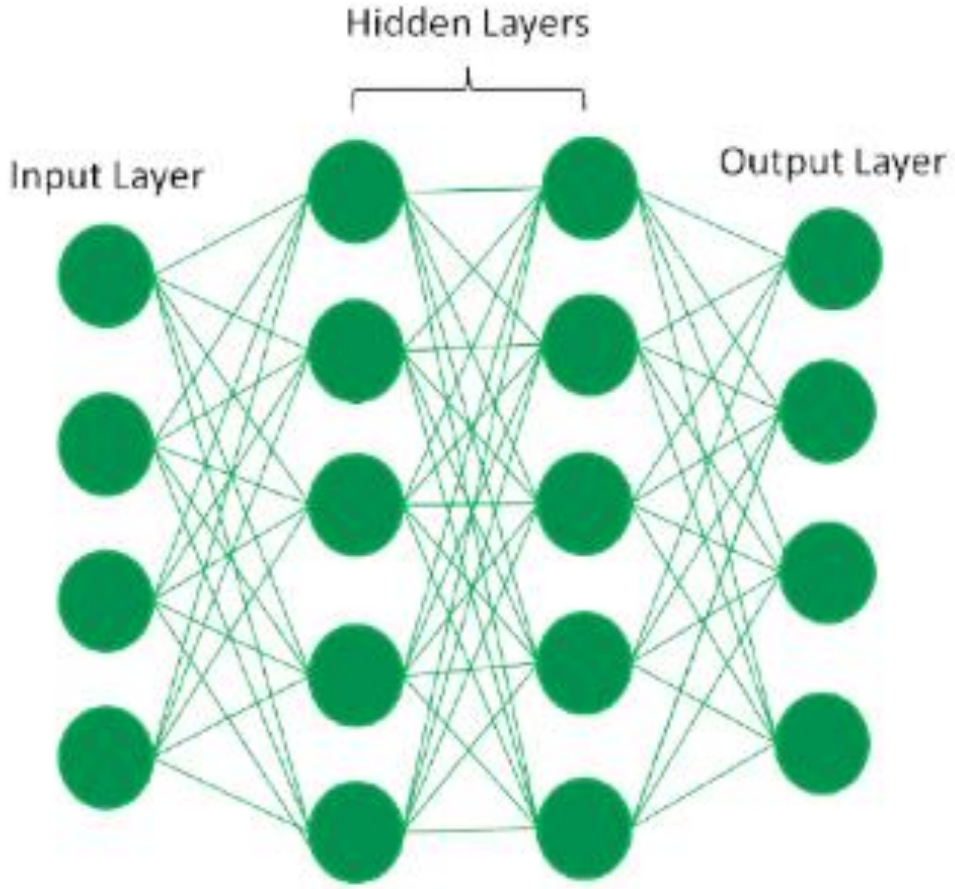
- Basit bir şekilde açıklarsak derin öğrenme, çok katmanlı sinir ağlarına verilen addır.
- Fotoğraflar veya ses gibi gözlemsel verileri anlamlandırmak için sinir ağları, verileri birbirine bağlı düğüm katmanlarından geçirir. Bilgi bir katmandan geçtiğinde, o katmandaki her bir düğüm, veriler üzerinde basit işlemler gerçekleştirir ve sonuçları seçerek diğer düğümlere iletir. Sonraki her katman, ağ çıktıyı oluşturana kadar bir öncekinden daha yüksek düzeyde bir özelliğe odaklanır.
- Girdi katmanı ile çıktı katmanı arasındaki katman gizli katmanlardır. Sinir ağları ile derin öğrenme arasındaki fark burada ortaya çıkar: Temel bir sinir ağı bir veya iki gizli katmana sahip olabilirken, bir derin öğrenme ağı düzinelerce hatta yüzlerce katmana sahip olabilir. Farklı katman ve düğümlerin sayısını artırmak ağın doğruluğunu artırabilir. Ancak, daha fazla katman ayrıca bir modelin daha fazla parametre ve hesaplama kaynağı gerektireceği anlamına da gelebilir.

