Gradyan Azalma (Gradient Descent)

Momentum Kavramı-Optimizer

2. türevi ortadan kaldırmanın sebebi maliyet

Momentumun temel işlevleri çıkabilir

3. Matematiksel Formül

Momentumun eklenmesi, klasik Gradyan İnişi güncelleme kuralını iki adıma ayırır:

1. **Hız Güncellemesi:**

$$v_t = \gamma \cdot v_{t-1} + \eta \cdot \nabla J(\theta_t)$$

2. **Ağırlık Güncellemesi:**

$$\theta_{t+1} = \theta_t - v_t$$

- v_t: Mevcut hız vektörü (momentum).
- v_{t-1}: Bir önceki hız vektörü.
- γ (Gama): **Momentum Oranı** (genellikle 0.9 olarak ayarlanır). Önceki hızın ne kadarının korunacağını belirleyen atalet katsayısıdır.
- η: Öğrenme Oranı (Learning Rate).
- $\nabla J(\theta_t)$: Mevcut konumdaki (θ_t) gradyan.

```
x = start_x
v = 0  # Hız vektörü (Momentum), başlangıçta sıfır
history = [x]

print(f"Momentumlu GD Başlatıldı (lr: {lr}, gamma: {gamma})")

for i in range(max_iter):
    grad = gradient(x)

    # 1. Hız Güncellemesi (Momentum)
    # Yeni hız, önceki hızın (v) ve mevcut gradyanın (grad) birleşimidir.
    v = gamma * v + lr * grad

    # 2. Konum/Parametre Güncellemesi
    # Konum, hesaplanan yeni hıza göre güncellenir.
    x = x - v
```

Gradyan Azalma (Gradient Descent)

AdaGrad (Adaptive Gradient), RMSProp (Root Mean Square Prop.), Adam

1. Standart Gradient Descent (GD)

Önce klasik halini yazalım.

Bir kayıp fonksiyonun olsun:

$$L(\theta)$$

Burada θ parametre vektörün. Standart gradient descent güncellemesi:

1. Gradyan hesapla:

$$g_t = \nabla_\theta L(\theta_t)$$

2. Parametreyi güncelle:

$$\theta_{t+1} = \theta_t - \eta\, g_t$$

- η = öğrenme oranı (learning rate), **sabit**.
- Her parametre bileşeni için aynı η kullanılıyor.
- Problem: Eğer bazı parametrelerin gradyanı çok büyük, bazılarınki çok küçükse, **zigzag** ve yavaş yakınsama görebiliyoruz.

Burada her parametre için "ortamı" aynı kabul ediyoruz: aynı hız, aynı adım.

Gradyan Azalma (Gradient Descent)

AdaGrad (Adaptive Gradient), RMSProp (Root Mean Square Prop.), Adam

1. Standart Gradient Descent (GD)

Önce klasik halini yazalım.

Bir kayıp fonksiyonun olsun:

$$L(\theta)$$

Burada θ parametre vektörün. Standart gradient descent güncellemesi:

1. Gradyan hesapla:

$$g_t = \nabla_\theta L(\theta_t)$$

2. Parametreyi güncelle:

$$\theta_{t+1} = \theta_t - \eta\, g_t$$

- η = öğrenme oranı (learning rate), **sabit**.
- Her parametre bileşeni için aynı η kullanılıyor.
- Problem: Eğer bazı parametrelerin gradyanı çok büyük, bazılarınki çok küçükse, **zigzag** ve yavaş yakınsama görebiliyoruz.

Burada her parametre için "ortamı" aynı kabul ediyoruz: aynı hız, aynı adım.

2. Fikir: Her parametre için farklı efektif öğrenme oranı olsun

Gözlem:

- Gradyanı **sürekli büyük** olan parametreler için biraz daha küçük adımlar atmak iyi olabilir.
- Gradyanı **sürekli küçük** olan parametreler için ise biraz daha büyük efektif adımlar fena olmaz.

Yani her parametre bileşeni için **adaptif bir ölçek** istiyoruz.

Bunu yapmak için, her parametrenin **geçmiş gradyanlarının büyüklüğüne** bakabiliriz.

Gradyan Azalma (Gradient Descent)

AdaGrad (Adaptive Gradient), RMSProp (Root Mean Square Prop.), Adam

3. Gradyanların karelerini biriktirme fikri (Adagrad yönüne ilk adım)

Diyelim her iterasyonda gradyanın karesini topluyoruz:

$$s_t = s_{t-1} + g_t^2$$

Burada:

- s_t parametrelerle aynı boyutta bir vektör.
- g_t^2 ifadesi **eleman bazlı kare** demek (Hadamard): her parametre için $g_t[i]^2$.

Sonra güncellemeyi şöyle yapabiliriz:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{s_t} + \epsilon} \odot g_t$$

- Burada "$\odot$" eleman bazlı çarpım.
- $\sqrt{s_t} + \epsilon$ her parametre için farklı bir ölçek veriyor.
- **Büyük gradyanlara sahip parametrelerde** s_t büyüyor → payda büyüyor → efektif öğrenme oranı küçülüyor.
- **Küçük gradyanlı parametrelerde** s_t küçük kalıyor → payda küçük → efektif öğrenme oranı büyüyor.

