Invatare automata in vedere artificiala

Curs 2: Clasificarea Imaginilor. Optimizare

Administrativ

- Proiect
  - 2 faze:
    - Mid term: Alegere dataset si related work (research SOTA) - Săptămâna 5
      - 26.03.2025
      - Redactare unei prezentari de reasearch
    - Experimentare si Prezentare rezultate: Săptămânile 9 & 10
  - Alegere proiect din lista de proiecte link
  - Proiect licentă Deep Learning
  - Alte propuneri de proiecte sunt binevenite

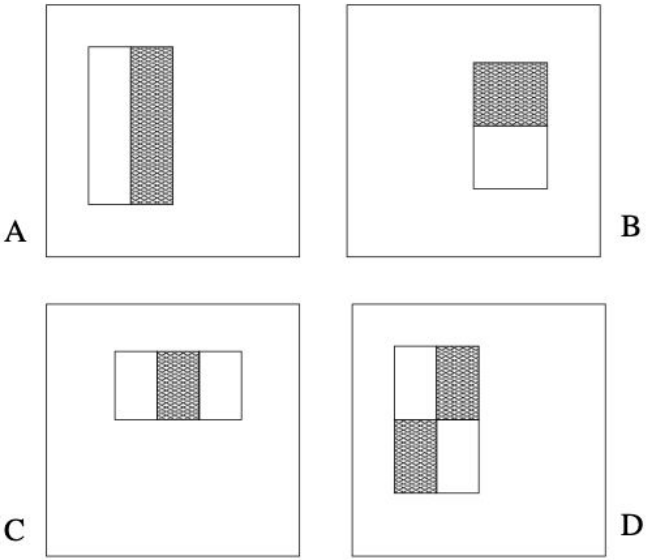
Descrierea problemei

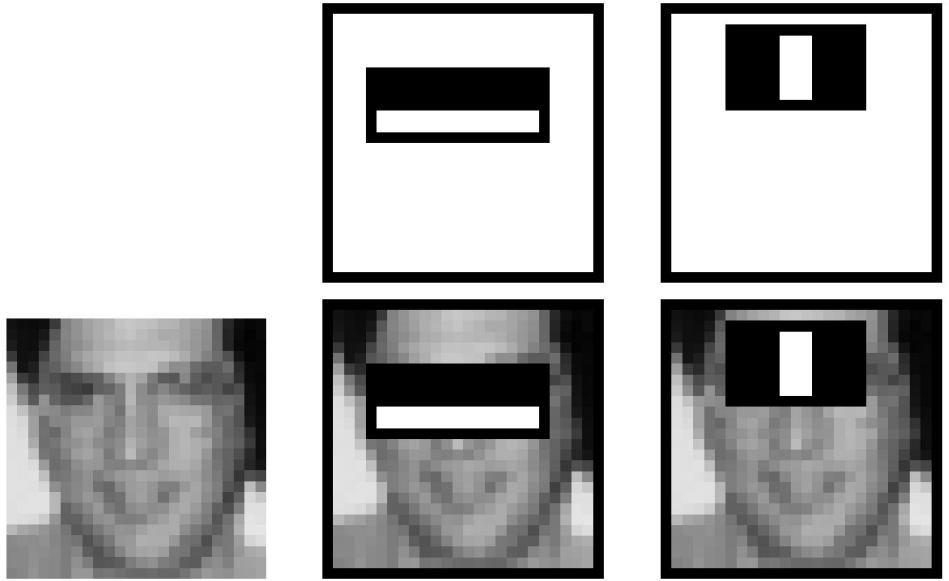
```
[[[ 78  75  66]
  [ 77  74  65]
  [ 72  69  62]
  ...,
  [255 255 255]
  [255 255 255]
  [255 255 255]]

 [[ 69  69  59]
  [ 69  69  59]
  [ 65  65  57]
  ...,
  [255 255 255]
  [255 255 255]
  [255 255 255]]

 [[ 68  68  58]
  [ 69  69  59]
  [ 65  65  57]
  ...,
  [255 255 255]
  [255 255 255]
  [255 255 255]]

 ...,
 [[114 106  93]
  [117 109  96]
  [119 111  98]
  ...,
  [134 139 133]
  [134 139 133]
  [134 139 133]]

 [[119 111  98]
  [121 113 100]
  [122 114 101]
  ...,
```

Probabilitate pisica

pisica	0.9

Caracteristici simple

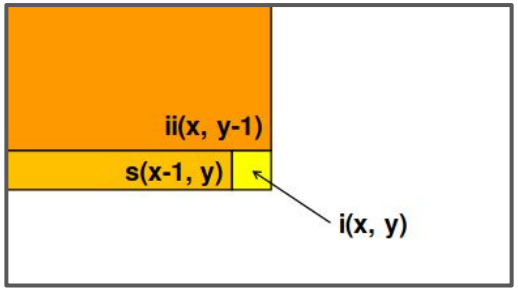




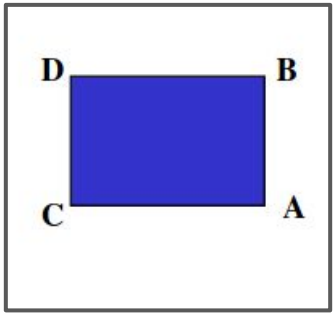
Viola, P., & Jones, M. (2001, December). Rapid object detection using a boosted cascade of simple features. In Proceedings of the 2001 IEEE computer society conference on computer vision and pattern recognition. CVPR 2001 (Vol. 1, pp. I-I). Ieee.

Calcularea eficienta a caracteristicilor

- Folosind programare dinamică, calculează sumele parțiale dintr-o matrice
  - Suma pe rand: s(x, y) = s(x, y-1) + i(x, y)
  - Suma pe imagine pornind din colțul din stânga dreapta: ii(x, y) = ii(x, y-1) + s(x-1, y)
  - Complexitate?



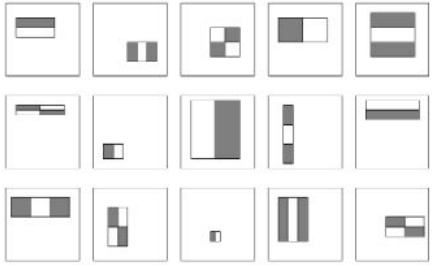
- În cazul în care vrem sa calculăm suma elementelor din oricare submatrice din cadrul unei imagini:
  - A, B, C, D - sumele partiale pe imagine
  - Suma întregului dreptunghi = A - B - C + D
  - Complexitate?



Viola Jones detector

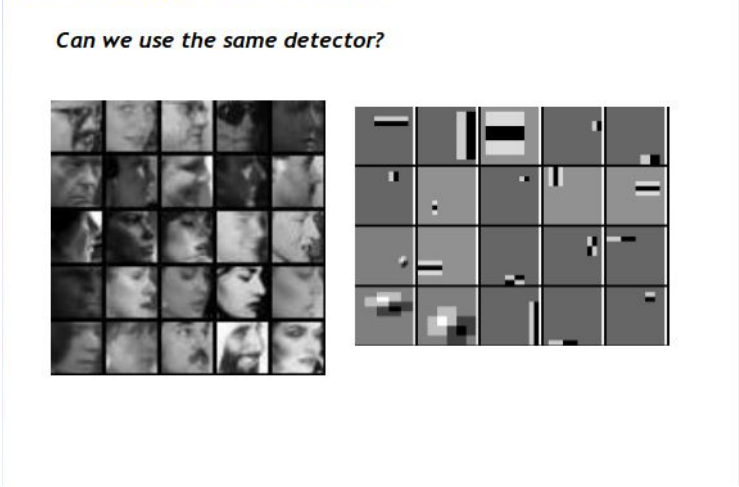
- Știind să calculăm eficient submatricile, trebuie ales un subset de filtre care determină cel mai bine dacă o fereastră conține o față
- Selectarea acestor filtre se realizează cu ajutorul unui algoritm de boosting (un ansamblu de modele 'weak' care oferă o performanță puternică)

- Putem folosi același detector când dorim să detectăm și 'profile faces'?



