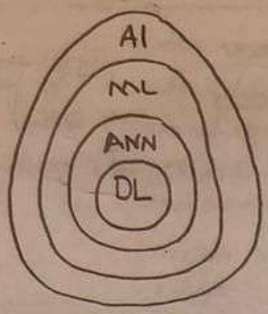
1) Derin Öğrenmeye Giriş



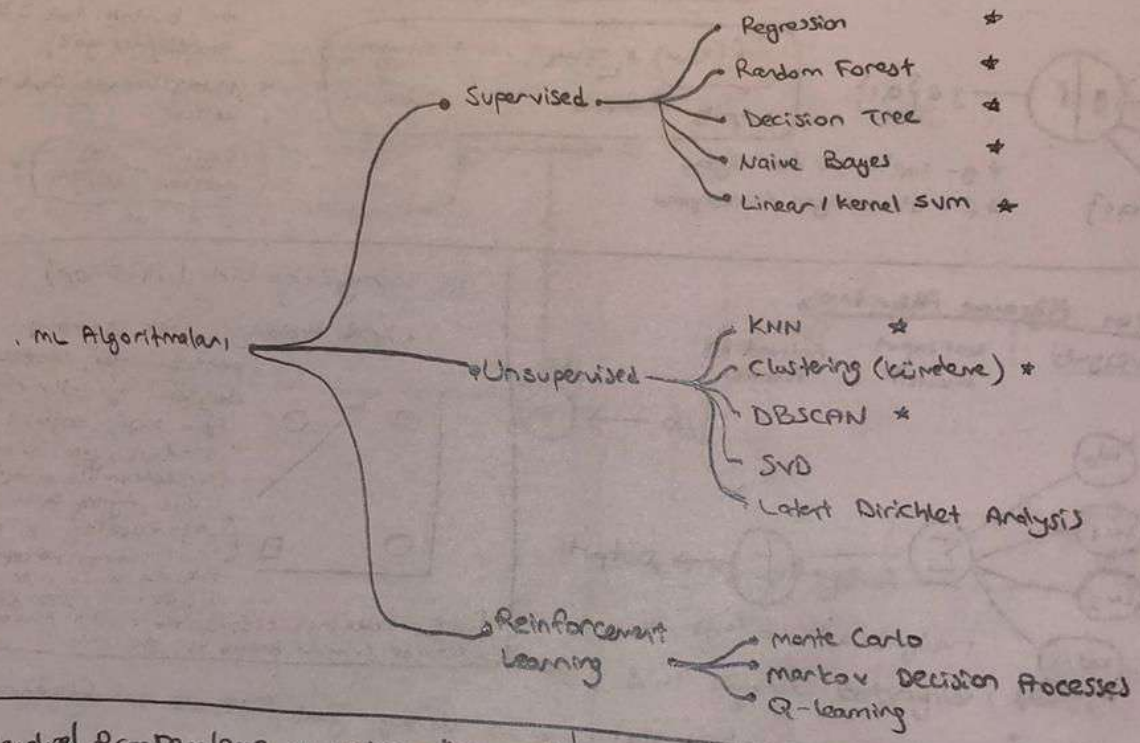
AI: İnsanlar gibi bir görevi gözlemek için makinelerin yeteneklerini sağlayan herhangi bir teknik.

ML: Bilgisayarların açıkça programlanmadan örneklerden öğrenmelerine olanak tanıyan algoritmalar

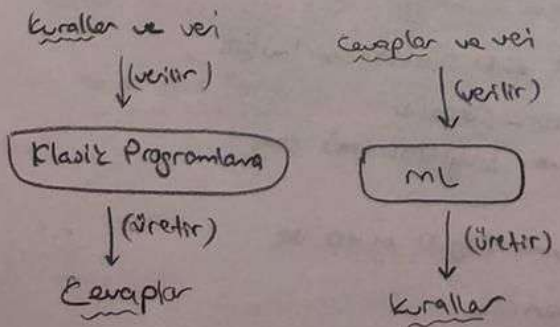
ANN: Beyin esinli makine öğrenimi modelleri.

DL: Derin yapay sinir ağlarını model olarak kullanan ve veri temsillerinin bir hiyerarşisini otomatik olarak oluşturan bir ML alt kümesi.
↳ * Temelde çok katmanlı sinir ağı

ML Algoritma Tipleri



Geleneksel Programlama ve Makine Öğrenmesi Farkı



Öğrenme Türleri

Supervised (Denetimli)

- etiketli veri
- doğrudan geri bildirim
- sonucu / geleceği tahmin eder
- ★ ağırlıkları güncelleyerek çalışır

(örn: email spam sistemi, resme sınıflandırma)

Unsupervised (Denetimsiz)

- etiketler / hedefler yok
- geribildirim yok
- veride gizli yapıları bulur.
- ★ benzer olanları gruplayarak çalışır. Grup sayısı belirsiz

(örn: türdeşme, boyut azaltma, anomali tespiti)

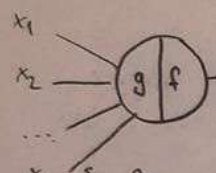
Reinforcement (Pekiştirmeli)

- karar süreci
- ödül sistemi (ödül - ceza) max min
- eylemler dizisini öğrenir
- ★ metasezgisel Algoritmalarda var.

(örn: robotik, finans, tıbbi tanı yetedavi önerme)

McCulloch ve Pitts Nöron modeli

- problem ağırlıkları bulmak



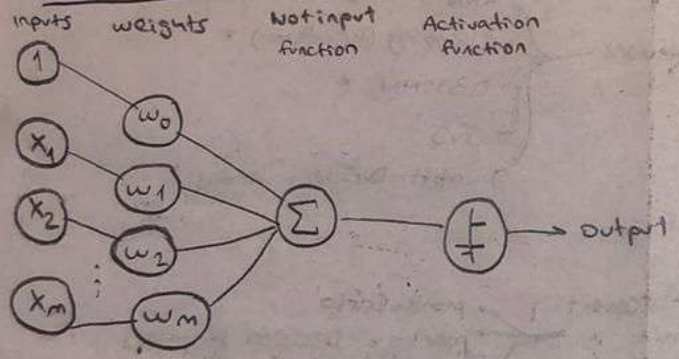
$$\underbrace{f(x,w)}_{\text{outputs}} = \underbrace{x_1 w_1 + \dots + x_n w_n}_{\text{inputs}} = \underbrace{\quad}_{\text{weights}}$$

- ★ g - toplamlar fonksiyonu
- ★ f - aktivasyon fonksiyonu

- ★ 1, yanlılık değeri
- ★ perceptron alg. da 2 çıktış var. Bunlar 1 ve 0 değerleri. Ara değerler yok!
- ★ Çıktı istenenden farklı gelirse:

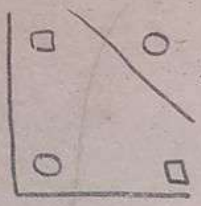
$$\underbrace{(\text{olması gereken} - \text{elde ettiğim})}_{w} \times X$$

Perceptron Öğrenme Algoritması



İlk Yapay Zeka kışı (1970'ler)

- XOR Problemi: Tek katmanlı perceptronlar, sadece doğrusal ayırıcılar öğrenebilir. Ama XOR doğrusal olmayan bir fonksiyon. Bu nedenle perceptron alg. modifiye edildi. Ayrıca farklı sinir ağı modelleri geliştirildi:
  1. Çok katmanlı perceptronlar
  2. Aktivasyon fonk.ları (Sigmoid,...)
  3. Diğer sinir ağları (CNN, RNN)



* XOR problemının aşılması, YZ kışının sona ermesine katkıda bulunan önemli bir gelişme.

Initialize $\vec{w} = \vec{0}$ ..... w'yi başlat. w=0 vektörü yanlış olarak sınıflandırır

```
while TRUE do
  m = 0                                  ... Yanlış sınıflandırma sayısını say.
  for (xi, yi) ∈ D do
    if yi (w⃗^T . x⃗i) ≤ 0 then           ... Eğer (xi, yi) çifti yanlış sınıflandırılmışsa
      w⃗ ← w⃗ + y x⃗                       ... w ağırlık vektörünü güncelle
      m ← m + 1                          ... yanlış sınıflandırma sayısını (m) say
    end if
  end for
  if m = 0 then                          ... yanlış sınıflandırma sayısı m = 0 ise
    break                                ... while döngüsünden çık
  end if
end while
```

★ ödev : 'bias' ?