Bu fikir aslında **Adagrad**'ın mantığı. Ama önemli bir problem var:

👉 s_t her iterasyonda sürekli büyür, hiç geri düşmez.

Uzun eğitimlerde **öğrenme oranı giderek sıfıra yaklaşır**, eğitim "donar".

Bu problemi düzeltmek RMSProp'a giden yolun ana motivasyonu.

$g_t = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$ $g_t^2 \begin{bmatrix} 1 \\ 4 \\ 9 \end{bmatrix}$

$S_t = \begin{bmatrix} 3 \\ 7 \\ 16 \end{bmatrix}$ $S_{t-1} = \begin{bmatrix} 2 \\ 3 \\ 7 \end{bmatrix}$

$\frac{0,1}{\sqrt{3,7,16}} \odot \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$

$\frac{0.1}{\sqrt{3}} \cdot 1$

$\frac{0.1 \cdot 7}{\sqrt{7}}$

$\frac{0.1 \cdot 16}{\sqrt{16}}$

Neden kare ve birikimli toplama?

Gradyan Azalma (Gradient Descent)

AdaGrad (Adaptive Gradient), RMSProp (Root Mean Square Prop.), Adam

Neden kare alınır?
→ türevlenebilir olması için

$g_t = \begin{bmatrix}1\\2\\3\end{bmatrix}$ $g_t^2 = \begin{bmatrix}1\\4\\9\end{bmatrix}$ $\theta_{t+1} = \theta - 0.1 \cdot \begin{bmatrix}1\\2\\3\end{bmatrix}$

$s_{t-1} = \begin{bmatrix}7\\3\\8\end{bmatrix}$ $s_t = \begin{bmatrix}7\\3\\8\end{bmatrix} + \begin{bmatrix}1\\4\\9\end{bmatrix} = \begin{bmatrix}8\\7\\17\end{bmatrix}$ $\theta = \begin{bmatrix}\frac{0,1}{\sqrt{8}} \cdot 1\\ \frac{0.1}{\sqrt{7}} \cdot 2\\ \frac{0.1}{\sqrt{17}} \cdot 3\end{bmatrix}$

1. 🤓 **Matematiksel Düzgünlük ve Türevlenebilirlik (Diferansiyellenebilirlik)**
   - **Kare Fonksiyonu ($f(x) = x^2$):** Kare alma işlemi düzgün bir fonksiyondur, yani her noktada, özellikle $x = 0$'da, **türevlenebilir** bir fonksiyondur.
   - **Mutlak Değer Fonksiyonu ($f(x) = |x|$):** Mutlak değer fonksiyonu ise $x = 0$'da **türevlenebilir değildir** (bir "köşe" noktası vardır). Optimizasyon algoritmaları, model parametrelerinin güncellenmesi için genellikle arka arkaya türev hesaplamalarına dayanır. Mutlak değer fonksiyonu, bu algoritmalarda matematiksel sorunlara veya düzgün olmayan davranışlara yol açabilir.

2. ⚡ **Büyük Gradyanları Vurgulama**
   - Kare alma (x^2), büyük değerleri daha da büyütürken, küçük değerleri daha da küçültür (örneğin, $10^2 = 100$ iken, $0.1^2 = 0.01$).
   - Uyarlanabilir öğrenme oranı algoritmalarında, amaç, bir parametrenin gradyanları tutarlı bir şekilde büyük olduğunda, o parametre için öğrenme oranını **daha fazla azaltmaktır**. Gradyanların karesinin alınması, büyük gradyanların toplam/ortalama üzerindeki etkisini **büyük ölçüde artırır**.

1. 🏃 **Yönün Tutarlılığı ve Hızlanma**
   - **Gradyan Gürültüsünü Azaltma:** Optimizasyon sırasında hesaplanan anlık gradyanlar, özellikle mini-toplu (mini-batch) eğitimde, **gürültülü** olabilir ve gerçek yönü tam olarak temsil etmeyebilir. Geçmiş gradyanların birikimi veya hareketli ortalaması, bu gürültüyü yumuşatarak daha kararlı bir yön sağlar.
   - **Tutarlı İlerlemeyi Ödüllendirme:** Eğer bir parametrenin gradyanları sürekli olarak aynı yönde (veya aynı büyüklükte) ise, bu, o yönde güvenle hareket edilebileceği anlamına gelir. Geçmiş gradyanlara bakmak, algoritmaların tutarlı ilerleme kaydeden parametreler için öğrenme oranını daha az azaltmasını veya diğer algoritmalarda (Momentum, Adam) **hızlanmasını** sağlar.

Gradyan Azalma (Gradient Descent)

AdaGrad (Adaptive Gradient), RMSProp (Root Mean Square Prop.), Adam

3. Gradyanların karelerini biriktirme fikri (Adagrad yönüne ilk adım)

Diyelim her iterasyonda gradyanın karesini topluyoruz:

$$s_t = s_{t-1} + g_t^2$$

Burada:

- s_t parametrelerle aynı boyutta bir vektör.
- g_t^2 ifadesi **eleman bazlı kare** demek (Hadamard): her parametre için $g_t[i]^2$.