1.5. Derin Öğrenme- Derin Öğrenme Nasıl Çalışır?

- **Sinir ağı,** giriş verilerini işlemek için iş birliği yapan birbirine bağlı düğüm veya nöron katmanlarından oluşur. **Tamamen bağlı derin bir sinir ağında** , veriler birden fazla katmandan geçer ve burada her nöron do**gizli katmanlardan** geçer . Son **çıktı katmanı,** modelin tahminini üretirğrusal olmayan dönüşümler gerçekleştirir ve bu da modelin verilerin karmaşık gösterimlerini öğrenmesine olanak tanır.
- Derin bir sinir ağında, **giriş katmanı** verileri alır ve bu veriler, verileri doğrusal olmayan işlevler kullanarak dönüştüren **gizli katmanlardan** geçer. Son **çıktı katmanı,** modelin tahminini üretir.

1.5. Derin Öğrenme- Derin Öğrenme Nasıl Çalışır?





Tam Bağlantılı Derin Sinir Ağı

• 1.5. Derin Öğrenme- Sinir ağlarının türleri

1. **İleri beslemeli sinir ağları (FNN'ler)** , verilerin girdiden çıktıya doğru tek yönde aktığı en basit ANN türüdür. Sınıflandırma gibi temel görevler için kullanılır.

2. **Evrişimsel Sinir Ağları (CNN'ler),** görüntüler gibi ızgara benzeri verileri işlemek için uzmanlaşmıştır. CNN'ler, mekansal hiyerarşileri tespit etmek için evrişimsel katmanlar kullanır ve bu da onları bilgisayarlı görüş görevleri için ideal hale getirir.

3. **Tekrarlayan Sinir Ağları (RNN'ler),** zaman serileri ve doğal dil gibi ardışık verileri işleyebilir. RNN'ler, dil modelleme ve konuşma tanıma gibi uygulamaları etkinleştirerek zaman içinde bilgileri tutmak için döngülere sahiptir. LSTM'ler ve GRU'lar gibi varyantlar, kaybolan eğim sorunlarını ele alır.

4. **Üretken Çelişkili Ağlar (GAN'lar),** gerçekçi veriler oluşturmak için yarışan bir üreteç ve bir ayırıcı olmak üzere iki ağdan oluşur. GAN'lar, görüntü oluşturma, stil aktarımı ve veri artırma için yaygın olarak kullanılır.

5. **Otokoderler** , verimli veri kodlamalarını öğrenen gözetimsiz ağlardır. Giriş verilerini gizli bir gösterime sıkıştırır ve yeniden yapılandırırlar, boyut azaltma ve anormallik tespiti için faydalıdır.

6. **Transformer Networks,** öz dikkat mekanizmalarıyla NLP'yi devrim niteliğinde değiştirmiştir. Transformatörler, GPT ve BERT gibi modelleri güçlendirerek çeviri, metin oluşturma ve duygu analizi gibi görevlerde mükemmeldir.

• 1.5. Derin Öğrenme- Sinir ağlarının türleri

1. **İleri beslemeli sinir ağları (FNN'ler)** , verilerin girdiden çıktıya doğru tek yönde aktığı en basit ANN türüdür. Sınıflandırma gibi temel görevler için kullanılır.

2. **Evrişimsel Sinir Ağları (CNN'ler),** görüntüler gibi ızgara benzeri verileri işlemek için uzmanlaşmıştır. CNN'ler, mekansal hiyerarşileri tespit etmek için evrişimsel katmanlar kullanır ve bu da onları bilgisayarlı görüş görevleri için ideal hale getirir.