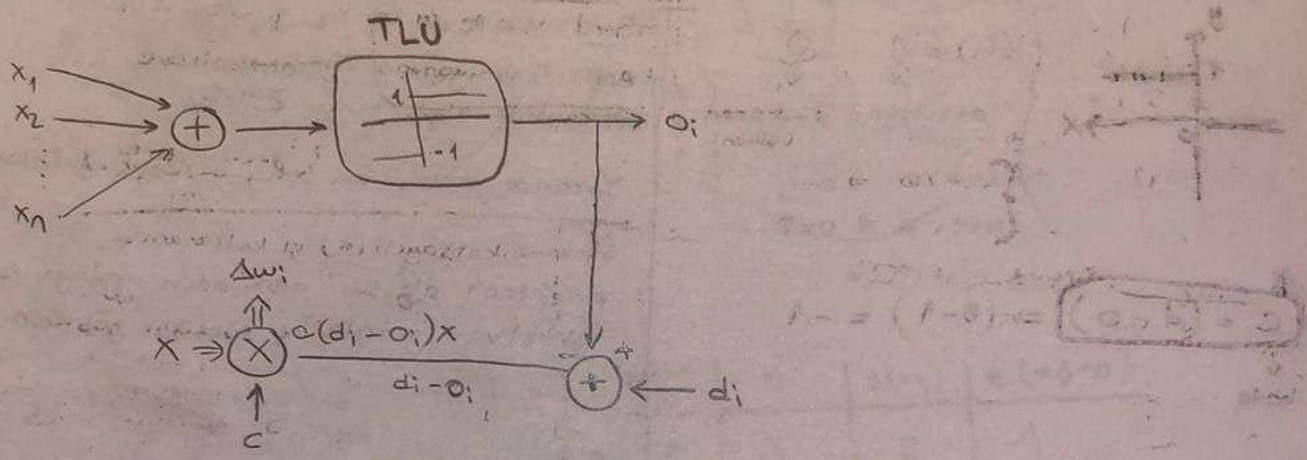
(!)a. Yapay sinir ağlarında bias nedir? b. Yapay zekada bias nedir?

- nöron aktivasyonunu etkilemek için kullanılır.
- Her nöronun kendi özel bias değeri vardır.
- model çıktısını etkiler, Ağı daha esnek ve uyumlu hale getirir.

- modelin veri üzerindeki yanlılığını ifade eder
- modelin belirli bir sonucu tercih etme eğilimini ifade eder.
- modelin genelleme yeteneğini etkileyebilir. gerçek dünyadaki performansını azaltabilir

② Çok katmanlı Sinir Ağları

Tek katmanlı Perceptron



$$NET = w_i^t * x$$

$$o_i = sgn(NET) = sgn(w_i^T * x)$$

$$r = d_i - o_i$$

$$\Delta w_i = crx = c\,[d_i - sgn(w_i^t * x)]x$$

$$w = w + \Delta w_i$$

x: giriş sinyalleri
w: ağırlık katsayıları
Δwi: ağırlık katsayıları farkı
r: öğrenme sinyali
d: beklenen çıkış değerleri
o: elde edilen çıkış değeri
i: örnek numarası (x1, x2...)

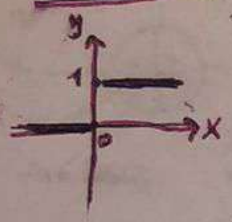
★ ★ ★★ ★ ★ ★ ★ ★ ★ ★ ★

Biz veriyoruz,
Yapay Sinir Ağı ağırlıkları (öğrenir)

örn. 1.0 + 1.0 + 1.1 = 1 = f(1)
(veriyor)
net Sign

x_1	x_2	AND (y)	NAND	OR	XOR
0	0	0	1		
0	1	0	1		
1	0	0	1		
1	1	1	0		

Tek katmanlı
Perceptron Öğrenme Algoritması formülü:



$f(1) = 1 \quad 0$
↓ ↓
o-output d-desired (istenen)

Net > 0 ⇒ o = 1
Net < 0 ⇒ o = 0

★ $e = (d - o) = (0 - 1) = -1$
↓
hata

★ $w_n = w_{n-1} + e \cdot lr \cdot x$
↓
α - learning rate (öğrenme katsayısı)

örn.
$[w_1, w_2, w_3]$

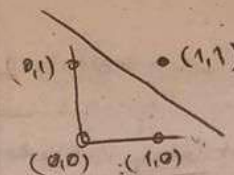
$$w_0 \begin{bmatrix} 0 \\ 0 \\ x_3 = 1 \end{bmatrix} = NET \Rightarrow f(NET) \rightarrow output(o)$$

- Amaç ağırlık değerlerinin zaman içinde değişip iyileşmesi.

$w_1 = w_0 + e \cdot lr \cdot x$

$$\Rightarrow w_1 = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} + (-1) \cdot (1) \cdot \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

$$= \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = w_1$$



x_1	x_2	bias $x_3 = b = 1$	$y \rightarrow d$
0	0	1	0
0	1	1	0
1	0	1	0
1	1	1	1

$w_0 = [1{,}5, 1{,}5, 1{,}5]$ (w_1, w_2, w_3)

$lr = 1 \rightarrow \alpha$
max-epoch = 10

Soru: Verilen x_1 ve x_2 girişleri için AND fonksiyonunu öğrenmek üzere perceptron algoritmasını kullanın.

Yukarıda verilen veri seti, başlangıç ağırlıkları (w_0), öğrenme katsayısı (α)'yı kullanarak perceptron alg.sını adım adım uygulayın ve ağırlıkların güncellenme sürecini gösterin.

(★ (Net, o, d, e) değişkenlerinin dördünün de değeri değişmiyorsa eğitim tamamlandı demektir.)

Çözüm:

1- $[0,0,1], w_0$

$x_1 \cdot w_0[0] + x_2 \cdot w_0[1] + x_3 \cdot w_0[2] = 0 \cdot 1{,}5 + 0 \cdot 1{,}5 + 1 \cdot 1{,}5 = 1{,}5$

NET	o (aktivasyon fonk.)	d(=y)	e(=d-o) (istenen hata)
1,5	1	0	-1

net > 0 ⇒ o = 1 oldu

(x_1, x_2, x_3, y) deki y'yi buraya yazdık

d - o = 0 - 1 = -1 oldu

$w_1 = w_0 + e \cdot lr \cdot x = \begin{bmatrix}1{,}5\\1{,}5\\1{,}5\end{bmatrix} + (-1) \cdot 1 \cdot \begin{bmatrix}0\\0\\1\end{bmatrix} = \begin{bmatrix}1{,}5\\1{,}5\\0{,}5\end{bmatrix} \rightarrow w_1$

2- $[0,1,1], w_1$

$x_1 \cdot w_1[0] + x_2 \cdot w_1[1] + x_3 \cdot w_1[2] = 0 \cdot 1{,}5 + 1 \cdot 1{,}5 + 1 \cdot 0{,}5 = 2$

NET	o	d(=y)	e(=d-o)
2	1	0	-1

$w_2 = w_1 + e \cdot lr \cdot x = \begin{bmatrix}1{,}5\\1{,}5\\0{,}5\end{bmatrix} + (-1) \cdot 1 \cdot \begin{bmatrix}0\\1\\1\end{bmatrix} = \begin{bmatrix}1{,}5\\0{,}5\\-0{,}5\end{bmatrix} \rightarrow w_2$

3- $[1,0,1], w_2$

$x_1 \cdot w_2[0] + x_2 \cdot w_2[1] + x_3 \cdot w_2[2] = 1 \cdot 1{,}5 + 0 \cdot 0{,}5 + 1 \cdot (-0{,}5) = 1{,}5$

NET	o	d(=y)	e(=d-o)
1,5	1	0	-1

$w_3 = w_2 + e \cdot lr \cdot x = \begin{bmatrix}1{,}5\\0{,}5\\-0{,}5\end{bmatrix} + (-1) \cdot 1 \cdot \begin{bmatrix}1\\0\\1\end{bmatrix} = \begin{bmatrix}0{,}5\\0{,}5\\-1{,}5\end{bmatrix} \rightarrow w_3$

4- $[1,1,1], w_3$

$x_1 \cdot w_3[0] + x_2 \cdot w_3[1] + x_3 \cdot w_3[2] = 1 \cdot 0{,}5 + 1 \cdot 0{,}5 + 1 \cdot (-1{,}5) = -0{,}5$

NET	o	d(=y)	e(=d-o)
-0,5	0	1	1

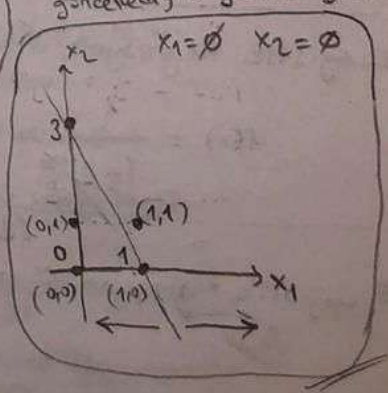
$w_4 = w_3 + e \cdot lr \cdot x = \begin{bmatrix}0{,}5\\0{,}5\\-1{,}5\end{bmatrix} + 1 \cdot 1 \cdot \begin{bmatrix}1\\1\\1\end{bmatrix} = \begin{bmatrix}1{,}5\\1{,}5\\-0{,}5\end{bmatrix} \rightarrow w_4$

5- $[0,0,1], w_4 = \begin{bmatrix}1{,}5\\1{,}5\\-0{,}5\end{bmatrix} \rightarrow$

Net	o	d	e
-0,5	0	0	0

6- $[0,1,1], w_5 = \begin{bmatrix}1{,}5\\1{,}5\\-0{,}5\end{bmatrix} \rightarrow$ 1 1 0 -1

perceptron algısının öğrenme sürecinde ağırlıklarını ve eşik değerlerini nasıl güncellediğini gösteren grafik



Lojistik Regresyon

- Giriş değerlerine göre olasılık hesaplar.
- Sigmoid kullanır.
- Loss fonksiyonu olarak ikili çapraz entropi (binary cross entropy) kullanır.
  (= log loss)

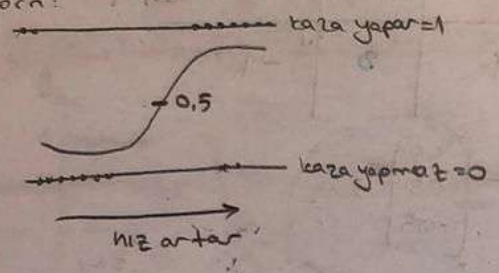
$$\mathcal{L}(y', y) = -(y \cdot \log(y') + (1-y)\log(1-y'))$$

- Cost fonksiyonu ortalama loss

$$J(w,b) = \frac{1}{m}\sum_{i=1}^{m}\mathcal{L}(y', y)$$

- ikili sınıflandırma prob.leri çözmekiçin kullanılan ML alg.