Sonra güncellemeyi şöyle yapabiliriz:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{s_t + \epsilon}} \odot g_t$$

- Burada "$\odot$" eleman bazlı çarpım.
- $\sqrt{s_t} + \epsilon$ her parametre için farklı bir ölçek veriyor.
- **Büyük gradyanlara sahip parametrelerde** s_t büyüyor → payda büyüyor → efektif öğrenme oranı küçülüyor.
- **Küçük gradyanlı parametrelerde** s_t küçük kalıyor → payda küçük → efektif öğrenme oranı büyüyor.

Bu fikir aslında **Adagrad**'ın mantığı. Ama önemli bir problem var:

👉 s_t her iterasyonda sürekli büyür, hiç geri düşmez.
Uzun eğitimlerde **öğrenme oranı giderek sıfıra yaklaşır**, eğitim "donar".

Bu problemi düzeltmek RMSProp'a giden yolun ana motivasyonu.

Hesaplama	θ_1 için	θ_2 için
Gradyan g_1	$g_{1,1} = 2.0$	$g_{2,1} = 0.1$
Gradyan Karesi g_1^2	$g_{1,1}^2 = 4.0$	$g_{2,1}^2 = 0.01$
V_1 **(Birikimli Toplam)**	$V_{1,1} = V_{1,0} + g_{1,1}^2 = 0 + 4.0 = 4.0$	$V_{2,1} = V_{2,0} + g_{2,1}^2 = 0 + 0.01 = 0.01$
Adaptif Öğr. Oranı	$\frac{\eta}{\sqrt{V_{1,1}}} = \frac{0.5}{\sqrt{4.0}} = \frac{0.5}{2.0} = \mathbf{0.25}$	$\frac{\eta}{\sqrt{V_{2,1}}} = \frac{0.5}{\sqrt{0.01}} = \frac{0.5}{0.1} = \mathbf{5.0}$
Güncelleme Miktarı	$\text{Rate} \times g_{1,1} = 0.25 \times 2.0 = 0.5$	$\text{Rate} \times g_{2,1} = 5.0 \times 0.1 = 0.5$
Yeni Parametre θ_2	$\theta_{1,2} = 0 - 0.5 = -\mathbf{0.5}$	$\theta_{2,2} = 0 - 0.5 = -\mathbf{0.5}$

Gradyan Azalma (Gradient Descent)

AdaGrad (Adaptive Gradient), RMSProp (Root Mean Square Prop.), Adam

4. Toplama yerine "hareketli ortalama" kullanma (RMSProp'un kalbi)

Sorun: $s_t = s_{t-1} + g_t^2$ çok büyüyor ve bir daha küçülmüyor.

Çözüm:
Her iterasyonda **eski bilgiyi biraz unut**, yeni gradyan bilgisini daha fazla önemse. Yani:

$$v_t = \beta\, v_{t-1} + (1-\beta)\, g_t^2$$

Bu bir **üstel hareketli ortalama** (EMA):

- $\beta \in [0,1)$, genelde 0.9 civarı.
- β büyük olursa: geçmiş gradyanlar daha fazla hatırlanır.
- $(1-\beta)$ yeni gradyanın ne kadar ağırlıkla dikkate alınacağını belirler.
- Artık v_t "son zamanlardaki ortalama kare gradyan büyüklüğü" gibi davranır, sonsuza gitmez, dengeye oturur.

Bu noktada hala güncellemeyi tanımlamadık, sadece "ölçek" için bir istatistik tuttuk.

5. RMSProp güncellemesi: Gradyanı bu ortalama ile ölçekle

Şimdi standart GD formülünü, bu yeni v_t ile modifiye ediyoruz:

Standart GD:

$$\theta_{t+1} = \theta_t - \eta\, g_t$$

RMSProp:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{v_t} + \epsilon} \odot g_t$$

Burada:

- $v_t = \beta v_{t-1} + (1-\beta) g_t^2$
- $\sqrt{v_t} + \epsilon$ → her parametre için **"son zamanlardaki ortalama gradyan büyüklüğünün karekökü"**.
- ϵ (ör: 10^{-8}) → sıfıra bölmeyi önlemek için küçük sabit.

Yorum:

- Eğer bir parametre için son zamanlarda gradyanlar çok büyükse:
  v_t büyük → $\sqrt{v_t}$ büyük → efektif öğrenme oranı küçük → daha küçük adımlar.
- Eğer bir parametre için gradyanlar genelde küçükse:
  v_t küçük → $\sqrt{v_t}$ küçük → efektif öğrenme oranı büyük → daha cesur adımlar.

Böylece:

- Öğrenme oranı **adaptif** hale geldi (parametreye ve zamana göre değişiyor).
- Ama **sürekli azalmıyor**, çünkü v_t üstel ağırlıklı ortalama → sabit bir seviyede dalgalanıyor.