Considering all possible filter parameters: position, scale, and type:

180,000+ possible features associated with each 24 x 24 window



Dificultati

Deformare

Ocluzie

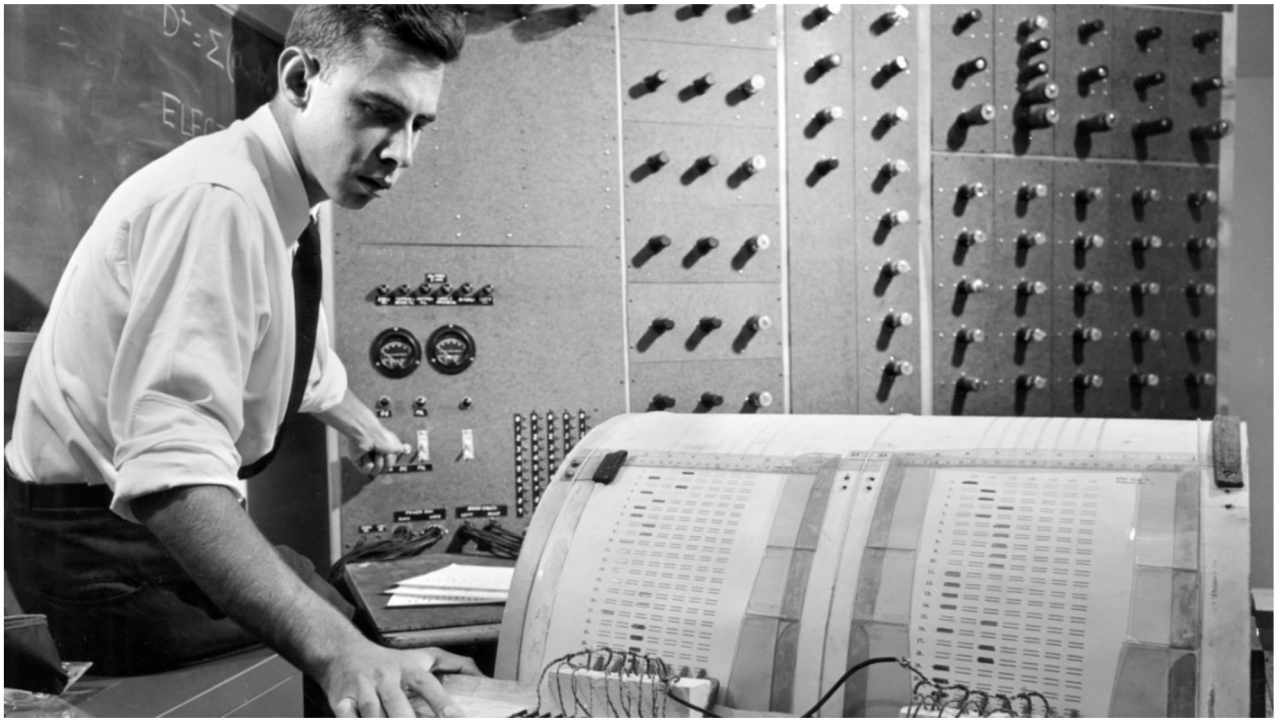
Iluminare

Camuflaj

Varietate

- Nu avem o soluție programatică intuitivă (`if magic then cat else dog`)

Perceptron

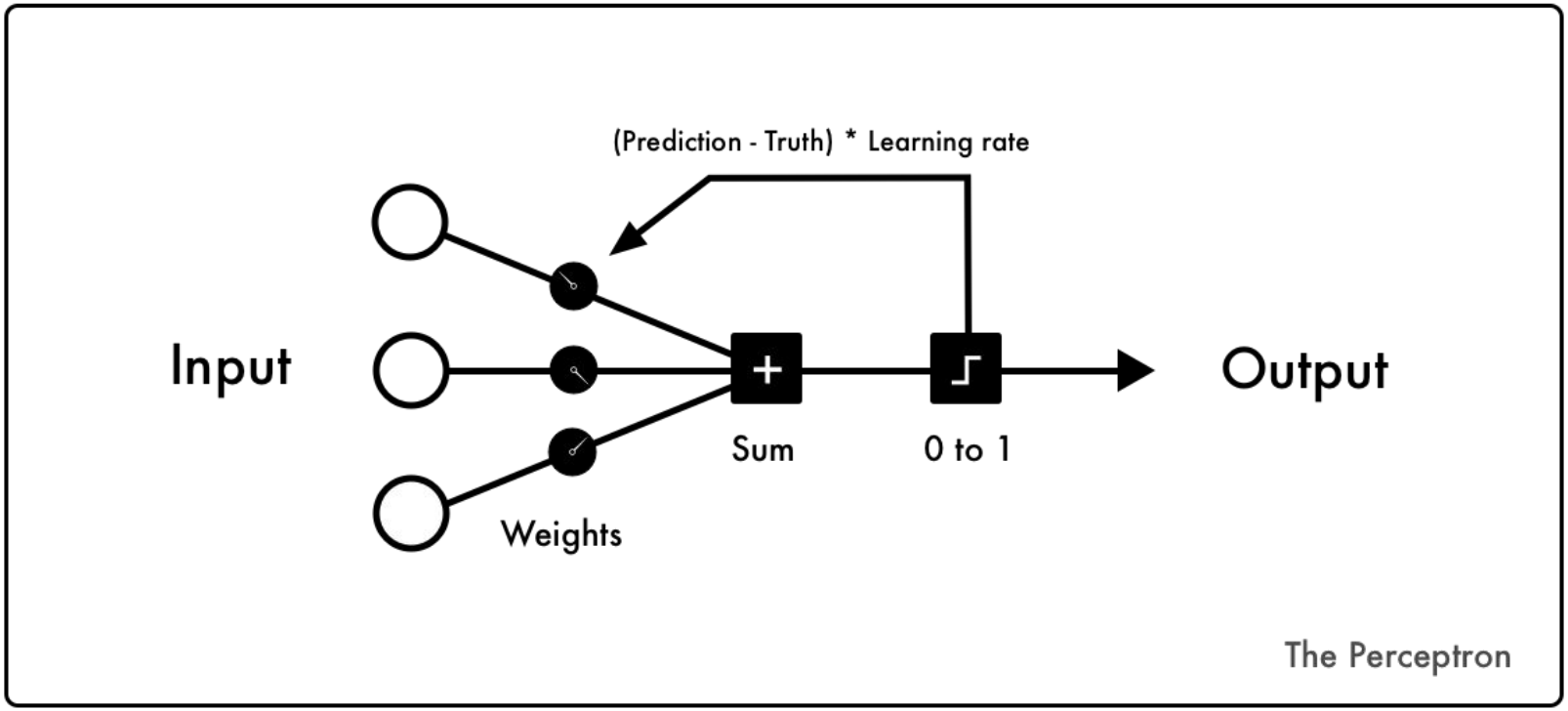


Division of Rare and Manuscript Collections

Frank Rosenblatt ’50, Ph.D. ’56, works on the “perceptron” – what he described as the first machine “capable of having an original idea.”

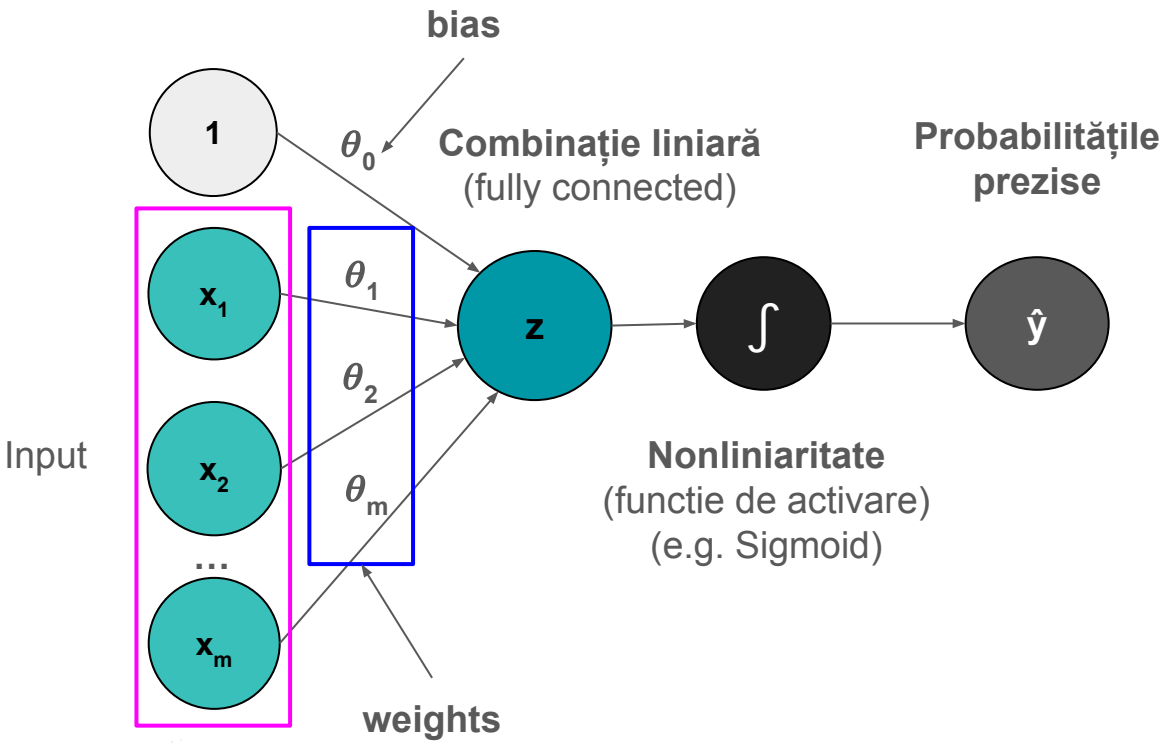
https://news.cornell.edu/stories/2019/09/professors-perceptron-paved-way-ai-60-years-too-soon