örn:



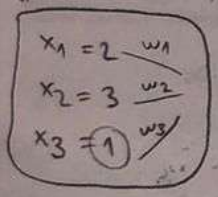
Lojistik Regresyon'da Doğrusal model

$y = wx + b$

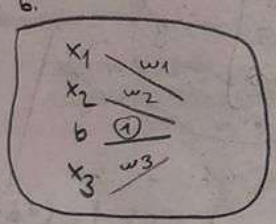
- w: ağırlık matrisi
- x: giriş

formel gösterimi:

a.



b.



- İki gösterim de aynıdır.
- Sadece b gösterimi lojistik regresyon'da daha yaygın kullanılır.

✱ Lojistik Regresyon, olasılığa dayanır.

$$f(x) = \frac{1}{1+e^{-x}} \rightarrow \text{lojistik fonksiyon } (= \text{Sigmoid fonksiyon})$$

✱ Lojistik Regresyon ≅ Perceptron

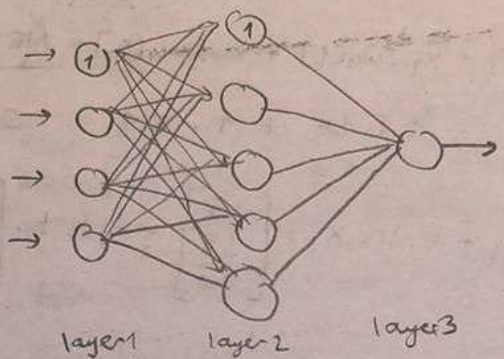
$$\mathcal{L} = -(y \cdot \log\hat{y} + (1-y)\log(1-\hat{y}))$$

(loss fonk.)

$o \rightarrow \hat{y}$ } elde edilen
(aktivasyon fonk.)

$d \rightarrow y$ } beklenen
(disered = istenen)

Çok katmanlı Sinir Ağları (ÇKSA)



- ÇKSA, YZ ve ML alanlarında yaygın olarak kullanılan bir tür yapay sinir ağıdır.
- Bu ağlar insan beyninin yapısından esinlenilerek tasarlanmıştır. Ve karmaşık problemleri çözme yeteneğine sahiptir.

İleri Yayılım

İleri yayılım, çok katmanlı sinir ağlarında bilginin giriş katmanından çıktı katmanına doğru nasıl aktığını tanımlayan bir işlemdir.

★ İleri Yayılım Temel Adımları:

1. Giriş katmanı: veriler giriş katmanındaki nöronlara girer.
2. Ağırlıklarla çarpma: Her nöron gelen verileri kendi ağırlıklarıyla çarpar.
3. Toplama: Çarpılan değerler toplanır. Ve net giriş değeri oluşur.
4. Aktivasyon fonksiyonu: Net giriş değeri, aktivasyon fonksiyonundan geçerek nöronun çıkışını belirler.
5. Çıkış değeri: Aktivasyon fonk.dan değer, nöron çıkış değeri olur.

6. Bir sonraki katmana Aktarma: Çıkış değeri, bir sonraki katmandaki nöronlar için giriş sinyali olarak kullanılır.

7. Tekrarlama: Aynı adımlar tüm katmanlarda tekrarlanır.

8. Çıktı: Son katmandaki nöronların çıkış değerleri, modelin tahminlerini oluşturur.

Geri Yayılım

- İleri yayılım ardından, geri yayılım işlemi gerçekleşir.
- Geri yayılımda modelin tahminleri, gerçek değerlerle karşılaştırılır ve aradaki hata hesaplanır.
- Hesaplanan hata değeri, modelin ağırlıklarını ve önyargılarını güncellemek için kullanılır.

Geri Yayılımda Kullanılan En Yaygın Hata Fonksiyonları

★1. Ortalama Kare Hata (MSE):
Model tahminleri ile gerçek değerler arasındaki ortalama kare farkı hesaplar.

$$MSE = \frac{1}{2}(d_i - o_i)^2$$

$\frac{1}{2}$ ↓ n: örnek

$(d_i - o_i)^2$: pozitif olması için kare alınır

} hocanın tahtaya yazdığı

$$MSE = \frac{1}{n} \cdot \sum (y_i - t_i)^2$$

n: veri kümesi örnek sayısı

y_i: modelin örnek için tahmini

t_i: örnek için gerçek değer

2. Çapraz Entropi:

3. Kullback-Leibler Divergence (KL Divergence)

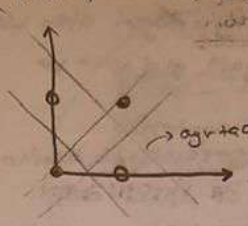
XOR Problemi

❀ Geri yayılım, çok katmanlı bir sinir ağı ile kullanıldığında XOR problemini çözebilir.

❀ Geri yayılım algısı, ÇKSA'daki her bir ağırlığın ve önyargının hata oranına göre nasıl güncelleneceğini belirler. Bu sayede model zamanla XOR prob. çözer

❀ XOR problemi, tek katmanlı perceptron ile çözülemez çünkü " " " modeli doğrusal ayırıcı. XOR problemi ise doğrusal olmayan bir problem.

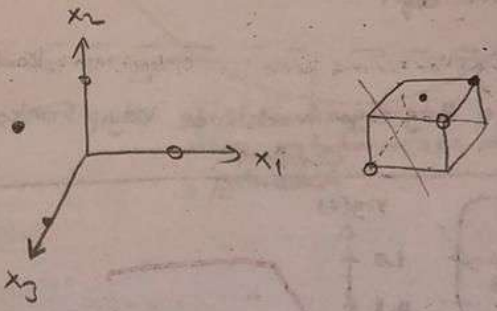
(XOR problemi: iki girişli mantıksal kapı.)



0, 0	1
0, 1	0
1, 0	0
1, 1	1

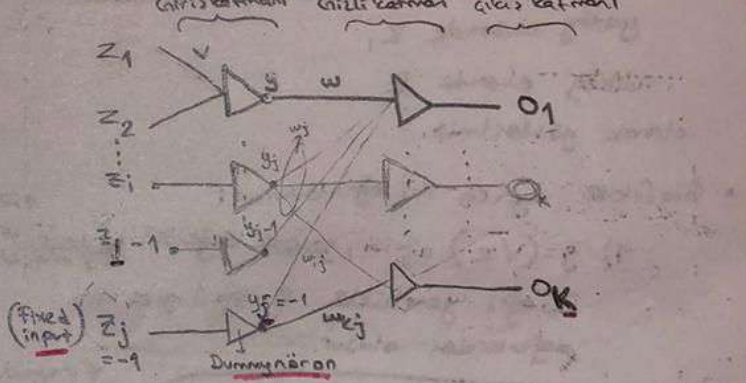
} x ≥ 0 = 1, x < 0 = 0

★XOR problemini çözmek için, geri yayılım alg.ması bulundu. Sonra ÇKSA kullanılabilir hale geldi. (1970)



★3. boyut eklendi ve bu sayede katman eklendikçe daha çok şey öğrenebiliyoruz.

Çok Katmanlı Sinir Ağı modeli Temel Yapısı Görseli



z → giriş bilgileri / giriş parametreleri

V → ilk katman ağırlıkları

y → ilk katman çıkışı / ikinci katman girişi

w → ikinci katman ağırlıkları (çıkış katmanı)

O → çıkış / output

d → beklenen / desired

I → toplam giriş verisi sayısı ($i = 1, 2, \dots, I$)

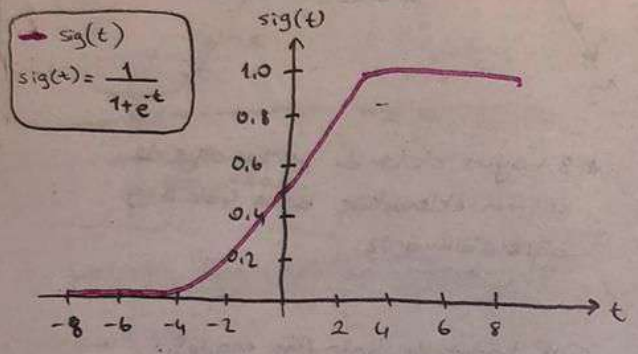
J → ilk katmandaki nöron sayısı ($j = 1, 2, \dots, J$)

K → son katmandaki nöron sayısı ($k = 1, 2, \dots, K$)

Fixed input:
- Her zaman aynı değeri alan bir nöron
- Bu değer genellikle 0 veya 1 gibi sabit bir değerdir
- Özel bilgileri modele girmek için kullanılır
- Herhangi bir veri girdisi ile ilişkili değil.