Gradyan Azalma (Gradient Descent)

AdaGrad (Adaptive Gradient), RMSProp (Root Mean Square Prop.), Adam

7. Mini pseudo-kod ile farkı netleştirelim

Standart Gradient Descent

python Kodu kopyala

```python
theta = theta_init
for t in range(T):
    g = grad_L(theta)           # ∇L(θ_t)
    theta = theta - eta * g     # sabit adım
```

RMSProp

python Kodu kopyala

```python
theta = theta_init
v = 0
beta = 0.9
eps = 1e-8

for t in range(T):
    g = grad_L(theta)                      # ∇L(θ_t)
    v = beta * v + (1 - beta) * (g * g)    # hareketli ortalama (eleman bazlı kare)
    theta = theta - eta * g / (np.sqrt(v) + eps)
```

RMSProp algoritması, standart bir bilimsel makale olarak yayımlanmasından ziyade, ilk olarak **Geoffrey Hinton**'ın **Coursera'daki bir derin öğrenme dersinde (Lecture 6e)** tanıtılmıştır.

Bu nedenle, RMSProp'un klasik anlamda tek, belirleyici bir "ilk makalesi" yoktur. Genellikle atıfta bulunulan kaynak, 2012 yılında yayımlanan bu ders notları veya slaytlarıdır:

- **Kaynak: Coursera Course: Neural Networks for Machine Learning, Lecture 6e.**
- **Yazar: Geoffrey Hinton** ve öğrencileri.
- **Yıl: 2012**.

Bu ders materyalleri, algoritmanın AdaGrad'ın öğrenme oranının çok hızlı azalması sorununa bir çözüm olarak nasıl çalıştığını ve gradyan karelerinin üstel hareketli ortalamasını kullandığını açıklamıştır.

Momentum Yok!

Gradyan Azalma (Gradient Descent)

AdaGrad (Adaptive Gradient), RMSProp (Root Mean Square Prop.), Adam

%85 Sınavda Çıkar

Adam algoritmasının amacı
Adaptif öğrenmeyi momentum ile
birleştirmektir.

2. Fikir: Momentum'u da ekleyelim (1. moment)

Momentum'da yaptığımız şey:

- Gradyanları doğrudan kullanmak yerine, onların **üstel hareketli ortalaması** ile güncelleme yapmak.

Momentum'da tipik olarak:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$

- m_t: 1. moment (ortalama gradyan).
- β_1: genelde ~0.9.

Klasik momentum güncellemesi:

$$\theta_{t+1} = \theta_t - \eta m_t$$

Yani:

- **RMSProp** sadece v_t (kare gradyan ortalaması) kullanıyor.
- **Momentum** sadece m_t (ortalama gradyan) kullanıyor.

Adam: İkisinin iyi yanlarını birleştiriyor.

Gradyan Azalma (Gradient Descent)

AdaGrad (Adaptive Gradient), RMSProp (Root Mean Square Prop.), Adam

3. Adam'ın asıl fikri: Hem 1. moment hem 2. moment

Adam'da aynı anda:

- **1.** moment (ortalama gradyan):

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t$$

- **2.** moment (ortalama kare gradyan):

$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2$$

Dikkat:

- β_1 (örn. 0.9) → momentum benzeri yumuşatma.
- β_2 (örn. 0.999) → RMSProp tarzı kare gradyan takibi.

Sonra, "RMSProp'taki gibi 2. moment ile ölçekleyip, momentum'daki gibi 1. momentle yön verelim" diyoruz:

👉 kaba haliyle:

$$\theta_{t+1} \approx \theta_t - \frac{\eta}{\sqrt{v_t} + \epsilon} m_t$$

Yani:

- m_t gradyanın yönünü ve ortalamasını temsil ediyor (momentum).
- v_t gradyan büyüklüğüne göre adaptif step size veriyor (RMSProp).

Ama hâlâ bir eksik daha var.

Gradyan Azalma (Gradient Descent)

AdaGrad (Adaptive Gradient), RMSProp (Root Mean Square Prop.), Adam

3. Adam'ın asıl fikri: Hem 1. moment hem 2. moment

Adam'da aynı anda:

- **1.** moment (ortalama gradyan):

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t$$

- **2.** moment (ortalama kare gradyan):

$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2$$

Dikkat:

- β_1 (örn. 0.9) → momentum benzeri yumuşatma.
- β_2 (örn. 0.999) → RMSProp tarzı kare gradyan takibi.

Sonra, "RMSProp'taki gibi 2. moment ile ölçekleyip, momentum'daki gibi 1. momentle yön verelim" diyoruz:

👉 kaba haliyle:

$$\theta_{t+1} \approx \theta_t - \frac{\eta}{\sqrt{v_t} + \epsilon} m_t$$

Yani:

- m_t gradyanın yönünü ve ortalamasını temsil ediyor (momentum).
- v_t gradyan büyüklüğüne göre adaptif step size veriyor (RMSProp).

Ama hâlâ bir eksik daha var.