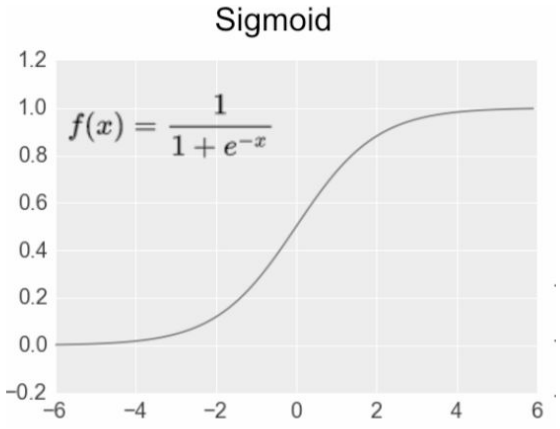
Perceptron



https://emilwallner.medium.com/the-history-of-deep-learning-explored-through-6-code-snippets-d0a0e8545202

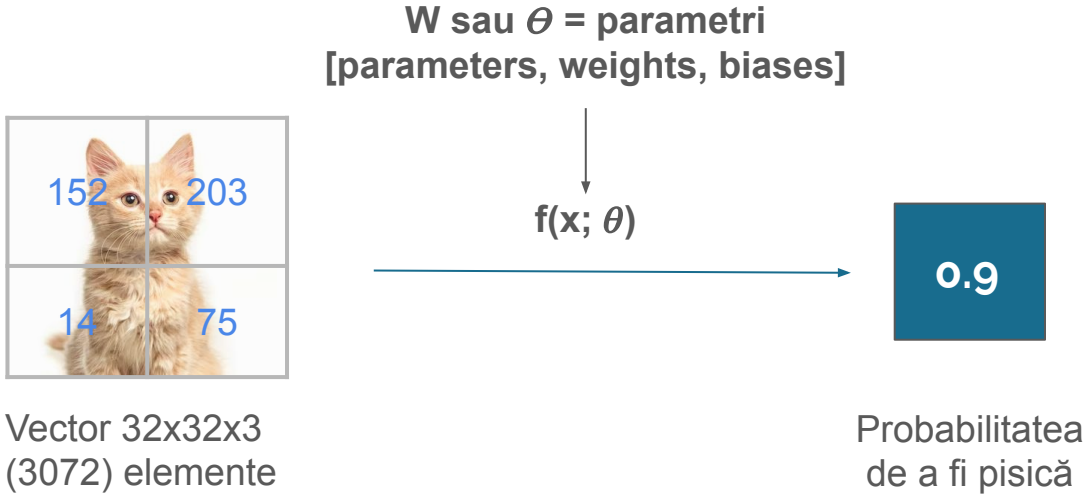
Perceptron



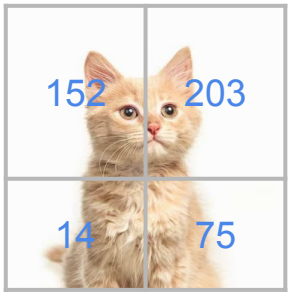


$$z = \theta_0 + \sum_{i=1}^{m} x_i \theta_i$$

Clasificare binară



Clasificare Multiclass

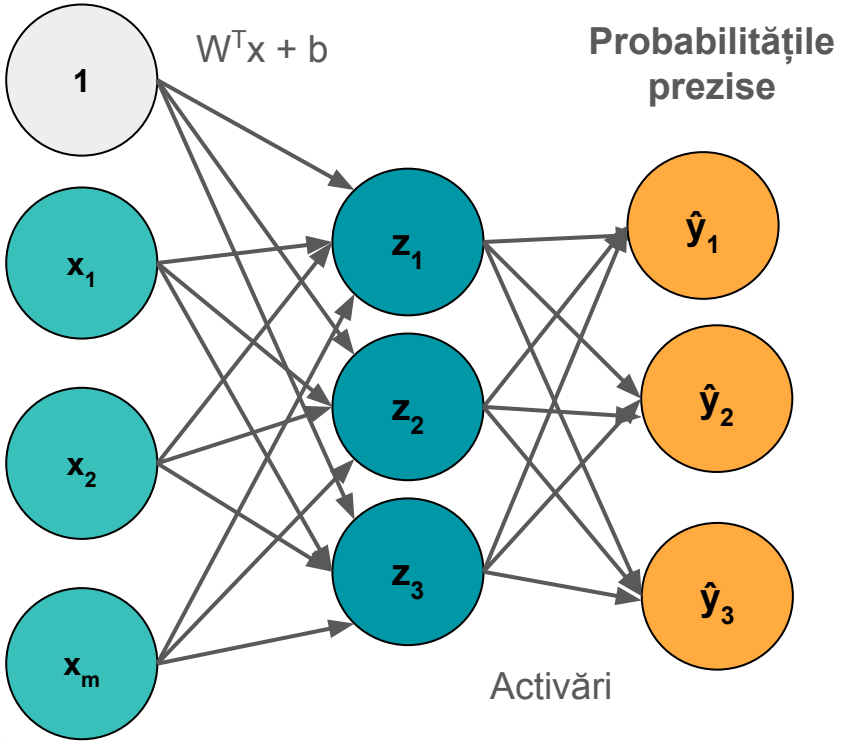


Vector 32x32x3
(3072) elemente

$f(x; \theta)$

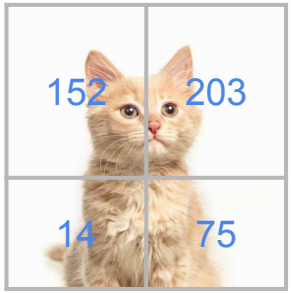
airplane	0.08
automobile	0.07
bird	0.04
cat	0.3
deer	0.06
dog	0.1
frog	0.09
horse	0.2
ship	0.04
truck	0.02

Perceptron multi-ieșire



$$z_j = \theta_{0,j} + \sum_{i=1}^{m} x_i \theta_{i,j}$$

Clasificare liniară



$$f(x; w, b) = W * X + b$$

clasa	scor	probabilitate
caine	-54.7	6.05956038e-78
pisica	123.1	1.00000000e+00
soarece	31.05	1.05485543e-40

W					X		b		Scor
0.2	-0.5	0.1	0.2	*	152	+	0.01	=	-54.7
0.4	0.3	0.1	0.0		203		0.03		123.1
0.0	0.25	0.2	-0.3		14		0.09		31.05
					75		0.02		