Dummy neuron:
- Herhangi bir bağlantısı olmayan bir nöronu simgeler
- Bu nöron 'herhangi bir giriş sinyali almaz' ve 'herhangi bir çıktı üretmez'
- Giriş katmanındaki herhangi bir nöronla bağlantılı değil

Lojistik Regresyon modelinde kayıp Fonksiyonu Grafiği



- Yukarıdaki grafikte kayıp fonksiyonu:
  yatay eksende Z,
  dikey eksende E
  olarak gösterilmiş.
- Grafikte ayrıca iki eğri bulunur:
  1) $y=(Vz)$ eğrisi, gerçek değerleri gösterir. Bu eğri genellikle 0 veya 1 gibi ikili değerlerden oluşur.

$$y = T(Vz) = f(V^T z)$$

  2) $o = r(Wz)$ eğrisi, modelin tahmin ettiği çıktıları gösterir. Bu eğri sigmoid veya tanh gibi bir aktivasyon fonk. ile hesaplanır. (f: sigmoid, tanh, vb.)

$$o = T(Wz) = f(W^T z)$$

- Görseldeki grafikte kayıp fonksiyonu E olarak gösterilmiştir.
- Bu ortalama kayıp anlamına gelir.

(ortalama kayıp: tüm veri noktaları için kayıp fonk.nun ortalaması.)

$$E = E + \underbrace{\frac{1}{2}\|d-o\|^2}_{\text{loss fonksiyonu}}$$

Error cost

- Error cost, en sonda elde edilen.
  !min olması istenir.

Gradyan İniş

- Gradyan, bir fonksiyonun bir noktadaki eğimini gösteren bir vektördür.

- Gradyan iniş algoritması, bir fonksiyonun minimum veya maximum değerini bulmak için gradyanı kullanır.

* Gradyan iniş, hataların türeve göre ağırlıklandırılması.

- Gradyan hesabında kullanılan formüller.

$$w := w - \alpha \frac{\partial J(w,b)}{\partial w}$$

→ w: ağırlıklar
→ w ağırlığına göre kısmi türev

$$b := b - \alpha \frac{\partial J(w,b)}{\partial b}$$

→ b: bias
→ b değerine göre kısmi türev

Formüllerin Gradyan İniş Alg.sında kullanımı:

1. Başlangıç değerleri w ve b için fonksiyonun gradyanı hesaplanır.
2. Gradyan, fonksiyonun min veya max değerine doğru bir adım atmak için kullanılır.
3. Adım atıldıktan sonra fonksiyonun gradyanı tekrar hesaplanır.
4. Bu adımlar, fonk.nun min veya max değeri bulunana kadar tekrarlanır.

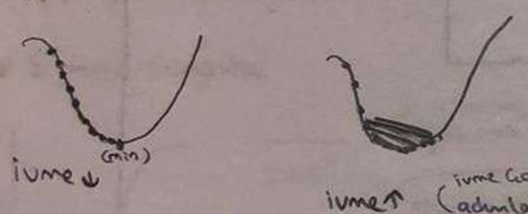
Slayt-2, sayfa 10-17

- Gradyan iniş matematik kuralları
- zincir türev kuralı
- kısmi türev

Kısmi türev: bir fonksiyonun bir değişkene göre kısmi türevi, Diğer değişkenler sabit tutulurken fonksiyonun o değişkene göre türevi

↓

Uzun mat. formüller vardı yazmadım.



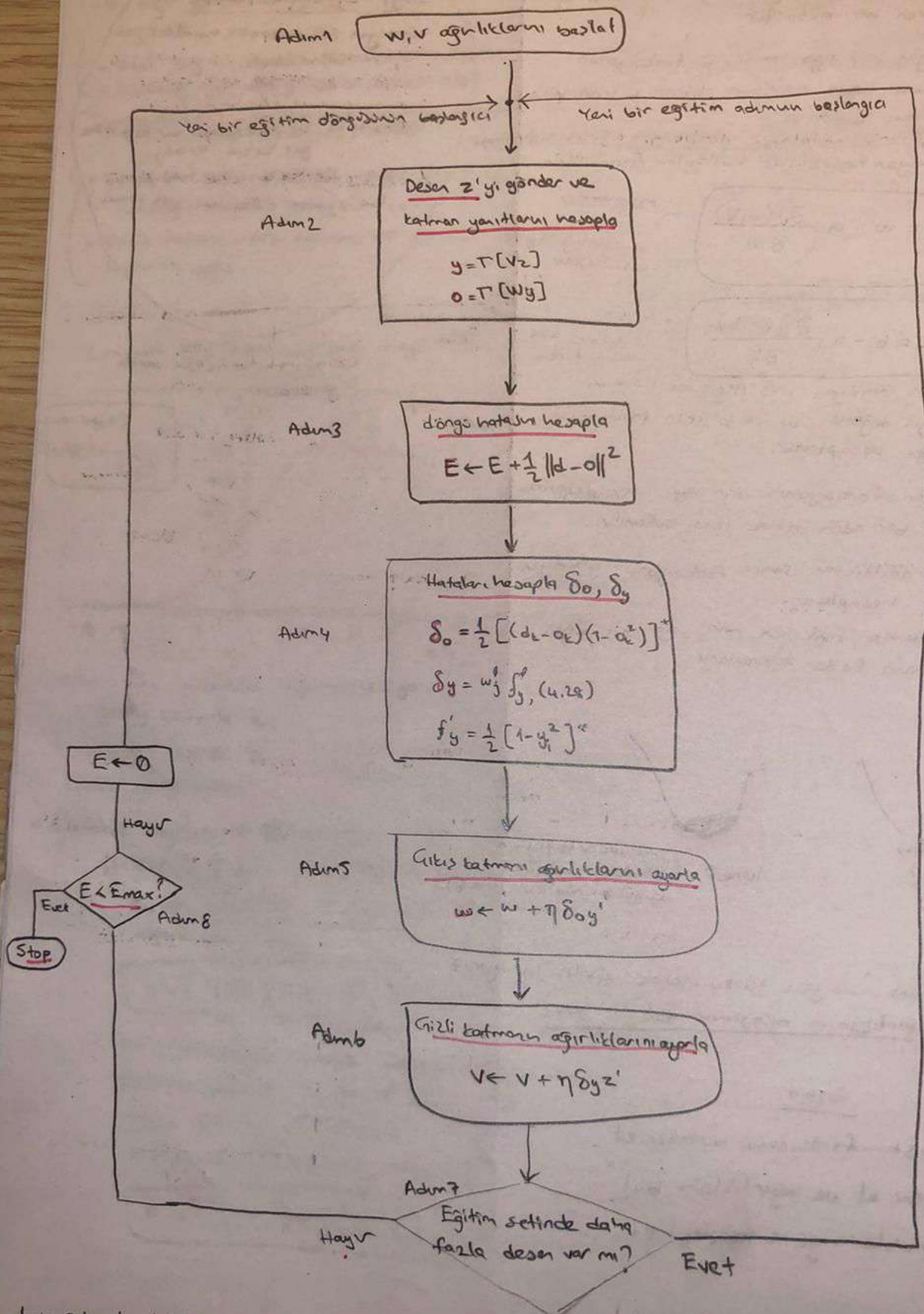
(ivme çok arttığı için adımları büyük büyük olduğundan min'a ulaşamaz)

- coast fonk.nuna göre türev alarak ağırlık buluyoruz ve bu *ağırlıkların değişimine göre ilerliyoruz.

Özet

1- coast fonk.nunu minimize et
2- türev al ve ağırlıkları bul
3- ağırlıklara göre hareket et

Hata Geri Yayılımı (Back propagation) Döngüsü Akış Şeması



! 4. Adımda tablodaki denklemler değil şu denklem kullanılır: $\delta_o = [(d_k - o_k)(1 - o_k)o_k]$, $f_y' = [(1 - y_j)y_j]$

③ Aktivasyon Fonksiyonları

- Yapay sinir ağlarında (YSA), aktivasyon fonk.nu bir nöronun aldığı girişleri işleyerek üretteği çıktıları belirleyen matematiksel fonksiyon.
- Bu fonksiyonlar, nöronların doğrusal olmayan davranışlar sergilemesini sağlar. Bu sayede YSA'nın karmaşık problemleri çözmesini mümkün kılar.

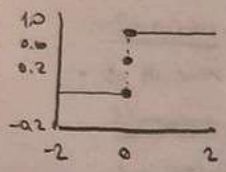
Aktivasyon Fonk. Temel İşlevleri

- Lineer olmayanlık katmak (doğrusal olmayanlık)
- Çıkış aralığını sınırlamak (0-1 / -1,1)
  ↳ (nöronlar arasında uyum sağlar)
- karar verme yeteneği kazandırmak
  ↳ (nöron aktif/değil, sinyal ilerlemesine yön verir)

★ Gradyan hesaplamasında kullanılır.

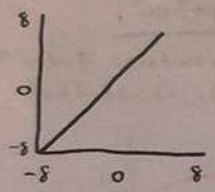
★ Amaç: Non-linearlik (doğrusal olmayanlık) eklemek.