Diederik P. Kingma

Other names ▸

Anthropic

Verified email at anthropic.com - Homepage

Machine Learning Deep Learning Neural Networks Generative Models Artificial Intelligence

FOLLOW

TITLE	CITED BY	YEAR
Adam: A method for stochastic optimization DP Kingma, J Ba arXiv preprint arXiv:1412.6980	$\boxed{236041}$	2014

$y = Xw$ $\quad (X^T \cdot X)^{-1} \cdot X^T \cdot y$

$\min |y - Xw|^2$

$A \in R$

Yapay Sinir Ağları

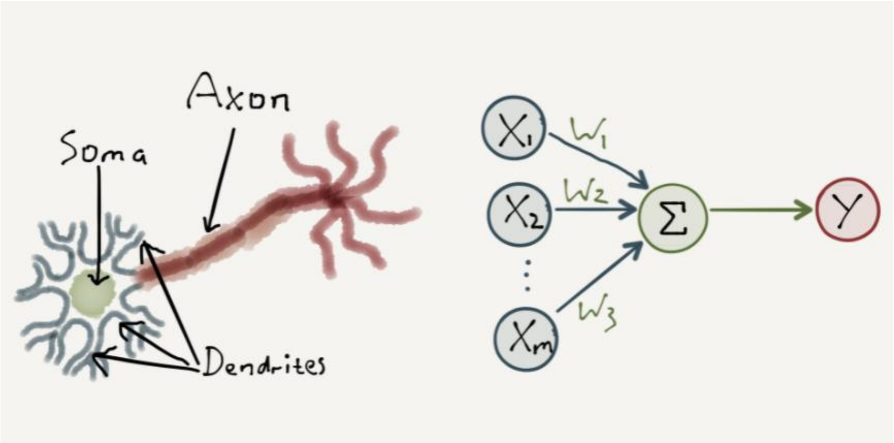
Bulletin of Mathematical Biology Vol. 52, No. 1/2, pp. 99–115, 1990.
Printed in Great Britain.

0092–8240/90$3.00 + 0.00
Pergamon Press plc
Society for Mathematical Biology

A LOGICAL CALCULUS OF THE IDEAS IMMANENT IN NERVOUS ACTIVITY*

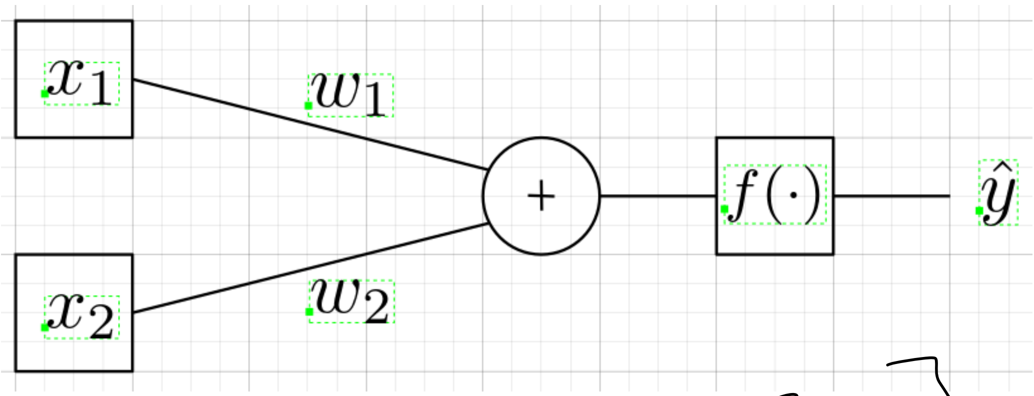
■ WARREN S. MCCULLOCH AND WALTER PITTS
University of Illinois, College of Medicine,
Department of Psychiatry at the Illinois Neuropsychiatric Institute,
University of Chicago, Chicago, U.S.A.

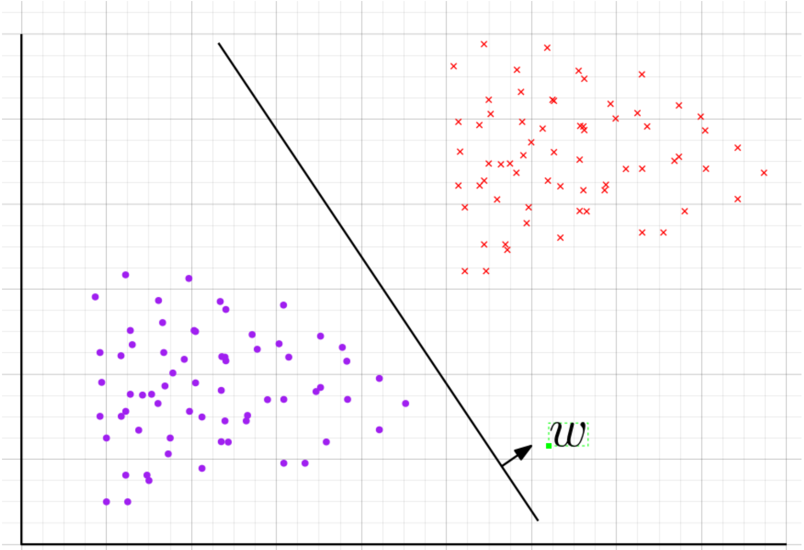
Because of the "all-or-none" character of nervous activity, neural events and the relations among them can be treated by means of propositional logic. It is found that the behavior of every net can be described in these terms, with the addition of more complicated logical means for nets containing circles; and that for any logical expression satisfying certain conditions, one can find a net behaving in the fashion it describes. It is shown that many particular choices among possible neurophysiological assumptions are equivalent, in the sense that for every net behaving under one assumption, there exists another net which behaves under the other and gives the same results, although perhaps not in the same time. Various applications of the calculus are discussed.