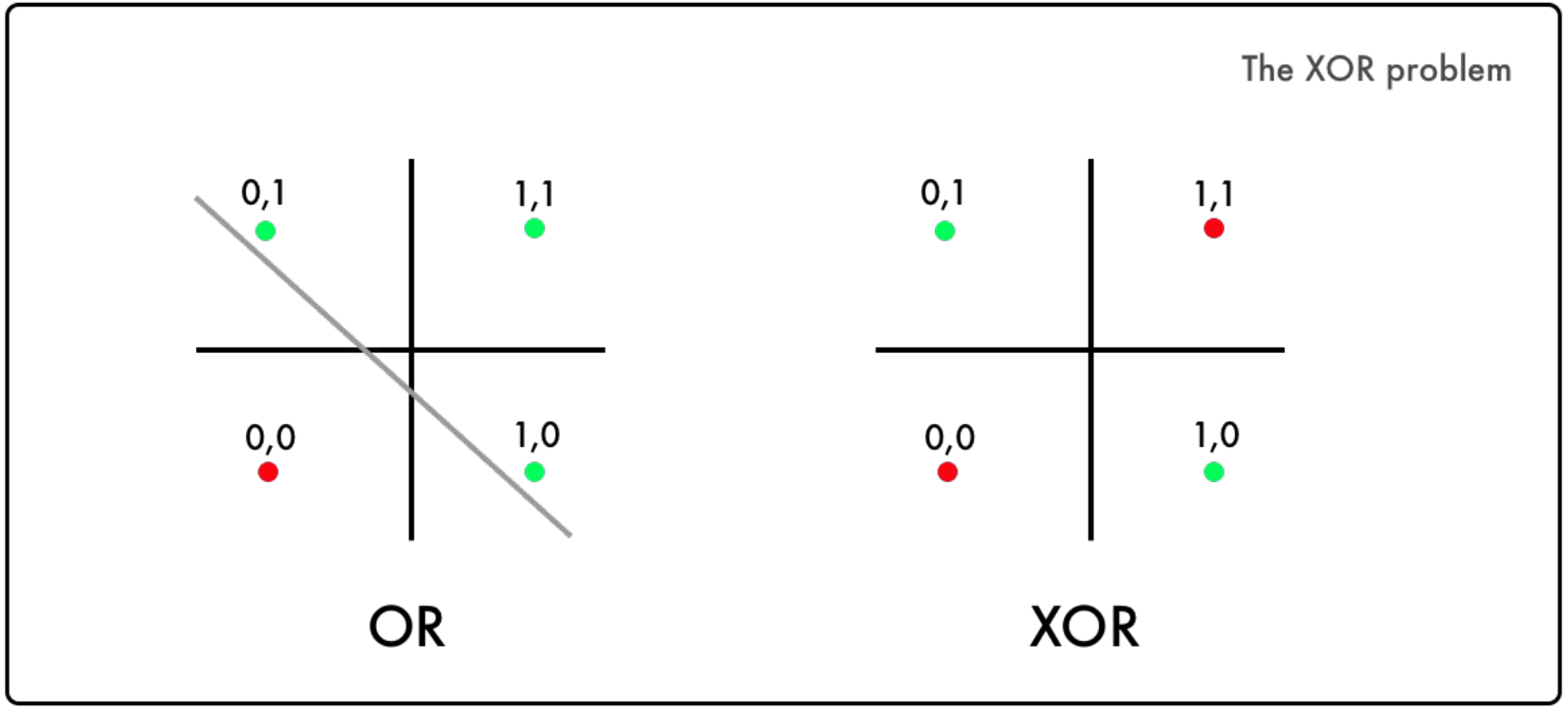
Multinomial Logistic regression

- **Scorurile (z) = valorile retelei inainte de softmax (logits)**
- Generalizare de la scoruri cu doua clase - functia sigmoid (cat-not_cat)
  - Softmax $P(Y = k|X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}}$
  - s_i = scorul aferent clasei *i*
- Sumeaza la 1 - probabilități

```python
import numpy as np

scores = [3.0, 1.0, 0.2]

def softmax(x):
    """Compute softmax values for each sets of scores in x."""
    return np.exp(x) / np.sum(np.exp(x), axis=0)

print(softmax(scores))
```

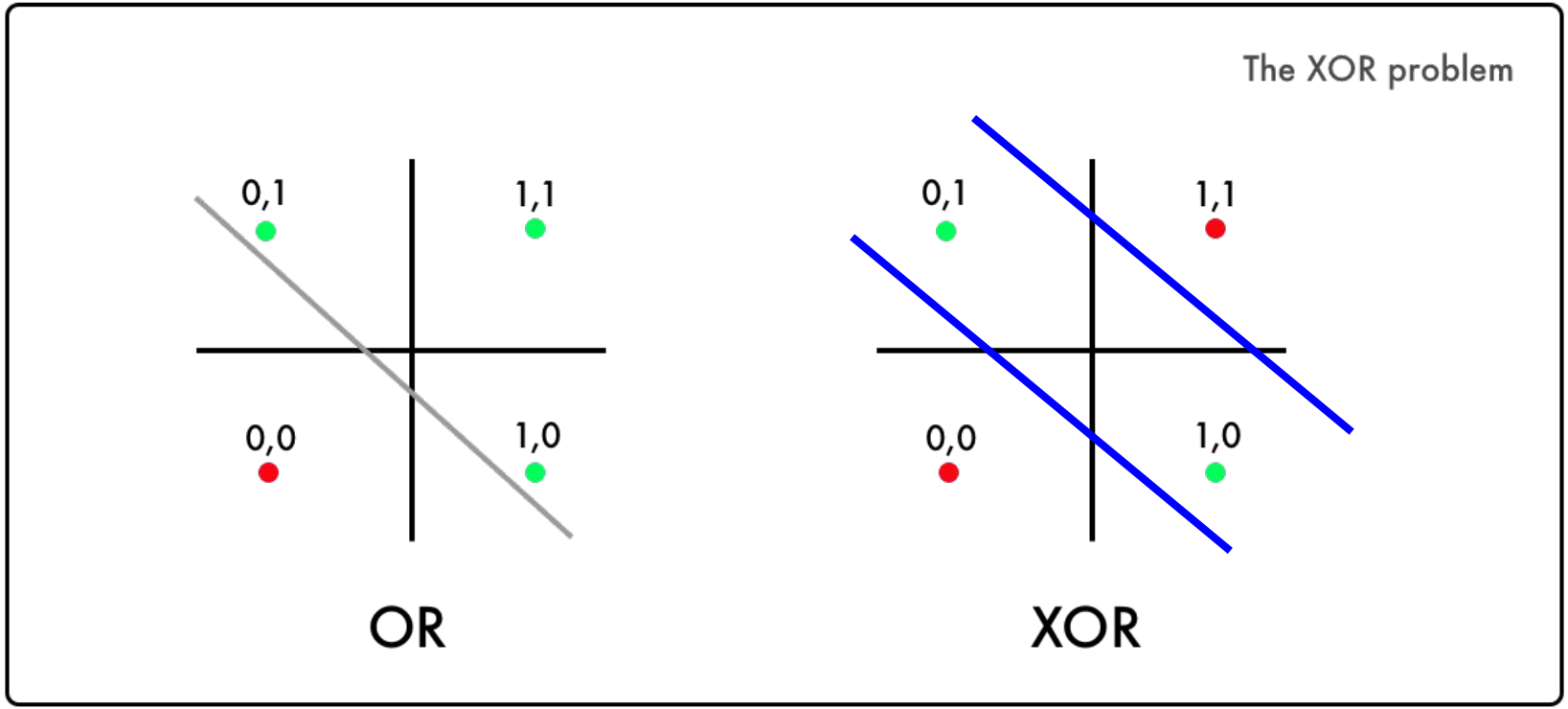
```
[ 0.8360188   0.11314284  0.05083836]
```

Limitările Perceptron-ului



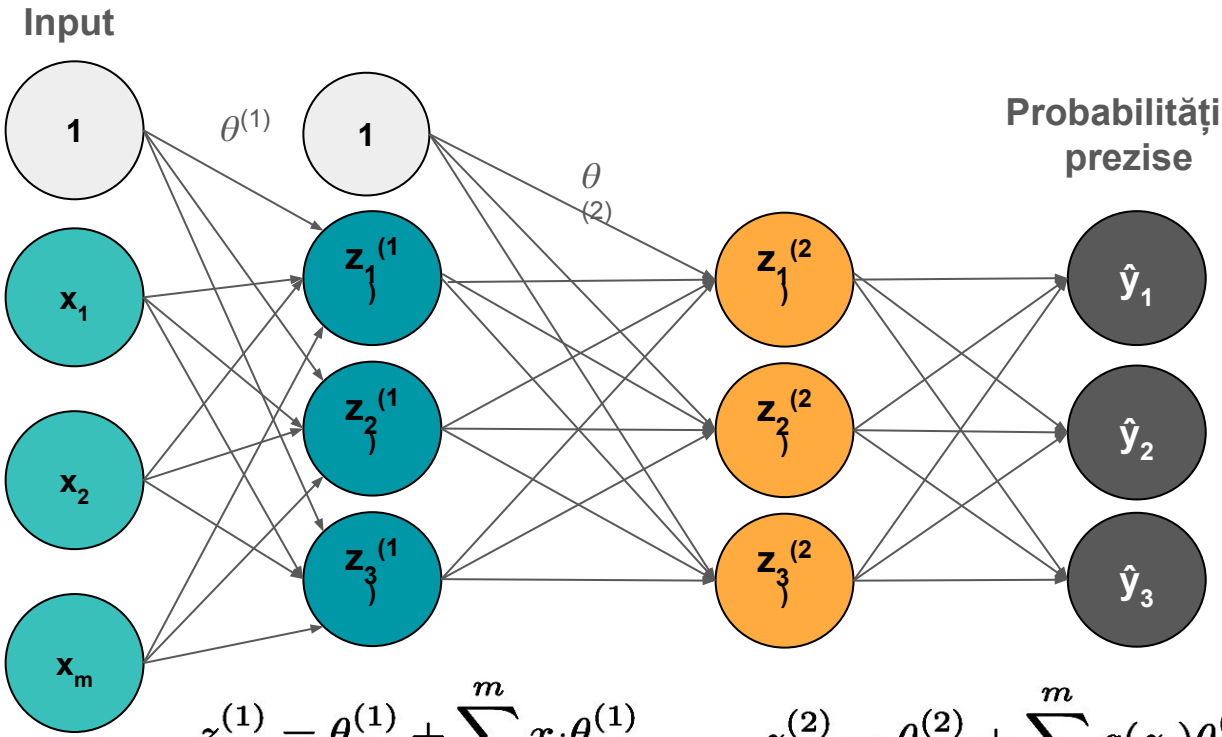
https://emilwallner.medium.com/the-history-of-deep-learning-explored-through-6-code-snippets-d0a0e8545202

Limitările Perceptron-ului



https://emilwallner.medium.com/the-history-of-deep-learning-explored-through-6-code-snippets-d0a0e8545202