♡ **Binary Step Function**



$$\begin{cases} f(x)=1, & x \geq 0 \\ f(x)=-1, & x<0 \end{cases}$$

♡ **Lineer Fonksiyon**



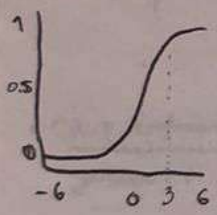
$f(x) = x$

↓

net

- Eğitim yapmak mümkün değil

♡ **Sigmoid / Logistic**



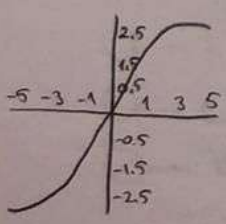
$\frac{1}{1+e^{-x}}$

→ x, 3'ten büyük ise gradyan değeri çok küçük

→ x, 3'ten küçük ise gradyan değeri büyür

★ Gradyan inişte kullanılabilir

♡ **Tanh (Hiperbolik Tanjant)**

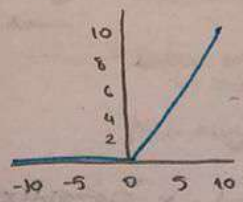


$\frac{e^x - e^{-x}}{e^x + e^{-x}}$

- Hesaplama karmaşıklığı var
- Sigmoid'e göre daha hızlı yakınsar Daha iyi sonuçlar verir.

★ Gradyan inişte kullanılabilir.

♡ **ReLU**



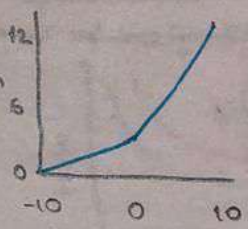
$f(x) = x, \quad x > 0$

$f(x) = 0, \quad x \leq 0$

- Belirli bir değerden sonra her 0 atandığında bir süre sonra öğrenmeyi durdurur. ve nöron ölmüş olur.
- 0 → dying neuron (ölü nöron)

♡ **Leaky ReLU**

$f(x) = \max(x, 0{,}01x)$



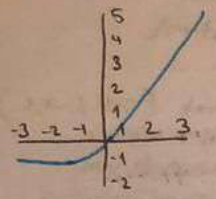
- $f(x) = \max(x, 0)$, ölü nöronlar oluşmasına sebep oluyor. Bu sorunu çözmek için 0,01 değeri ile çarpılan formül bulunmuş.
- parametrik relü:

$f(x) = \max(ax, x)$

- a (alfa) için sürekli yeni bir hesaplama gerektirmiyor.
- Bu bir dezavantaj. Çünkü sisteme ekstra iş yükü biner.

ELU

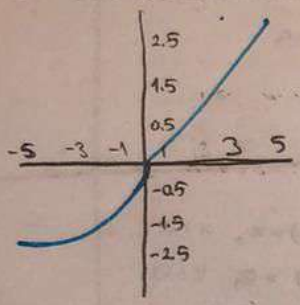
$$\begin{cases} \alpha_x^x (e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$$



ELU'daki α, sabit değer.
(biz belirliyoruz)

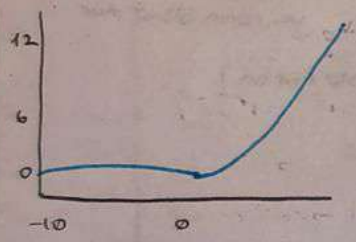
SELU

$$f(\alpha, x), \lambda \begin{cases} \alpha (e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 1 \end{cases}$$



- Gizli katmanların çıkışını normalize etmeyi sağlar.
- Bu her katmanın kendi kendine öğrenmesini sağlar.

Swish



$$f(x) = x \cdot \text{sigmoid}(x)$$

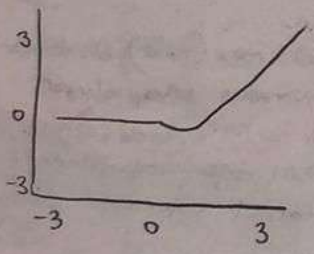
$$= \frac{x}{1+e^{-x}}$$

- Sigmoid'den daha iyi performans

GELU

- Hesaplama yükü fazla

Bu yüzden her zaman kullanılmıyor



Aktivasyon Fonksiyonlarındaki Sorunlar

Sigmoid ve Tanh → 1. Vanishing Gradient (kaybolan Gradyanlar)

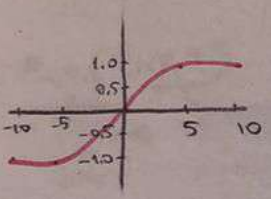
- Sigmoid ve Tanh gibi aktivasyon fonksiyonları, giriş değerleri çok büyük veya çok küçük old.da gradyan (eğim) değerini sıfıra yaklaştırabilir.
- Bu durum sinir ağı eğitimi sırasında parametrelerin güncellenmesini zorlaştırır. ve modelin öğrenmesini engeller.

ReLU → 2. Dying ReLU (Ölen ReLU)

- ReLU aktivasyon fonk. giriş değeri negatif old.da sıfır çıktısı verir.
- Bu durum sinir ağının bazı nöronlarının ölmesine veya eğitim sürecinde etkilerini kaybetmelerine neden olur.

Softmax

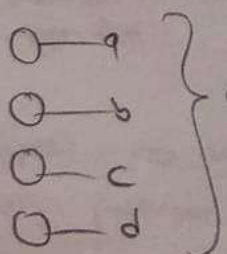
- Çıkış katmanlarında verilen değerlerin herbir farklı sınıfı için bir temsilci oluyor



- Herbir çıkışın toplam olasılığının 1 olmasını sağlar.

$$\frac{e^x}{\sum_1^2 e^x}$$

- Çıkış katmanları sınıflandırma problemlerinde yardımcı.



a+b+c+d = 1

Hiperparametreler

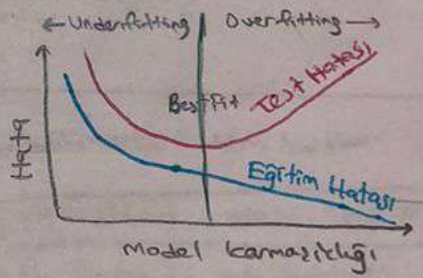
★ Aşırı öğrenmeyi engelleyebilmek için geliştirdiğimiz parametreler

- Öğrenme oranı (learning rate)
  ↳ parametrelerin ne kadar güncelleneceği
- Ağırlık azaltma (weight decay)
  ↳ aşırı öğrenmeyi engeller
- Dropout
  ↳ aşırı öğrenmeyi engeller
  ↳ eğitim sırasında rastgele nöronları devre dışı bırakır.
- Epoch sayısı
- Batch sayısı
  ↳ batch: tek seferde verilen örnek sayısı (batch↑ ağ yükü↑)
- Aktivasyon fonksiyonları

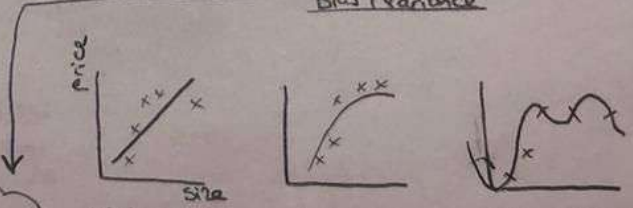
Loss Fonksiyonları

- Mean Squared Error (MSE)
- Binary Cross-Entropy
- Categorical Cross-Entropy Loss

Aşırı öğrenme — Eksik öğrenme
(overfitting) (underfitting)



Bias / variance



High bias (under fit) | Just right | High variance (overfit)

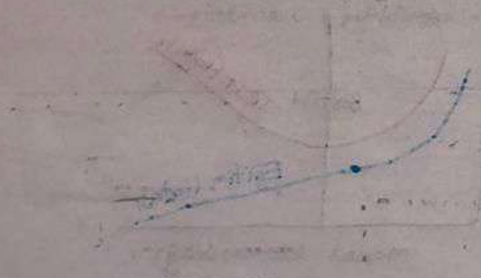
bias: gerçek değ. arasındaki ort. fark (bias↓ gerçeğe yakınlık↑)

varyans: tahminlerin değişkenliği (varyans↓ tahminlerin yakınlığı↑)

★ Ağın genelleştirme yeteneği yüksek olsun isteniyor.

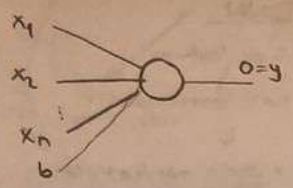
Optimizasyon Teknikleri

- Momentum : hızlı opt.
- AdaGrad : öğrenme oranı değiştirir
  (güncelleme sayı ↓ öğrenme oranı ↑)
- RMSprop : öğrenme oranı için momentum ekler.
- Adam : momentum + RmSprop
- AdaMax : Adam yönt.e benzer.
  max yerine infinity norm kullanır
- Nadam : Nestrov yönt. ile Adam yönt. iyileştirir
- Stokastik Gradyan İnişi (SGİ) :
  ↳ Gradient hesaplamada tüm veri yerine
  küçük batch'ler kullanır.



online | 27.03.2024

4 - Perceptron



x_1	x_2		d
0	0	1	0
0	1	1	0
1	0	1	0
1	1	1	1

0 1 (2)

AND problemi

x = np.insert(X[i], 2, 1) ile 2. değer eklediğ ve değeri 1 oldu (bias=1 değeri için)

$(d-o) = e$ → bu hata değeri ile* olması gerekenden ne kadar uzaktayız bulmayı sağlar.