https://jontysinai.github.io/jekyll/update/2017/11/11/the-perceptron.html

Yapay Sinir Ağları

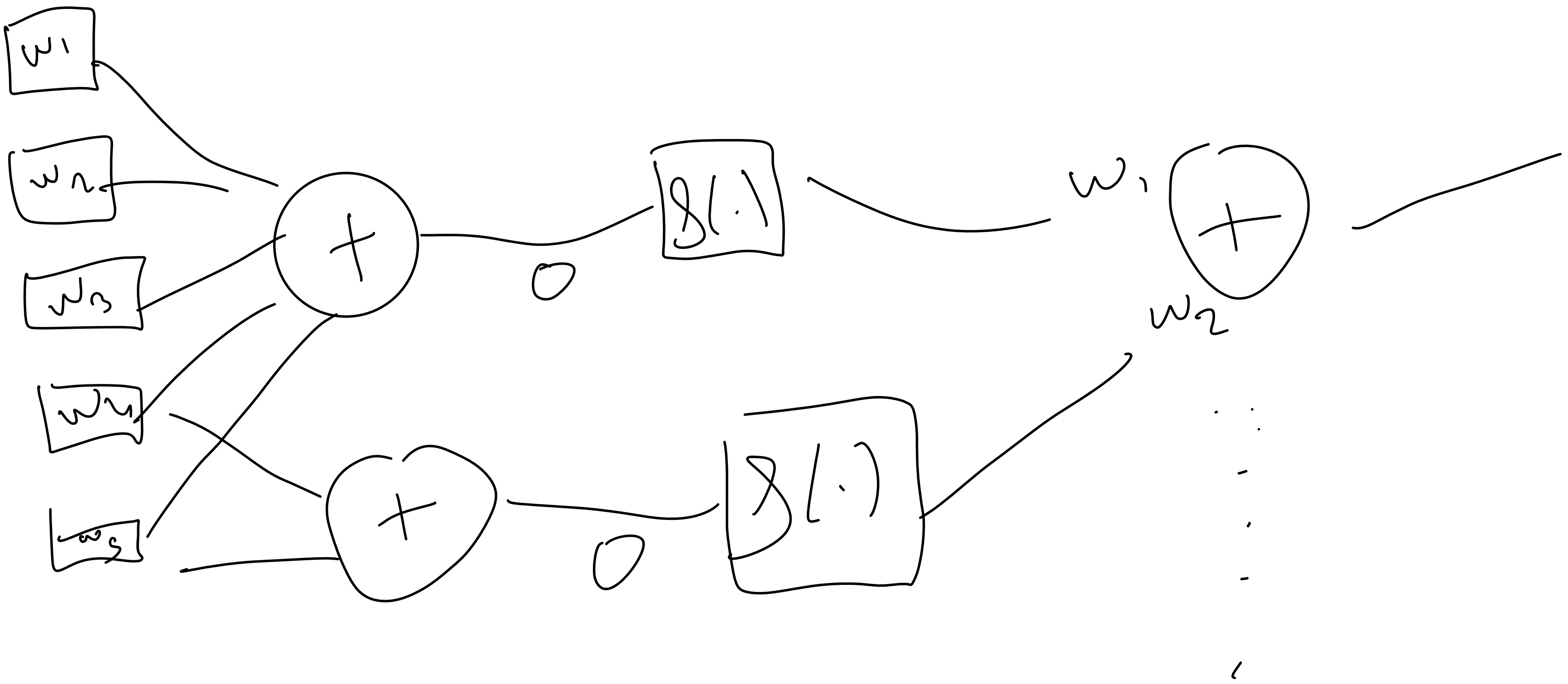




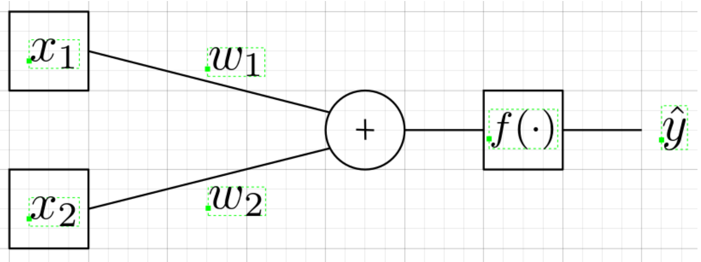
$$\bar{X} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \quad w = \begin{bmatrix} w_1 \\ w_2 \end{bmatrix}$$