Rețea cu mai multe straturi

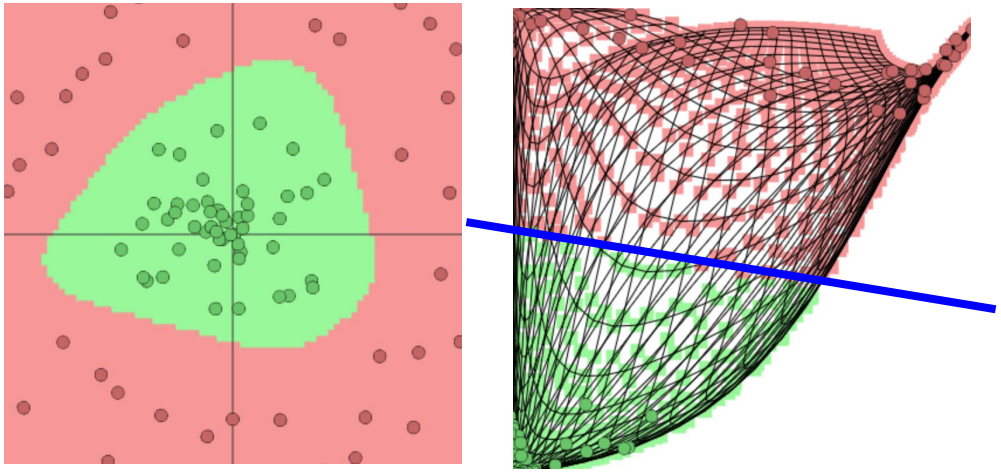


$$z_j^{(1)} = \theta_{0,j}^{(1)} + \sum_{i=1}^{m} x_i \theta_{i,j}^{(1)}$$

$$z_j^{(2)} = \theta_{0,j}^{(2)} + \sum_{i=1}^{m} g(z_i) \theta_{i,j}^{(2)}$$

g = funcție de activare (nonliniaritate)

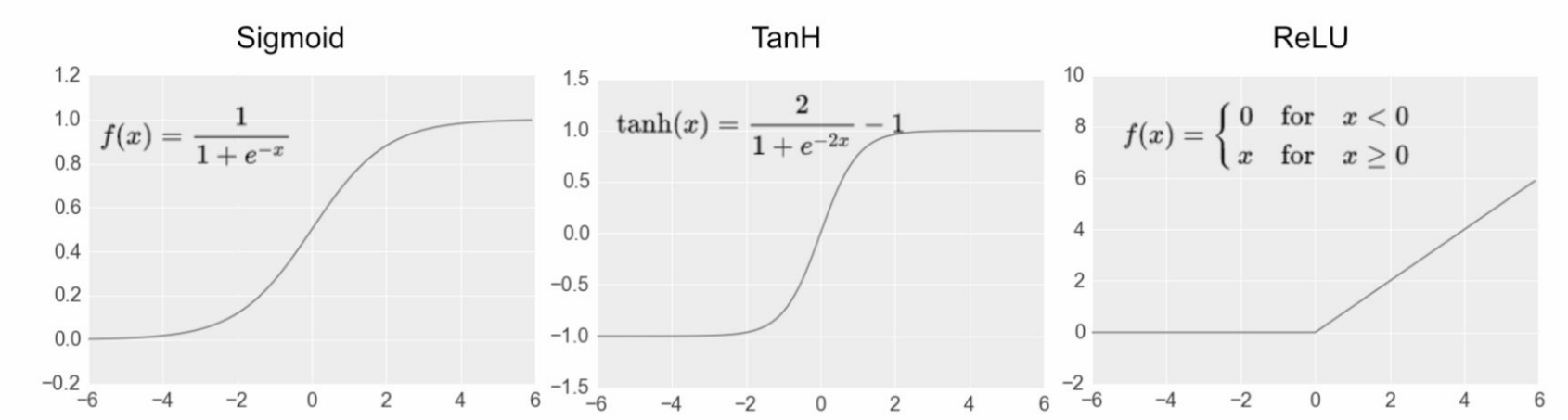
Date care nu sunt liniar separabile



https://cs.stanford.edu/people/karpathy/convnetjs/demo/classify2d.html

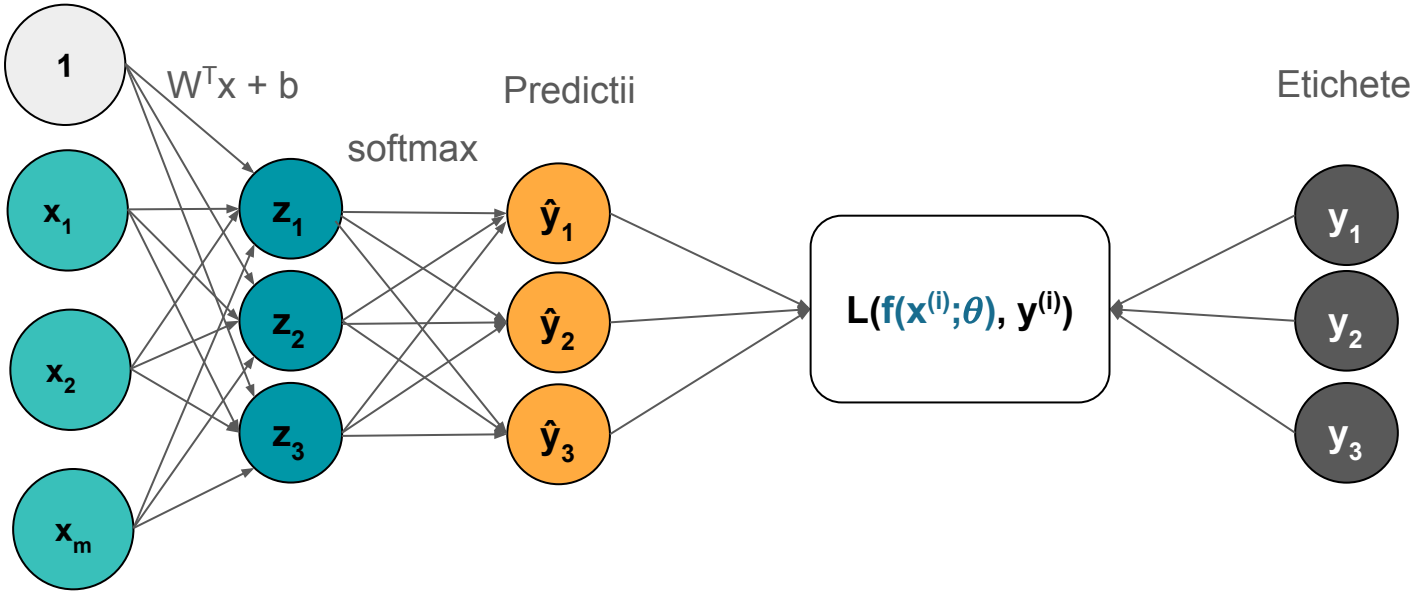
- Functiile de activare liniare gasesc un hiperplan (decision boundary) care imparte spatiul in 2 semiplane
- Cateodata datele **nu sunt liniar separabile**
- Avem nevoie de o **transformare** intr-un alt spatiu in care acestea pot fi separate de un hiperplan

Functii de activare



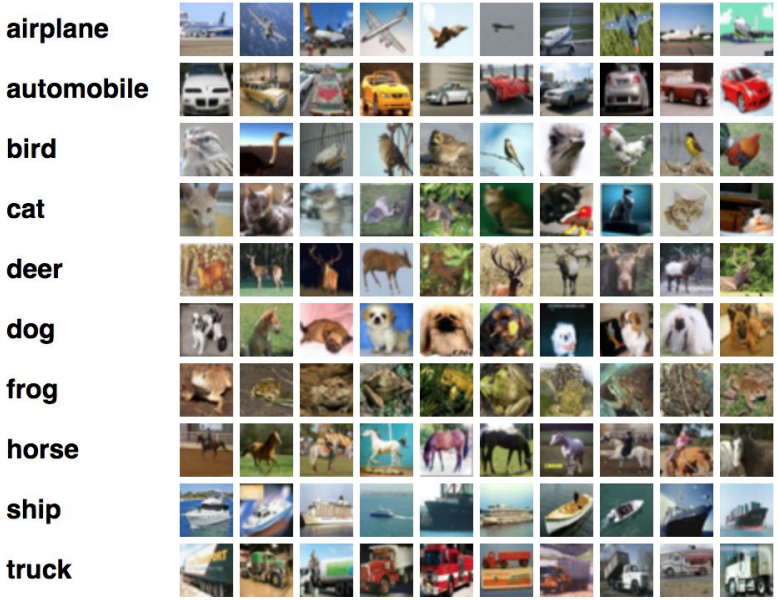
$$f'(x) = f(x)(1 - f(x)) \qquad f'(x) = 1 - f(x)^2 \qquad f'(x) = \begin{cases} 1, & x > 0 \\ 0, & \text{otherwise} \end{cases}$$

Masurarea costului



- **Cât de bun este un clasificator?**
- Avem un dataset de exemple $\{x_i, y_i\}_1^N$, vrem sa calculăm $\hat{y}_i \cong yi$
- **Funcția de cost** măsoară costul care trebuie plătit pentru predicții incorecte.