3. **Tekrarlayan Sinir Ağları (RNN'ler),** zaman serileri ve doğal dil gibi ardışık verileri işleyebilir. RNN'ler, dil modelleme ve konuşma tanıma gibi uygulamaları etkinleştirerek zaman içinde bilgileri tutmak için döngülere sahiptir. LSTM'ler ve GRU'lar gibi varyantlar, kaybolan eğim sorunlarını ele alır.

4. **Üretken Çelişkili Ağlar (GAN'lar),** gerçekçi veriler oluşturmak için yarışan bir üreteç ve bir ayırıcı olmak üzere iki ağdan oluşur. GAN'lar, görüntü oluşturma, stil aktarımı ve veri artırma için yaygın olarak kullanılır.

5. **Otokoderler** , verimli veri kodlamalarını öğrenen gözetimsiz ağlardır. Giriş verilerini gizli bir gösterime sıkıştırır ve yeniden yapılandırırlar, boyut azaltma ve anormallik tespiti için faydalıdır.

6. **Transformer Networks,** öz dikkat mekanizmalarıyla NLP'yi devrim niteliğinde değiştirmiştir. Transformatörler, GPT ve BERT gibi modelleri güçlendirerek çeviri, metin oluşturma ve duygu analizi gibi görevlerde mükemmeldir.

3.6. Softmax Regresyon

- Softmax regresyonu, çok sınıflı sınıflandırma problemlerinde kullanılan bir lineer modeldir. Bir veri kümesine ait örnekleri belirli sınıflara ayırmak için kullanılır. Lojistik regresyonun genişletilmiş versiyonudur ve birden fazla sınıfa tahmin yapmayı sağlar. Bir örnek **X** verildiğinde, model **W ağırlıkları** ve **b bias** terimi kullanarak aşağıdaki gibi bir doğrusal ilişki kurar:
- t_i = XW + b
- Bu işlem, sınıflara ait ham skorları üretir. Ancak, bu skorları (t_i) **olasılıklara** çevirmek için **Softmax fonksiyonu** kullanılır:

$$\hat{y}_i = \frac{e^{t_i}}{\sum_{j=1}^{C} e^{t_j}}$$

Burada:

- t_i = i. sınıfın ham skoru
- $\hat{y}_i$ = i. sınıfın tahmini olasılığı
- C = sınıf sayısı
- e^{t_i} = e tabanında üstel değer
- **Toplamı 1 yapan bir olasılık dağılımı oluşturur.**

3.6. Softmax Regresyon

- Softmax regresyonu **basit bir modeldir**, ancak aşağıdaki sınırlamalara sahiptir:
- ✅ **Avantajları**:
- Kolay uygulanabilir ve hızlıdır.
- Küçük ve orta büyüklükteki veri setlerinde iyi çalışır.
- ❌ **Dezavantajları**:
- Daha karmaşık modeller (CNN gibi) karşısında düşük doğruluk gösterebilir.
- Görsellerdeki karmaşık özellikleri öğrenemez.
- Fashion MNIST veri kümesi için Softmax Regresyonu genellikle **%80-85 doğruluk oranına ulaşabilir**, ancak **CNN gibi derin öğrenme modelleri ile %95+ doğruluk elde edilebilir.**

3.6. Softmax Regresyon

Şimdi, Softmax regresyonunun mantıksal VE (AND) kapısı üzerinde nasıl çalıştığını adım adım teorik ve matematiksel olarak inceleyeceğiz.

2. Mantıksal VE (AND) Kapısı

Mantıksal VE kapısı, giriş değerleri yalnızca **her iki giriş de 1 olduğunda 1 çıktısı üretir**.

Giriş A	Giriş B	Çıktı Y
0	0	0
0	1	0
1	0	0
1	1	1

Bu tablo, **Softmax regresyonu ile bir sınıflandırma problemi** olarak ele alınabilir.