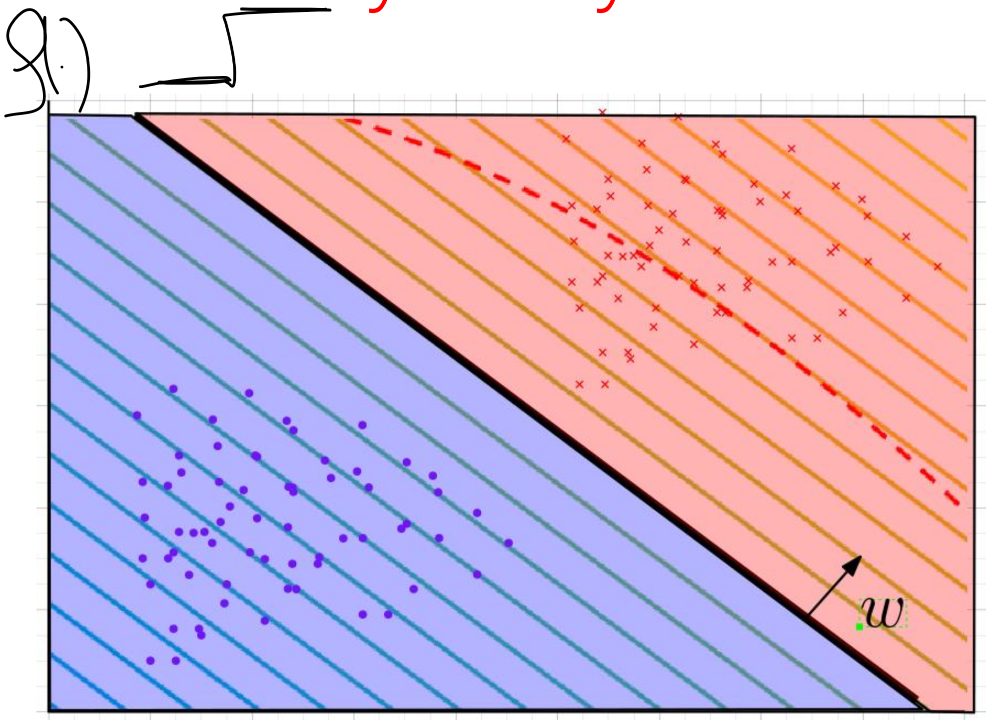
$$o = \langle \bar{x}, \bar{w} \rangle = x_1 w_1 + x_2 w_2$$

Yapay Sinir Ağları

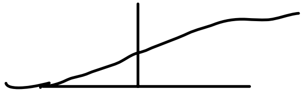


Yapay Sinir Ağları: Hatanın Geriye Yayılması





$$\hat{y}_i = f(\,\langle \boldsymbol{w}, \boldsymbol{x_i} \rangle\,)$$

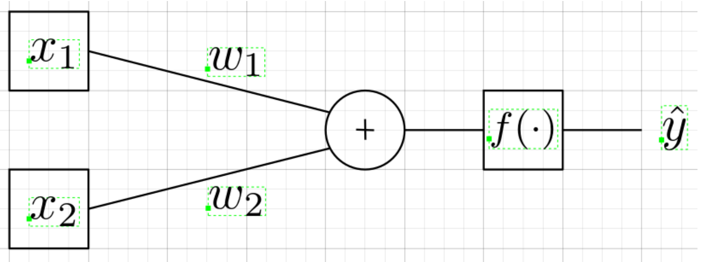


$$e_i = \frac{1}{2}(\hat{y}_i - y_i)^2 \Rightarrow \text{hata}$$

$$\frac{de_i}{d\boldsymbol{w}} = (\hat{y}_i - y_i)\frac{d\hat{y}_i}{d\boldsymbol{w}}$$

$$\frac{de_i}{dw} = \frac{de_i}{d\hat{y}_i} \cdot \frac{d\hat{y}_i}{do_i} \cdot \frac{do_i}{dw}$$