Algoritmi de învățare din date



The CIFAR-10 dataset

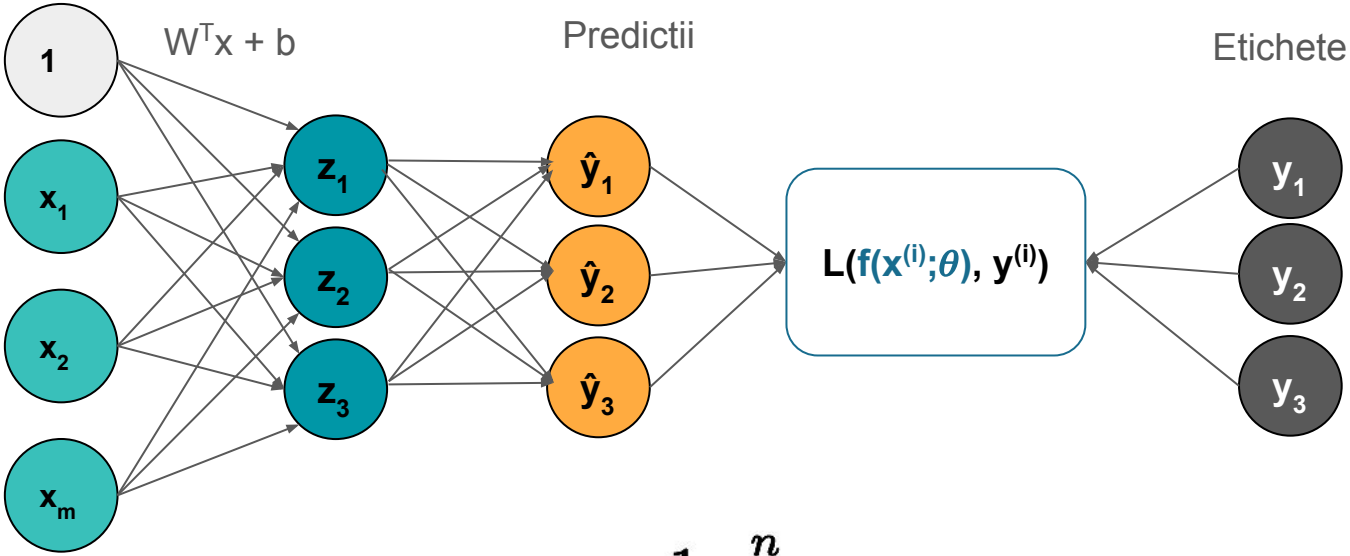
Dataset - perechi (imagine, eticheta)

50,000 imagini - antrenare [32x32x3]
10,000 imagini - evaluare

Abordare

- Algoritmul de clasificare = functie:
  - **f(imagine) = [p0, p1, p2 ...pn]**
- Aproximam functia cu niste parametri
  - **f(imagine; w) = [p0, p1, ...pn]**
- **Invatam** parametrii w din imaginile de antrenare
- **Evaluam** functia pe imaginile de evaluare

Costul empiric



pred	actual
0.2	0
0.5	1
0.3	0

$$J(\theta) = \frac{1}{n} \sum_{i=1}^{n} L(f(x(i); \theta), y^{(i)})$$

Exemple de funcții de cost (Loss functions)

Regression loss:

- Mean Squared Error

$$MSE = \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{n}$$

Classification loss:

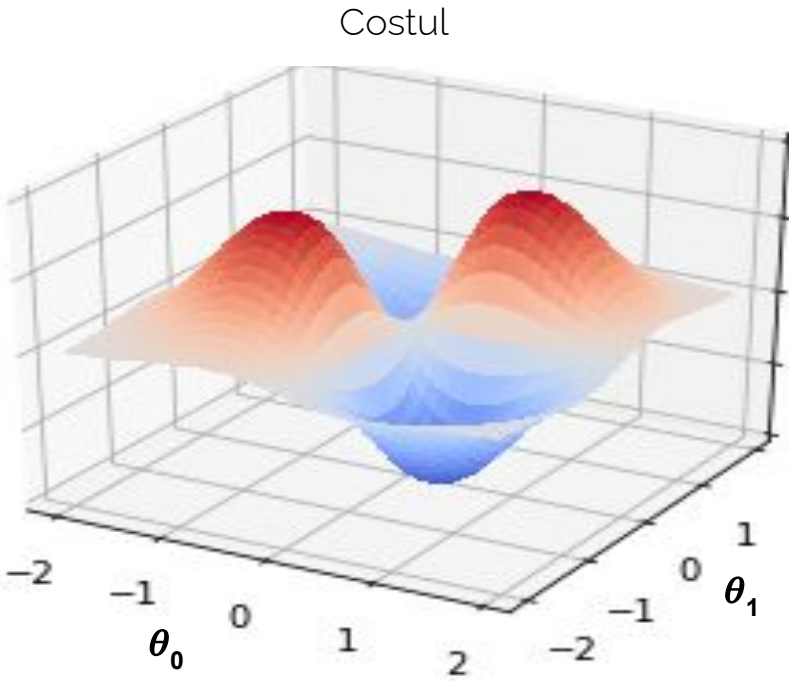
- Cross Entropy

$$L = -\frac{1}{m}\sum_{i=1}^{m} y_i \cdot \log(\hat{y}_i)$$

Optimizare

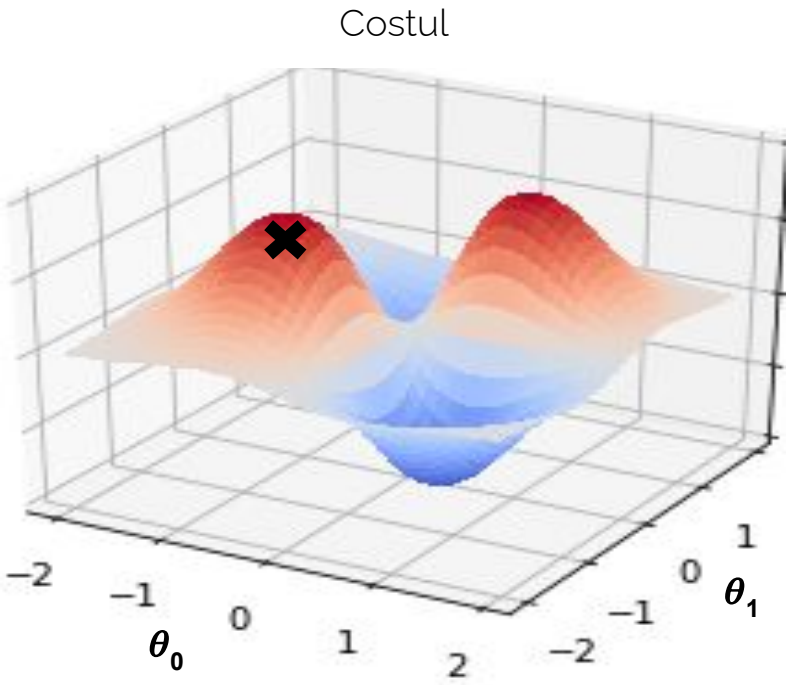
- Vrem sa găsim parametrii care minimizează costul

$$\theta* = argmin_\theta \frac{1}{n} \sum_{i=1}^{n} L(f(x(i);\theta), y^{(i)})$$