3.6. Softmax Regresyon

- Softmax regresyonu, bir giriş vektörüne karşılık birden fazla olasılık üretir ve en yüksek olasılığa sahip olanı tahmin olarak belirler. Model aşağıdaki matematiksel formüle dayanır:

$$t_i = WX + b$$

Burada:

- X = Giriş vektörü (A ve B değerleri)
- W = Ağırlık matrisi
- b = Bias terimi
- t_i = Sınıf skorları

3.6. Softmax Regresyon

- Daha önce bahsedildiği gibi, Softmax fonksiyonu ise bu skorları olasılığa dönüştürür:

$$t_i = WX + b$$

$$\hat{y}_i = \frac{e^{t_i}}{\sum_{j=1}^{C} e^{t_j}}$$

Burada:

- t_i = i. sınıfın ham skoru
- $\hat{y}_i$ = i. sınıfın tahmini olasılığı
- C = sınıf sayısı
- e^{t_i} = e tabanında üstel değer
- **Toplamı 1 yapan bir olasılık dağılımı oluşturur.**

3.6. Softmax Regresyon

- Daha önce bahsedildiği gibi, Softmax fonksiyonu ise bu skorları olasılığa dönüştürür:

$$t_i = WX + b$$

$$\hat{y}_i = \frac{e^{t_i}}{\sum_{j=1}^{C} e^{t_j}}$$

Burada:

- t_i = i. sınıfın ham skoru
- $\hat{y}_i$ = i. sınıfın tahmini olasılığı
- C = sınıf sayısı
- e^{t_i} = e tabanında üstel değer
- **Toplamı 1 yapan bir olasılık dağılımı oluşturur.**

Burada , ***C*** sınıf sayısını gösterir. AND kapısı için **iki sınıf** vardır: 0 ve 1.

3.6. Softmax Regresyon

4. Mantıksal VE Kapısı için Model Eğitimi

Adım 1: Veri Seti Oluşturma

Mantıksal VE kapısı için **giriş ve çıkış verileri**:

$$X = \begin{bmatrix} (0,0) \\ (0,1) \\ (1,0) \\ (1,1) \end{bmatrix}, Y = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

Adım 2: Ağırlık ve Bias Değerlerinin Başlatılması

Modelin rastgele başlatılan ağırlık ve bias değerleri ile eğitime başlaması gerekir.

$$W = \begin{bmatrix} w_1 & w_2 \end{bmatrix}, b = 0$$

Adım 3: Lineer Kombinasyon Hesaplama

Örnek giriş için:

$$t = w_1 A + w_2 B + b$$

Eğer $w_1 = w_2 = 1$ ve $b = -1.5$ olarak seçilirse:

A	B	t = w_1 A + w_2 B + b	e^t	Softmax(0)	Softmax(1)
0	0	-1.5	0.22	0.87	0.13
0	1	-0.5	0.61	0.74	0.26
1	0	-0.5	0.61	0.74	0.26
1	1	0.5	1.65	0.39	0.61

Burada **çıkış 1 için softmax olasılığı, girişler 1 olduğunda daha yüksektir.**

Adım 4: Kayıp Fonksiyonu (Cross-Entropy)

Cross-entropy kaybı aşağıdaki gibi hesaplanır:

$$\mathcal{L} = -\sum_{i} y_i \log(\hat{y}_i)$$

Model, kaybı minimize edecek şekilde Stokastik Gradient Descent (SGD) veya **Adam** optimizasyon yöntemleriyle güncellenir.