Yapay Sinir Ağları: Hatanın Geriye Yayılması

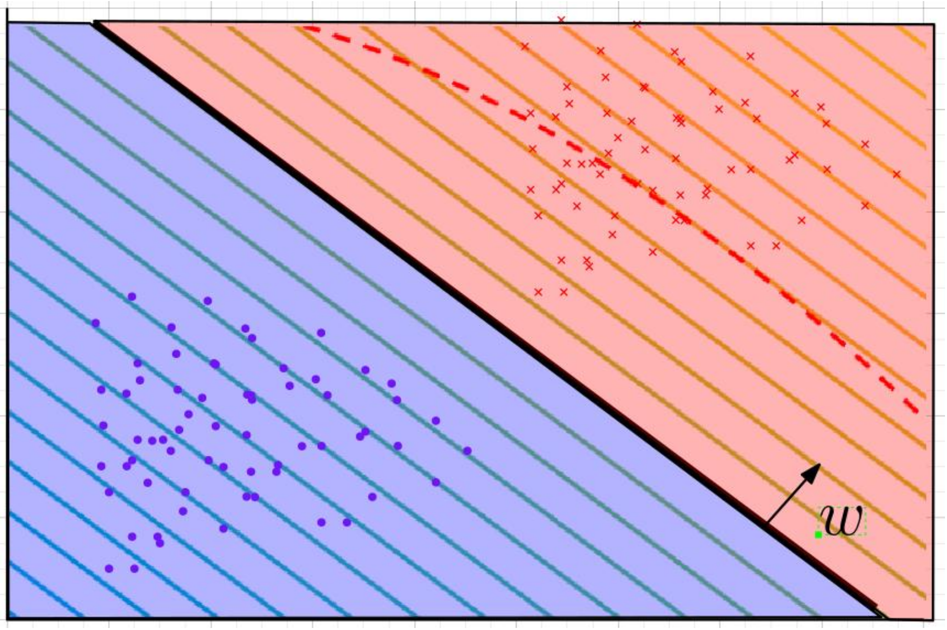


$$\hat{y}_i = f(\langle \boldsymbol{w}, \boldsymbol{x_i} \rangle)$$

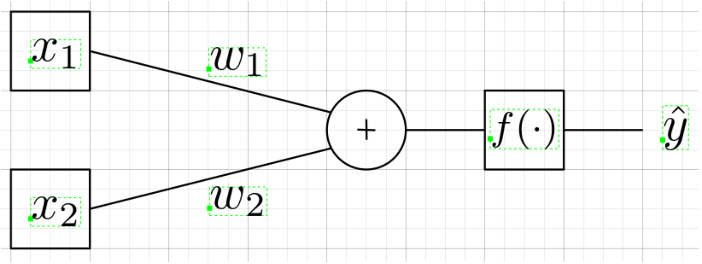
$$e_i = \frac{1}{2}(\hat{y}_i - y_i)^2$$

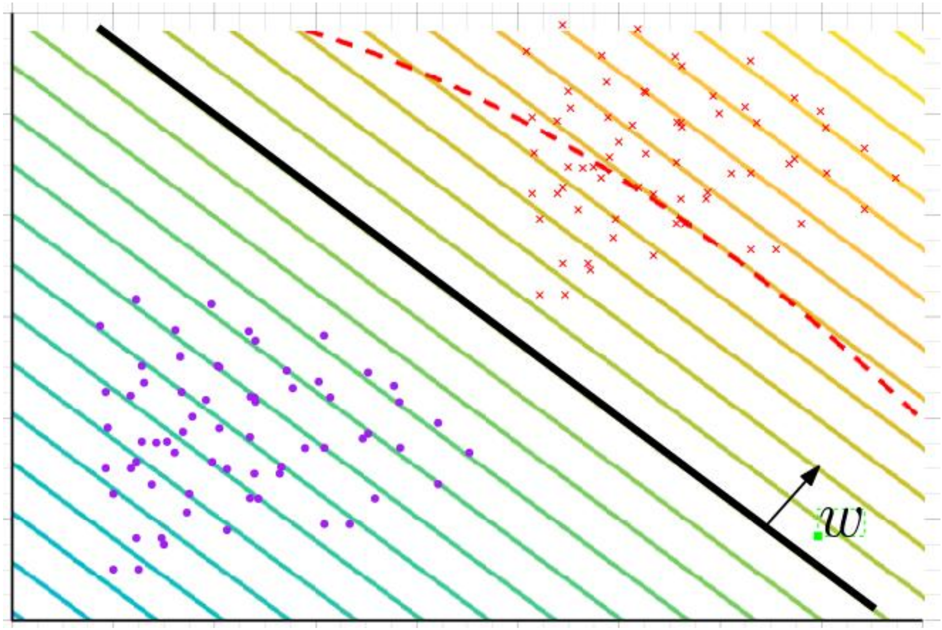
$$\frac{de_i}{d\boldsymbol{w}} = (\hat{y}_i - y_i)\frac{d\hat{y}_i}{d\boldsymbol{w}}$$

$$\frac{d\, f(x)}{dx}$$



Yapay Sinir Ağları: Hatanın Geriye Yayılması





$$\hat{y}_i = f(\langle \boldsymbol{w}, \boldsymbol{x_i} \rangle)$$

$$e_i = \frac{1}{2}(\hat{y}_i - y_i)^2$$

$$\frac{de_i}{d\boldsymbol{w}} = (\hat{y}_i - y_i)\frac{d\hat{y}_i}{d\boldsymbol{w}}$$

$$\frac{d\ f(x)}{dx}$$

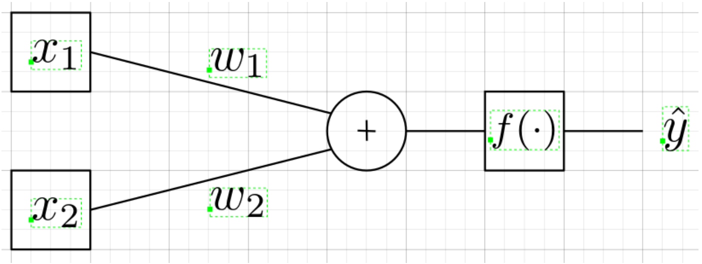
$$f(x) = \frac{1}{1 + e^{-x}}$$

$$\frac{de_i}{d\bar{w}} = \frac{de_i}{d\hat{y}_i} \cdot \frac{d\hat{y}_i}{do} \cdot \frac{do}{d\bar{w}}$$

$$\underbrace{(\hat{y}_i - y_i)}\ \underbrace{(\hat{y}_i \cdot (1-\hat{y}_i))} \cdot \underbrace{\bar{x}_i}$$

Katsayı

Vektör



$\mathcal{L}\{\ f(\boldsymbol{x}_i;\boldsymbol{w})\ \ ,y_i\ \}$ $\qquad y_i \in \{0,1\}$

Lojistik Regresyon

$$P\{y=1|\boldsymbol{x}\} = \frac{1}{1+e^{-\langle \boldsymbol{w},\boldsymbol{x}\rangle}}$$