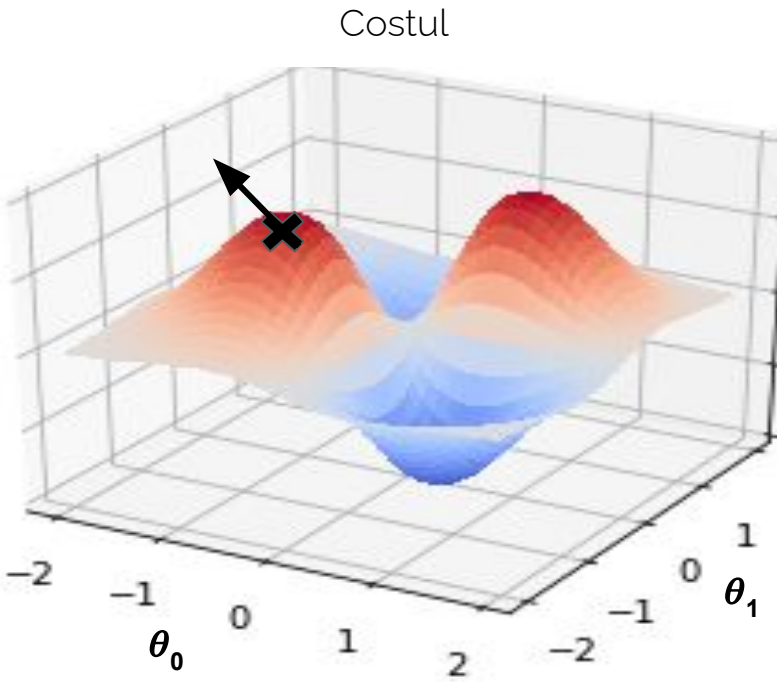
Stochastic Gradient Descent

- **Inițializăm aleator parametrii $\theta_0^{(0)}$ $\theta_1^{(0)}$**



Stochastic Gradient Descent

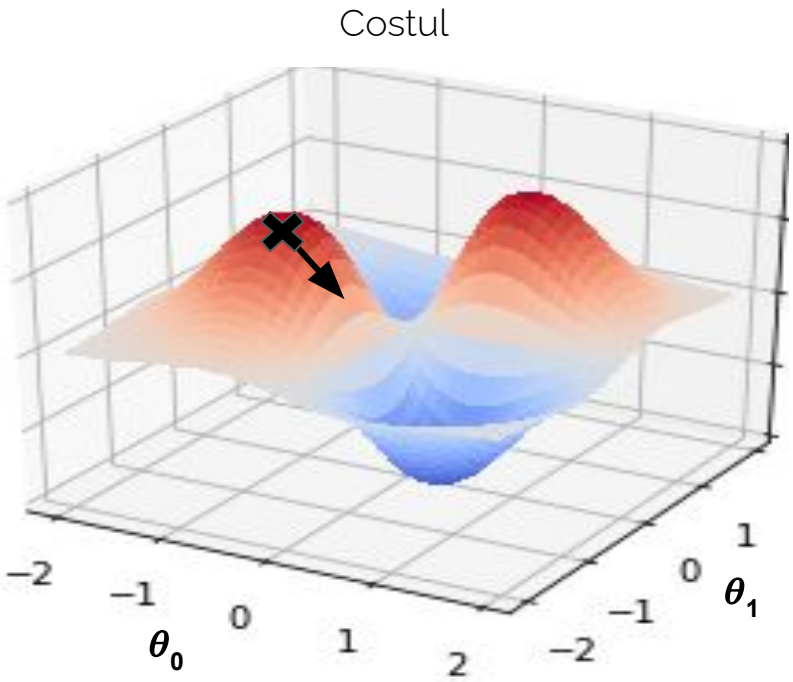
- Inițializăm aleator parametrii $\theta_0^{(0)}\ \theta_1^{(0)}$
- **Calculam gradientul (derivata)** $\frac{\partial L(\theta)}{\partial \theta}$



Stochastic Gradient Descent

- Inițializăm aleator parametrii $\theta_0^{(0)}\ \theta_1^{(0)}$
- Calculam gradientul (derivata) $\frac{\partial L(\theta)}{\partial \theta}$
- **Facem un pas mic în direcția opusă gradientului**

$$\theta^{(t)} \leftarrow \theta^{(t-1)} - \alpha \frac{\partial L(\theta^{(t-1)})}{\partial \theta^{(t-1)}}$$

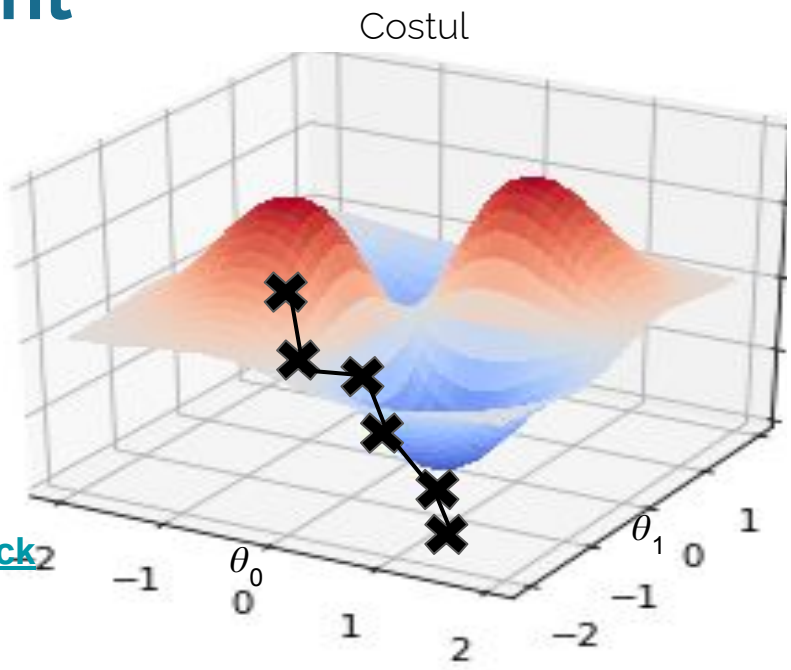


Stochastic Gradient Descent

- Inițializăm aleator parametrii $\theta_0^{(0)}\ \theta_1^{(0)}$
- Calculam gradientul (derivata) $\frac{\partial L(\theta)}{\partial \theta}$
- Facem un pas mic în direcția opusa gradientului

$$\theta^{(t)} \leftarrow \theta^{(t-1)} - \alpha \frac{\partial L(\theta^{(t-1)})}{\partial \theta^{(t-1)}}$$

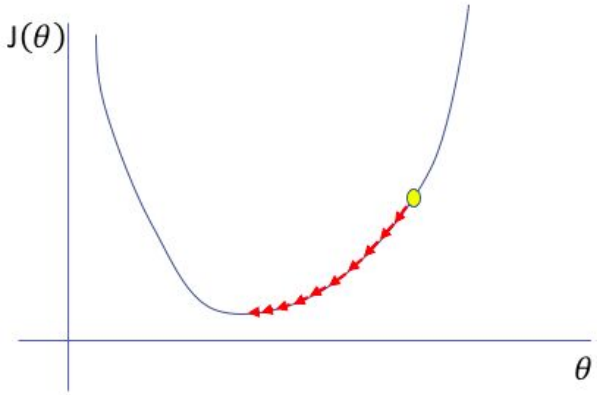
- **Repetăm până la convergență**
- https://www.youtube.com/watch?v=qg4PchTECck
- animation




```
while not_converged:
    weights_grad = evaluate_gradient(loss, data, weights) //backpropagation
    weights -= step_size * weights_grad //updatarea parametrilor
```

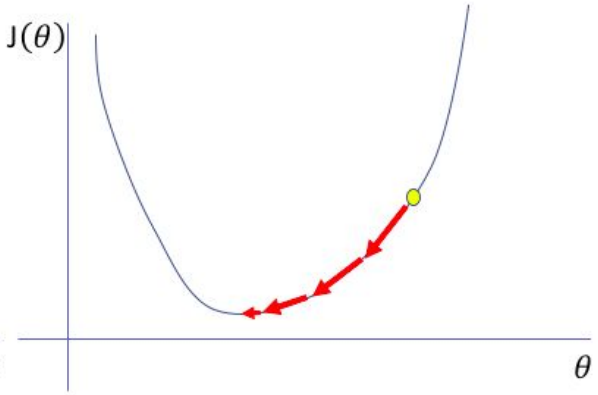
Cum sa alegem learning rate-ul?

Too low



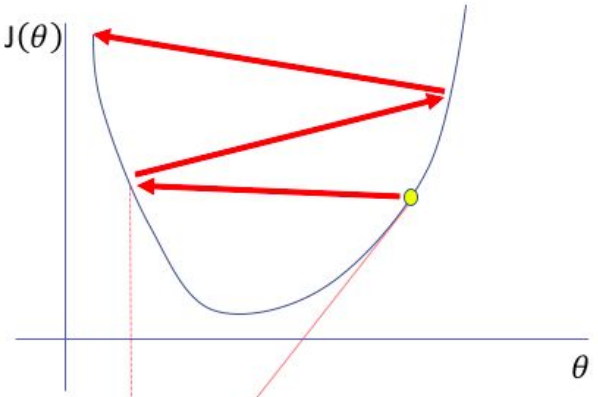
A small learning rate requires many updates before reaching the minimum point

Just right



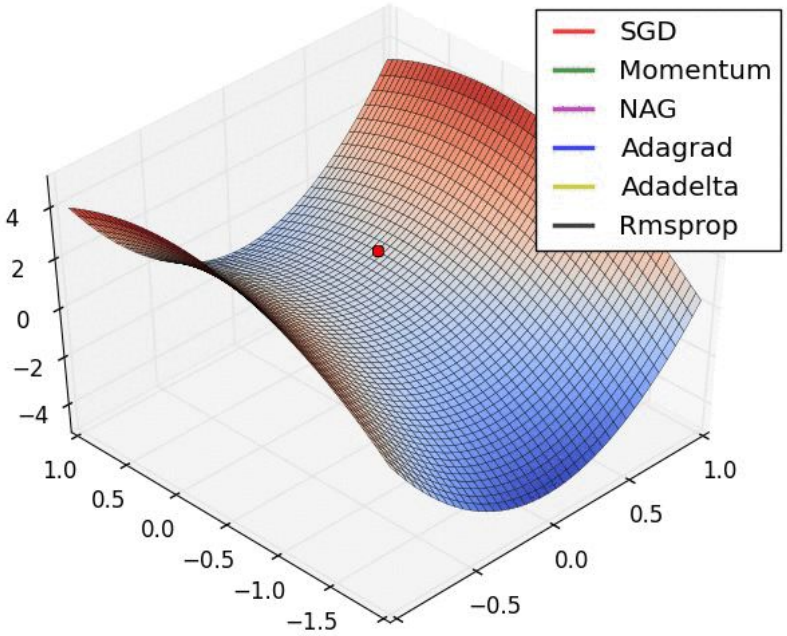
The optimal learning rate swiftly reaches the minimum point