Örnek Hesaplama (Mantıksal VE Kapısı için)

Tablomuzda **(0,0) girişine karşılık hesaplanan ham skor**:

$$t_0 = -1.5$$

Bu değer, softmax fonksiyonuna girdi olarak verilir.

$$e^{t_0} = e^{-1.5} \approx 0.2231$$

Eğer **t = -1.5 ve diğer bir sınıf için t = 0** olarak belirlenirse:

Sınıf	Ham Skor t	Üstel Değer e^t
0 (Yanlış)	-1.5	$e^{-1.5} \approx 0.2231$
1 (Doğru)	0	$e^0 = 1$

Toplam üstel değerler:

$$\sum e^{t_j} = e^{-1.5} + e^{0} = 0.2231 + 1 = 1.2231$$

Softmax(0) Olasılığı:

$$\hat{y}_0 = \frac{e^{-1.5}}{e^{-1.5} + e^{0}} = \frac{0.2231}{1.2231} \approx 0.182$$

Softmax(1) Olasılığı:

$$\hat{y}_1 = \frac{e^{0}}{e^{-1.5} + e^{0}} = \frac{1}{1.2231} \approx 0.818$$

Yani:

- **Softmax(0) ≈ 0.182**
- **Softmax(1) ≈ 0.818**

2. Neden $t = -1.5$ ve $t = 0$?

Softmax fonksiyonunda **her sınıf için bir skor (t) belirlenmelidir.**

Mantıksal VE (AND) kapısı için **giriş (0,0) olduğunda çıktı 0 olmalıdır.**

Ancak model **bu sonucu üretmek için skorlara bağlı çalışır.**

Örneğin:

- Eğer giriş (0,0) ise ve çıktının 0 olması bekleniyorsa, **bu sınıfa ait skor diğerine göre düşük olmalıdır.**
- Bu yüzden **sınıf 0 için** $t = 0$ ve **sınıf 1 için** $t = -1.5$ gibi bir seçim yapabiliriz.
- Bu durumda softmax fonksiyonunun hesaplayacağı **olasılıklar**, sınıf 0'ın (çıkış 0 olan durum) daha yüksek ihtimale sahip olmasını sağlar.

Eğer **skorlar arasında bir fark yaratmazsak**, softmax fonksiyonu **her iki sınıfı eşit olasılıkla tahmin eder** ve öğrenme gerçekleşmez.

3. Alternatif Skor Seçimleri Mümkün mü?

Evet, $t = -1.5$ **ve** $t = 0$ **rastgele bir seçim değildir**, ancak başka değerler de seçilebilir.

Önemli olan **sınıf 0'a (yanlış tahmin) düşük bir olasılık, sınıf 1'e (doğru tahmin) yüksek bir olasılık verilmesini sağlamaktır.**

Alternatif olarak şu gibi seçimler de yapılabilir:

- $t = -2, t = 0$
- $t = -3, t = 1$
- $t = -1, t = 0.5$

Bu seçimler **modelin nasıl optimize edileceğine ve veri dağılımına bağlıdır.**

Genellikle **bu değerler eğitim sırasında optimize edilerek güncellenir** ve en iyi ağırlıklar bulunur.

4. Sonuç

Softmax fonksiyonunun **doğru çalışması için skorlarda bir ayrım olmalıdır.**

Bu yüzden:

- **Yanlış sınıfın skoru düşük tutulur** (örneğin -1.5)
- **Doğru sınıfın skoru daha yüksek olur** (örneğin 0)

Bu seçim **tamamen sabit değildir** ve **model eğitimi sırasında güncellenir.**

Ancak, **temel mantık şudur: Daha yüksek skor → Daha yüksek olasılık.**

Bu yüzden $t = -1.5$ ve $t = 0$ seçimi, softmax'ın doğru çalışmasını sağlamak için kullanılan bir örnektir!

Keras kullanılarak oluşturulmuş bir Conv2D (2D Evrişimli Sinir Ağı) modelini tanımlama

```
from keras import layers
from keras import ops

model = keras.Sequential()
model.add(keras.Input(shape=(100, 100, 1)))  # 100x100x1 RGB görüntü
model.add(layers.Conv2D(2, 2, strides=1, use_bias=False, activation="relu"))
model.add(layers.Conv2D(2, 2, strides=2, use_bias=False, activation="relu"))
model.add(layers.MaxPooling2D(3))


model.summary()
```

Kod Açıklama

`Sequential()`, katmanları sırasıyla ekleyerek **basit, lineer bir model** oluşturur.

```
model.add(keras.Input(shape=(100, 100, 1)))  # 100x100x1 gri görüntü
```

Model **100x100 boyutunda, tek kanallı (1 kanal = gri tonlamalı) görüntüleri** kabul eder.