- d: beklenen
- o: gerçek çıktı

- Buradaki fark 0 ise iyi;
- Ama fark 1-2 ise ($e \neq 0$)

* buradaki fark, ağırlıkların güncellenmesinde kullanılıyor.

$$w_2 = w_1 + \underbrace{\mu (d-o) x}_{\Delta w}$$

μ: öğrenme katsayısı

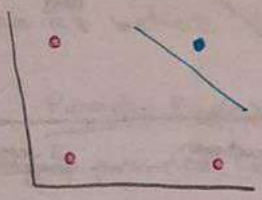
- net = np.matmul(w, x) → matrix multiplation

- Ayırtaç Doğrusu Nasıl oluşturulur?

w_1, w_2, w_3

Doğru denklemi: $w_1 x_1 + w_2 x_2 + w_3 = 0$

2 noktası bilinen doğru denkleminden (x_1 ve x_2'ye sırayla 0 vererek) bulunur



sınıfların birbirlerinden doğru şekilde ayrılmasını sağlar.

- e (hata), sıfıra (0) ulaşmazsa ne kadar süre icra edilecek?
  ↳ epoch sayısı

$o = f(NET)$ (Net: girişler ve ağırlıkların çarpımının toplamı)

- f: signume fonk.
- NET: $(w^T \cdot x)$ yada $(w^T \cdot x + b)$

(belli değerlerin 0, 1, -1 olarak kabul edildiği, ikili fonk.)

aktivasyon fonk

Bu bölümdeki tüm kodlar

perceptron_dersi.ipynb

adlı dosyada mevcuttur.

- .shape() → boyut bilgisi verir

Çıktı: (satır say, sütun say) → her boyuttaki eleman say. içeren tuple. numpy arrayle kullanılır.

- $e > 0$ → eğitim devam etmeli. w (ağırlıklar) güncellenir
- $e = 0$ → Eğitim tamamlandı. w değişmez.

5- Evrişimli Sinir Ağları (CNN)

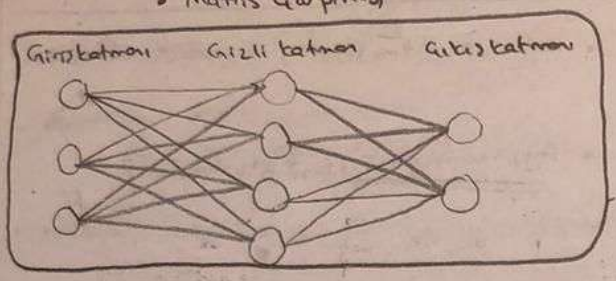
YSA:

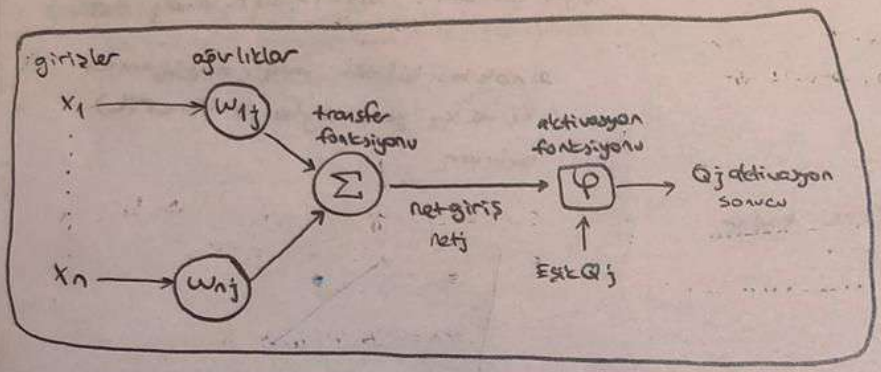
- Derin öğrenme
- Uygun ağırlık değerlerini bulur (Gradyan iniş)
- matris çarpımı

ANN	vs	DNN
az katman		çok katman
az karmaşıklık		çok karmaşıklık

↓

- Derin adı, katman sayısının fazlalığından gelir.

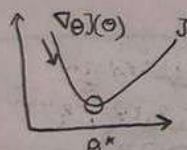




1 veya 2 gizli katman — 3 veya daha fazla gizli katman

Yapay Sinir Ağları (ANN)
(basit regresyon, sınıflandırma problemleri)

- Geri yayılım (Gradyan iniş) kullanır.



- Klasik sinir ağları, kendi özellik çıkarmaz.
- Problem odaklı değil
  ↳ Girdi verisinin yapısını kullanmaz
- Ağırlık sayısı artışı geometrik
  ↳ daha düşük genelleme yeteneği
  ↳ daha fazla overfitting

Derin Sinir Ağları (DNN)
(görüntü tanıma, NLP, ses tanıma)

- Günümüzde başarılı şekilde kullanılmasında iki etken:
  1- Yeterli miktarda veri
  2- Yeterli işlem gücü
- Derin ağları işleme:
  - Farklı alanlarda başarıyla uygulanır (bilgisayarlı görü, görüntü işleme, zaman serileri, ilaç tasarımı, ses tanıma, ...vb.)
  - DÖ ilerledikçe yeni derin ağ mimarileri ortaya çıkmıştır: (Evrişimli Sinir Ağları, Derin inanç Ağları, RNN/LSTM, GAN, Transformer, VAE, ... vb.)

Evrişimli Sinir Ağları (CNN)

- Özelleşmiş derin sinir ağı mimarisi
- Izgara tipi topoloji:
  örn: zaman serisi (1 boyutlu), görüntüler verisi (2 boyutlu)
- ★ Görüntüler, ağa direkt verilebilir. Bu sayede diğer öğrenme alg.larında gereken özellik çıkarımı'na burada gerek yok.
- Evrişim adı, veriler matematiksel bir işlemin gerçekleştirildiğini gösterir. (konvolüsyonel)
- ★ CNN ağlarında veriler önişleme geçer. Özellik çıkarımını ağ yapar.

Evrişim işlemi

Veri süreklilikten:

$$\begin{cases} s(t) = \int x(a)\, w(t-a)\, da \\ s(t) = (x * w)(t) \end{cases}$$

Veri ayrık gelince:

- Ayrık Evrişim

$$\begin{cases} s(t) = (x * w)(t) \\ \quad = \sum_{a=-\infty}^{\infty} x(a) w(t-a) \end{cases}$$

(örn: saniyede 1 örnek alınır)

(x: giriş
w: çekirdek
s: özellik haritası (feature map))

- ML uygulamalarında:

  giriş → verinin çok boyutlu dizisi

  çekirdek → Öğrenme alg.sı tarafından uyarlanan parametrelerin çok boyutlu dizisi (=tensör)

Girişler ile Çıkışlar Arasındaki Etkileşim
ANN vs. CNN

Klasik Sinir Ağları (ANN)	Evrişimli Sinir Ağları (CNN)
Tüm girişler, tüm çıkışları	
etkiler	etkilenez

CNN'de üç ana fikir

- Seyrek Etkileşim
- Parametre Paylaşımı
- Eşdeğişkenlik

Evrişim işlemi Özellikleri

- Birden fazla eksende aynı anda yapılabilir.

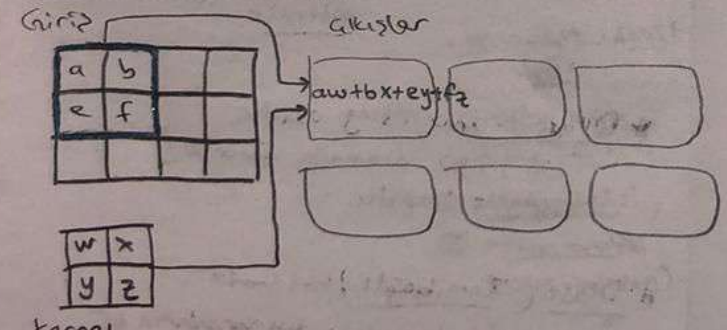
$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)\, K(i-m, j-n)$$

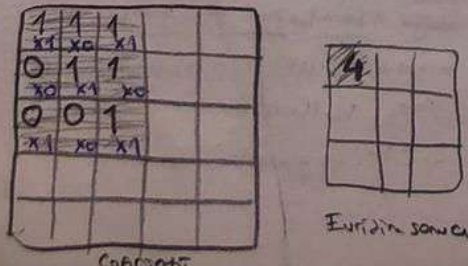
- Değişme özelliğine sahip.

$$S(i,j) = (K * I)(i,j)$$

★ Pratikte çapraz korelasyon (cross-correlation) kullanılır. Tek farkı çekirdek ile giriş yer değiştirmeyecek şekilde düzenlenmiş.

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(i+m, j+n)\, K(m,n)$$





Seyrek Etkileşim

- Çok az sayıda parametrenin tutulması
- daha az bellek ve işlem

Parametre Paylaşımı

- Birgrup parametre, her yer için öğrenilir.

  Bu sayede:
  - işlem say. değişmez
  - bellek kullanımı azalır.

(ANN'de her parametre 1 kez kullanılır)

Eşdeğişkenlik

- CNN, ötelemeye karşı eşdeğişimli. Yani giriş değişirse çıkış da değişir.

$$f(g(x)) = g(f(x))$$

$$T_{\Delta x}(C_K(f)) = C_K(T_{\Delta x}(f))$$

($T_{\Delta x}$: ötelere işlemi; $C_K(f)$: evrişim işlemi)

Buna göre ötelemenin:
- konvolüsyon işleminden sonra uygulanması
- CNN işleminde çekirdeğe "
- CNN " girişe "

ile elde edilen sonuçlar eşittir

17 ★ CNN, hangi aşamada uygulanırsa uygulansın sonuç değişmez.

Uygulamada CNNs

- Görüntüler genişlik, yükseklik ve derinlik (RGB) olarak 3 boyuta sahip.
- ★ Gerekli işlemeyi sağlamak için, çekirdek de 3 boyutlu olmalı.
- CNN'de birden fazla çekirdek kullanılır.

(RGB)
0-255

- CNN üç aşama içerir:

1- (Çok sayıda) Evrişim Uygulaması
(kerneli gezdir → harita elde et)
2- Aktivasyon fonk uygula
★ algılayıcı (detector) aşaması
3- Havuzlama (pooling) uygula

katmanlara girişlerin verilmesi
↓
Evrişim
↓
Algılayıcı
↓
Havuzlama
↓
Sonraki katman

Algılayıcı

- Non-linearlik sağlar (çıkışlarda)

Havuzlama

- Ağın bir noktadaki çıkışı, komşu çıkışların bir özeti ile değişir.
- küçük ötelemelere, değişikliklere karşı esneklik sağlar
- probleme göre değişebilir:
  - max pooling → dizi 0 ile doldur (etkisi olmaması için)
  - min pooling → " 255 " "
  - ortalama pooling

Adım aralığı (stride) → işlem verimliliği
Sözde doldurma (padding) → havuzlama sonucu boyut değişmemesi için

CNN'de Hiperparametreler

Çekirdek (filtre) boyutu → genelde 3×3
Filtre sayısı
Adım aralığı
Sözde doldurma

★ Bu değerler problemle göre seçilir.
★ Filtre say genelde ikinin katları olarak (32-1024 arasında) seçilir.
★ Çok filtre, ağı güçlendirir
Eğitim süresi artar (risk)

Tam Bağlı Sinir Ağları Aşaması

- Klasik sinir ağı yapısı
- Çok boyutlu işlem yapılınca:
  son havuzlama'dan sonra düzleştirme (flatten) gerekir.

★ Düzleştirme (Flatten): çok boyutlu veri, yan yana sıralanır 1 boyutlu vektör haline getirilir.

★ Dropout: Overfitting azaltır.
%25 / %50 oranında nöronları rastgele kapatır.

★ Dense (Tam bağlı) katmanlar:
önceki katmandaki tüm nöronların bir sonraki katmandaki tüm nöronlarla bağlı olduğu 'tam bağlı' bir yapı.
*non linearlik için aktivasyon fonk. kullanılmasını sağlar
(... başka görevleri de var)

⑥ CNN & ANN

ANN_Delta.ipynb

CNN_adimlari.ipynb

---

online | 05.05.2024 | ⑦

RakamTanima2_CNN.ipynb

[0., 0., 0., 1., 0., 0., 0., 0., 0., 0.]

↳ 3

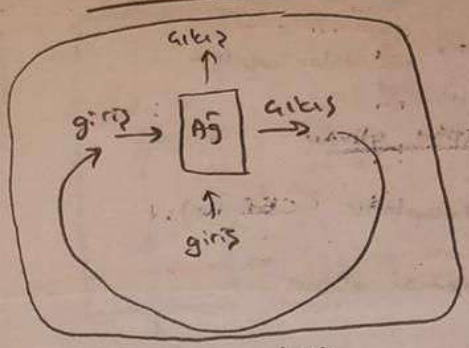
---

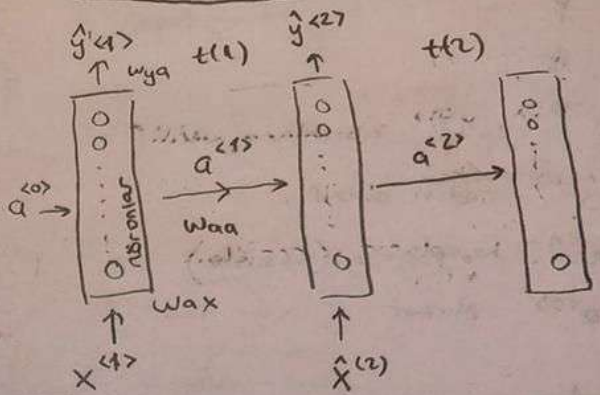
online | 11.05.2024 | ⑧

HayvanTahminModeli.ipynb

GereRöntgeriGörüntüleri k-E Tanima.ipynb

RNN Ağ Yapısı





Çıkışların hesaplanması (aktivasyon fonk.(tanh))

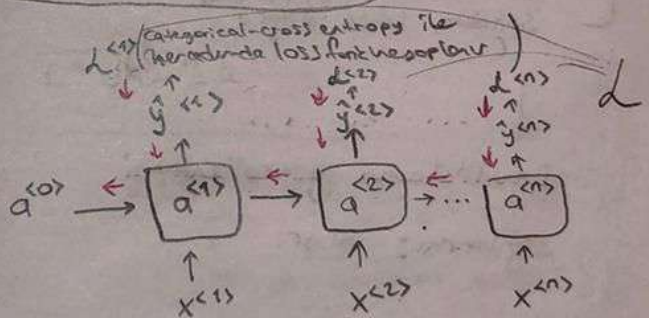
$$a^{\langle 1\rangle} = g_1(W_{aa} * a^{\langle 0\rangle} + X^{\langle 1\rangle} * W_{ax} + b_a)$$

$$y^{\langle 1\rangle} = g_2(W_{ya} * a^{\langle 1\rangle} + b_y)$$

$$\begin{bmatrix} W_{aa} : W_{ax} \end{bmatrix} \begin{bmatrix} a^{\langle t-1\rangle} \\ x^{\langle t\rangle} \end{bmatrix}$$ → Waa ve Wax birleştirilmesi

---

CNN	vs.	RNN
• görüntü verileri için		• giriş–çıkış sabit değil
• giriş – çıkış aynı		• zamana bağlı çıkış değişir.
• zamana bağlı çıkış değişmez		⊛ zaman serisi,
		⊛ metin analizi (cümle üzerinden)
		↙ Duygu Analizi ↘ NER (adlandırılmış varlık tanıma)
		⊛ Finans verileri analizi



$$\mathcal{L}^{\langle 1\rangle} + \mathcal{L}^{\langle 2\rangle} + \dots \mathcal{L}^{\langle n\rangle} = \mathcal{L}$$

loss fonk.ları toplanır. Toplam kayıp hesaplanır.

CCE (Categorical Cross Entropy).

$$-y^{\langle t\rangle} \log \hat{y}^{\langle t\rangle} - (1 - y^{\langle t\rangle}) \log (1 - \hat{y}^{\langle t\rangle})$$

RNN Adımları

$a^{<0>}, x^{<1>}$ nöronlar verilir

$\hat{y}^{<1>}$ çıktısı oluşur

$\mathcal{L}^{<1>}$ hesaplanır (CCE ile)

$a^{<1>}$ çıktısı oluşur

↓

⋮

↓

$a^{<n-1>}, x^{<n>}$ nöronlara verilir

$\hat{y}^{<n>}$ çıktısı oluşur

$\mathcal{L}^{<n>}$ hesaplanır (CCE ile)

$a^{<n>}$ oluşur

↓

$\mathcal{L}^{<1>} + \dots + \mathcal{L}^{<n>}$ toplanır

Toplam loss hesaplanır.

↓

Her loss değeri için zamanda geriye dağıtım

Backpropogation through time

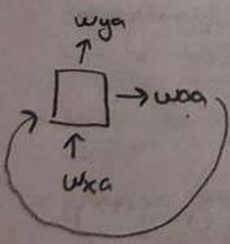
yapılır

sondan başa doğru.

↓

Ağırlıkların güncellenmesi sağlanmış olur.

RNN'de Problem



waa her seferinde başa döndürülüp kullanılıyor

her seferinde tekrardan sürece dahil ediliyor.

waa > 1 olursa herseferinde çarpıma dahil olur.

örn: 50 kere dahil edildi $(waa)^{50}$ alınır.

n waa için $(waa)^n$

n büyükse Gradyan patlaması olur.

↓

- gradyanlar anormal şekilde artar.

Ağın öğrenmesini olumsuz etkiler:

- global'in yakalanamayabilir
- gradyanlar 0'a çok yakın değer alıp gradyanlar kaybolabilir.