`(100, 100, 1)` : **100 yükseklik × 100 genişlik × 1 kanal**

RGB görüntü olsaydı, `shape=(100, 100, 3)` olurdu.

```
model.add(layers.Conv2D(2, 2, strides=1, use_bias=False, activation="relu"))
```

2 filtreli (`filters=2`) bir evrişim katmanı ekler.

Çekirdek boyutu (`kernel_size=2`): `(2,2)` boyutunda küçük evrişim çekirdekleri kullanır.

Adım sayısı (`strides=1`): Filtre, görüntü üzerinde **1 piksel kayarak** ilerler.

Önyargı terimi yok (`use_bias=False`): Evrişim katmanı sadece çekirdek ağırlıklarını öğrenir.

Aktivasyon fonksiyonu (`activation="relu"`): ReLU fonksiyonu kullanılarak negatif değerler sıfırlanır.

Kod Açıklama

- Çıktı boyutu nasıl hesaplanır?

Çekirdek (**2x2**) ve `strides=1` kullanıldığı için:

$$\text{Çıktı yüksekliği} = (100 - 2)/1 + 1 = 99$$

$$\text{Çıktı genişliği} = (100 - 2)/1 + 1 = 99$$

Bu yüzden, **çıktı boyutu** `(99, 99, 2)` **olur.**

Conv2D için:

$$\frac{(\text{Giriş Boyutu} - \text{Kernel Boyutu} + 2 \times \text{Padding})}{\text{Strides}} + 1$$

MaxPooling için:

$$\frac{(\text{Giriş Boyutu} - \text{Pool Size})}{\text{Strides}} + 1$$

CNN katman çıktılarının boyutunu hesaplama formülleri

Kod Açıklama

- İkinci Evrişim Katmanı

```
model.add(layers.Conv2D(2, 2, strides=2, use_bias=False, activation="relu"))
```


```
model.add(layers.Conv2D(2, 2, strides=2, use_bias=False, activation="relu"))
```


2 filtreli bir başka evrişim katmanı.

Çekirdek boyutu yine `(2,2)`.

Adım sayısı (`strides=2`): **Her adıma 2 piksel atlatarak ilerler** (bu, boyutu küçültür).

- Çıktı boyutu nasıl hesaplanır?

Bu katmana gelen giriş **(99,99,2)** idi.

Çekirdek `(2,2)` ve `strides=2` olduğu için:

$$\text{Çıktı yüksekliği} = (99 - 2)/2 + 1 = 49$$

$$\text{Çıktı genişliği} = (99 - 2)/2 + 1 = 49$$

Bu yüzden, **çıktı boyutu** `(49, 49, 2)` **olur.**

Kod Açıklama

- MaxPooling Katmanı → `model.add(layers.MaxPooling2D(3))`

MaxPooling (3x3 havuzlama katmanı) eklenmiştir.

Her 3x3 bölgedeki en büyük değeri alarak görüntü boyutunu küçültür.

Varsayılan olarak `strides=3` **kabul edilir.**

Yani, **3'er 3'er atlatarak** ilerler.

- Çıktı boyutu nasıl hesaplanır?

Bu katmana gelen giriş **(49,49,2)** idi.

Havuzlama `(3,3)` ve `strides=3` olduğu için:

$$\text{Çıktı yüksekliği} = (49 - 3)/3 + 1 = 16$$

$$\text{Çıktı genişliği} = (49 - 3)/3 + 1 = 16$$

Bu yüzden, **çıktı boyutu** `(16, 16, 2)` **olur.**