= Gradyan vanishing

Bu problemin önüne geçebilmek için LSTM önerilmiş (Long short term memory)

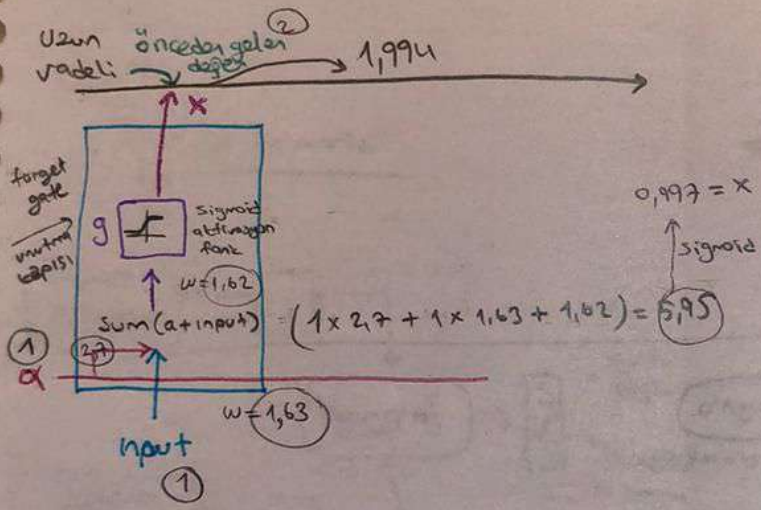
online | 18.05.2024 | (9) RNN & LSTM

Tekrarlayan YZ ders kitabı – Stanford.edu
(CS-230 – Derin Öğrenme)

↳ Geleneksel RNN mimarisi
↳ RNN uygulamaları
↳ kayıp fonk
· Zamanla ileri yayılım
· kaybolan/patlayan Gradyan

LSTM – (Long Short Term Memory)

• Uzun kısa Vadeli Bellek

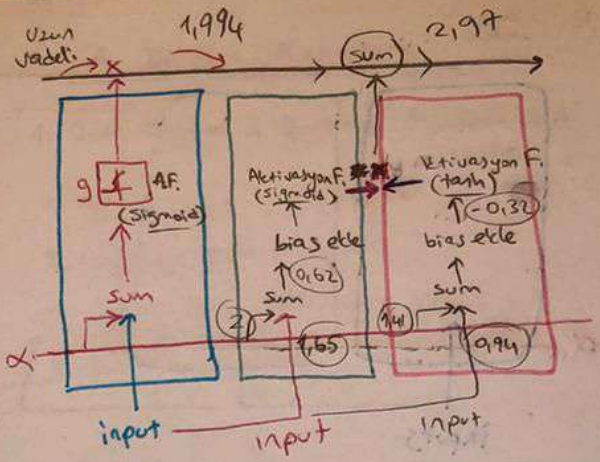


(önceden gelen değer) * x = (elde edilen değer)

↳ uzun vadeli bellek üzerinden aktarılır.

2 * 0,997 = 1,994

• elde edilen değer 0'a çok yakınsa unutulur.
• " " " ile ne unutulmalı ne hatırlanmalı belirlenir.



• y değeri [-1,1] arasında -1'e yakınsa unutmaya neden olur.

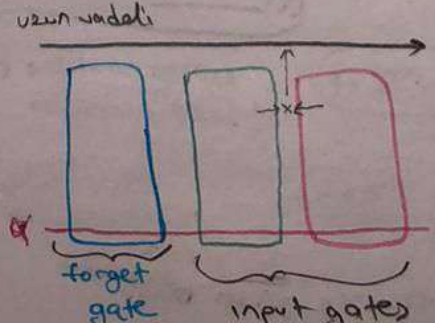
1,41 + 0,994 - 0,32 = 2,03
↓ tanh
0,97 = y

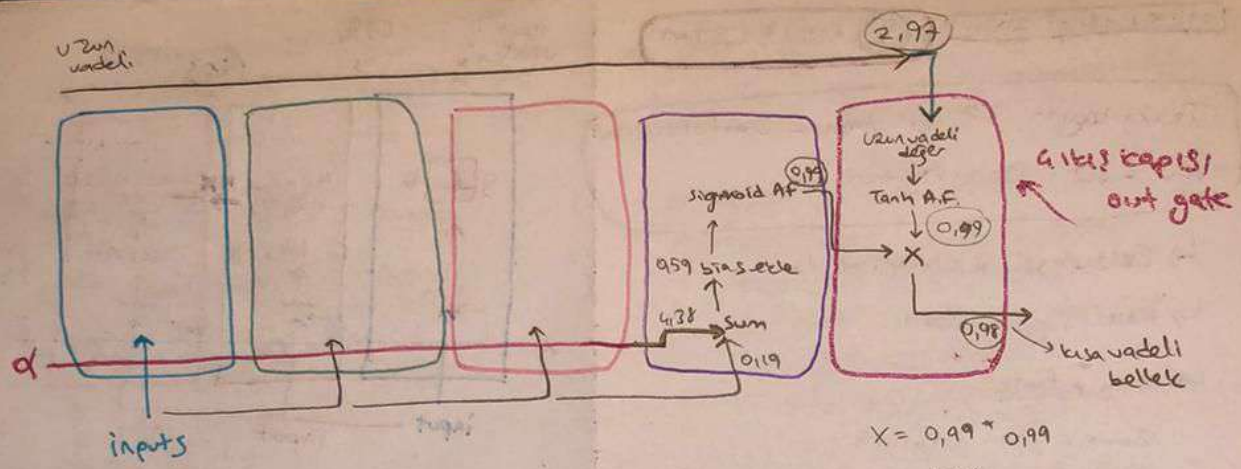
1*2 + 1*1,65 + 0,62 = 4,27
↓ sigmoid
1'e çok yakın değer = 1 dedik

X = 1 * 0,97 'deki 0,97 'nin tamamı uzun vadeli öğrenme kapısının bilgisine etki edecek

1 * 0,97 ≈ 2

2 + 0,97 = 2,97
↓
1,994 yazmak gerek bence (?)

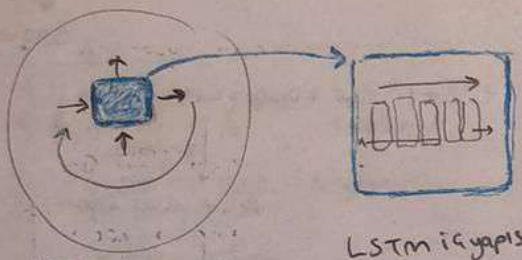




$X = 0{,}99 * 0{,}99$

$= 0{,}98$

- Bu LSTM'in iç yapısıdır.
- LSTM'in dış yapısı RNN'in dış yapısı ile aynıdır.



LSTM dış yapısı

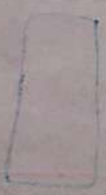
LSTM iç yapısı

t vektörü

LSTM Sentiment Analysis | keras.ipynb

↳ get_dummies etiket değerlerini 0-1 olarak ayırır

IMDB RNN Modeling.ipynb



loss düşük, accuracy yüksek olmalı
(hata) (doğruluk)

⑩ GANs

Üretici YZ

- ortada olmayan şeylerin ortaya çıkarılması

Üretici Çekişmeli Ağlar (GANs) - (GÜA)

- ANN
- CNN
- RNN

Uygulama alanları:

- ML yöntemleri için veri arttırma
- Yüz görüntüsü üretme
- Görüntü - görüntü dönüşümü
- metin - görüntü "
- süper çözünürlük

... vb.

GÜA Eğitim Adımları

1. GÜA mimarisi Belirle (Probleme Göre seçilir)
2. Ayırıcı Eğit
3. Üreticiyi Eğit
4. Tekrar Et (200 - 400 epoch)
5. Modeli kaydet

GÜA Problemler

★ 1- Mod Göçmesi (mode collapse):
GAN'ların sürekli benzer/aynı türde örnekler üretmesi.

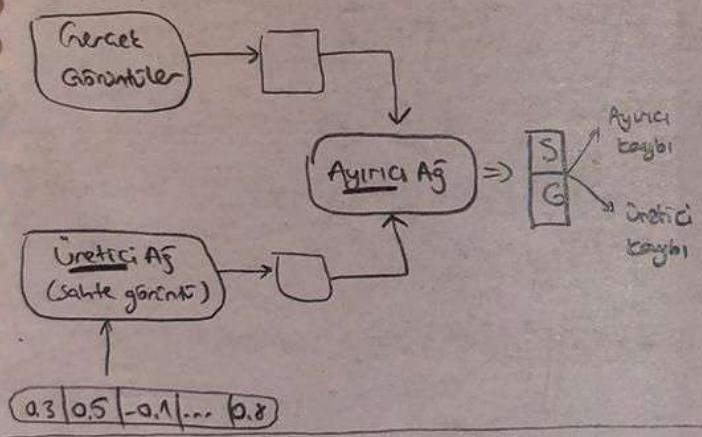
2- Eğitim kararsızlığı (Training Instability):
Üretec ve ayırıcı ağların eğitim sürecinde uyumsuzluk ve kararsızlık yaşanması

Ek zorluklar

★ - Kaybolan Gradyanlar (Vanishing Gradients):
Ayırıcı ağ çok başarılı,
üretici ağ için gradyanlar çok küçük

- yavaş eğitim süreci:
- Aşırı uyum (overfitting):

GÜA Mimarisi



min-max Game

$$V(D,G) = E_{x \sim p_{data}(x)}[\log D(x)] + E_{z \sim p_z(z)}[\log(1 - D(G(z)))]$$

- $D(x)$ → gerçeğin gerçek algılanma olasılığı
  ★ Üretici düşürmeye çalışır
- $D(G(z))$ → sahtenin gerçek algılanma olasılığı
  ★ üretici yükseltmeye, ayırıcı düşürmeye
- $1 - D(G(z))$
  ★ Üretici düşürmeye, ayırıcı yükseltmeye

GANs

G-PATE
medGAN
CorGAN
DP-CGAN
OCT-GAN
DP-WGAN
PacGAN
TableGAN
PostGAN Boost
PATECTGAN
DPGAN
DPCTGAN
CTGAN
DP-CGANS
CTAB-GAN
PATEGAN
RDP GAN

GAN_CIFAR_keras.ipynb

↳ generate_fake_samples()
sahte deger uretir
{ sahte - 0
gercek - 1