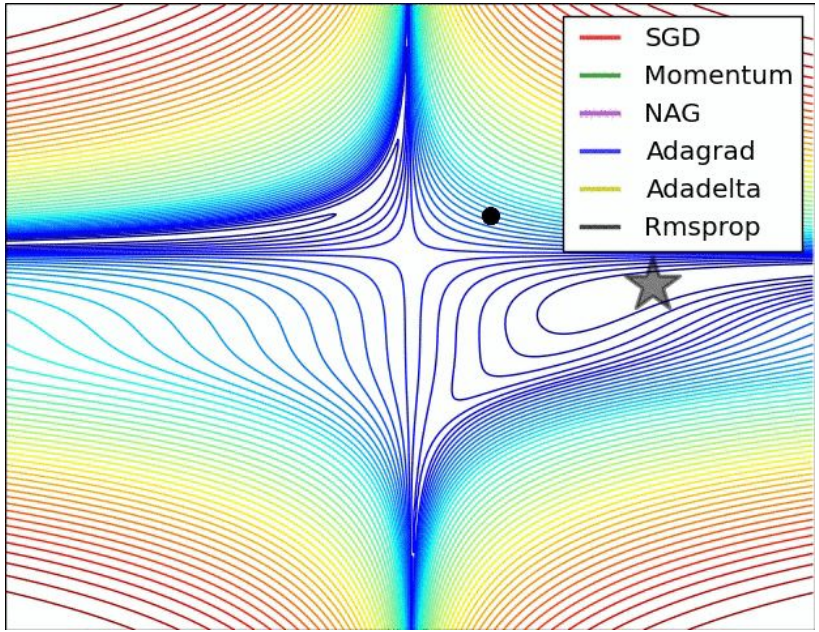
Too high



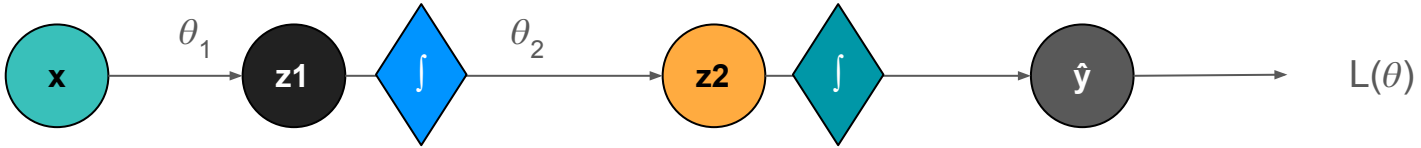
Too large of a learning rate causes drastic updates which lead to divergent behaviors

More optimizers :)



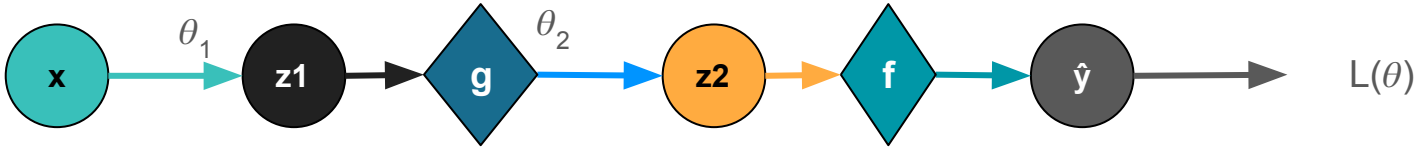


Calcularea gradientului - Backpropagation



- **Gradientul ne spune cum costul final L este afectat de o schimbare mică a parametrilor θ**
- In 1D, derivata unei functii L: $\frac{\partial L(\theta)}{\partial \theta} = \lim_{h \to 0} \frac{L(\theta + h) - L(\theta)}{h}$
- În mai multe dimensiuni gradientul este un **vector de derivate parțiale** pentru fiecare dimensiune
- Funcția obiectiv este parametrizata de θ => putem folosi reguli pentru a calcula **gradientul analitic**

Calcularea gradientului - Backpropagation



- The chain rule

$$\frac{\partial L(\theta)}{\partial \theta_2} = \frac{\partial L(\theta)}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial f} * \frac{\partial f}{\partial z_2} * \frac{\partial z_2}{\partial \theta_2}$$

$$\frac{\partial L(\theta)}{\partial \theta_1} = \frac{\partial L(\theta)}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial f} * \frac{\partial f}{\partial z_2} * \frac{\partial z_2}{\partial g} * \frac{\partial g}{\partial z_1} * \frac{\partial z_1}{\partial \theta_1}$$

Updatarea parametrilor

$$\theta \leftarrow \theta - \alpha \frac{\partial L(\theta)}{\partial \theta}$$

- **Stochastic vs batch gradient descent**
  - **Stochastic Gradient Descent**
    - Modificăm parametrii după fiecare exemplu
    - Exemple On-line, dataset-uri redundante foarte mari
  - **Batch Gradient Descent**
    - Modificăm parametrii după ce calculăm eroarea (L) peste toate exemplele din dataset
    - Poate fi paralelizat, eroarea (L) este estimată foarte bine
  - **Mini-batch Gradient Descent (cateodată denumit și SGD - stochastic gradient descent)**
    - Modificăm parametrii după ce calculăm eroarea (L) peste un mini-batch de exemple din dataset (32, 64, 128, 256, 512, 1024)

Initializarea parametrilor

- **Initializare cu zero ?**
  - Fiecare neuron calculeaza acelasi output, același gradient și execută același update
- **Inițializare cu numere mici random**
  - Fiecare neuron este unic și calculează update-uri distincte
  - Inițializare dintr-o gaussiana centrată în 0 cu varianta = sqrt(0.01)

```
W = 0.01 * np.random.randn(n)
```

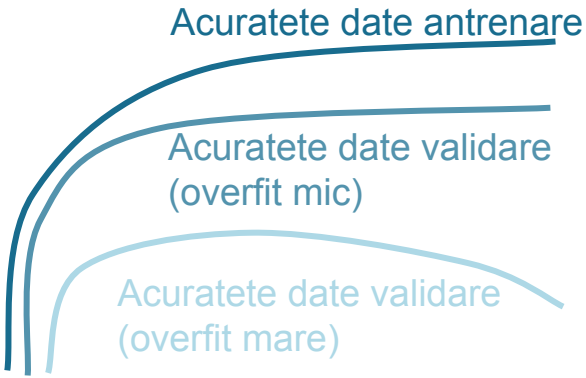
  - Dacă parametrii sunt prea mici, gradientul (care e proporțional pe valoarea parametrilor) va fi mic
- **Calibrarea variantei** `w = np.random.randn(n) / sqrt(n)`
  - Distribuția activărilor unui neuron inițializat random crește cu numărul de intrări
  - Putem normaliza varianța fiecărui neuron pentru ca output-ul să aibă varianța 1

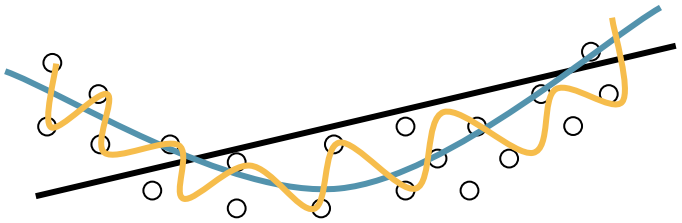
Overfitting

- **Generalizare**
  - Proprietatea unui estimator funcțional de a generaliza dincolo de exemplele pe care a fost antrenat

antrenare		evaluare
antrenare	validare	evaluare

50,000 imagini - antrenare [32x32x3]
10,000 imagini - evaluare





underfitting

Ideal fit

overfitting

Overfitting

- **Regularizare**
  - **Aplicarea de constrângeri asupra problemei de optimizare pentru a descuraja modele complexe**
  - Îmbunătățește capacitatea de a generaliza a modelului pe date pe care nu le-a mai văzut
  - Tehnici: **L1 norm, L2 norm, Dropout, Early stopping, label smoothing** (in episodul urmator…)

Sumar concepte fundamentale

Perceptronul

- ★ Unitatea de baza
- ★ Clasificare
- ★ Funcții de activare neliniare

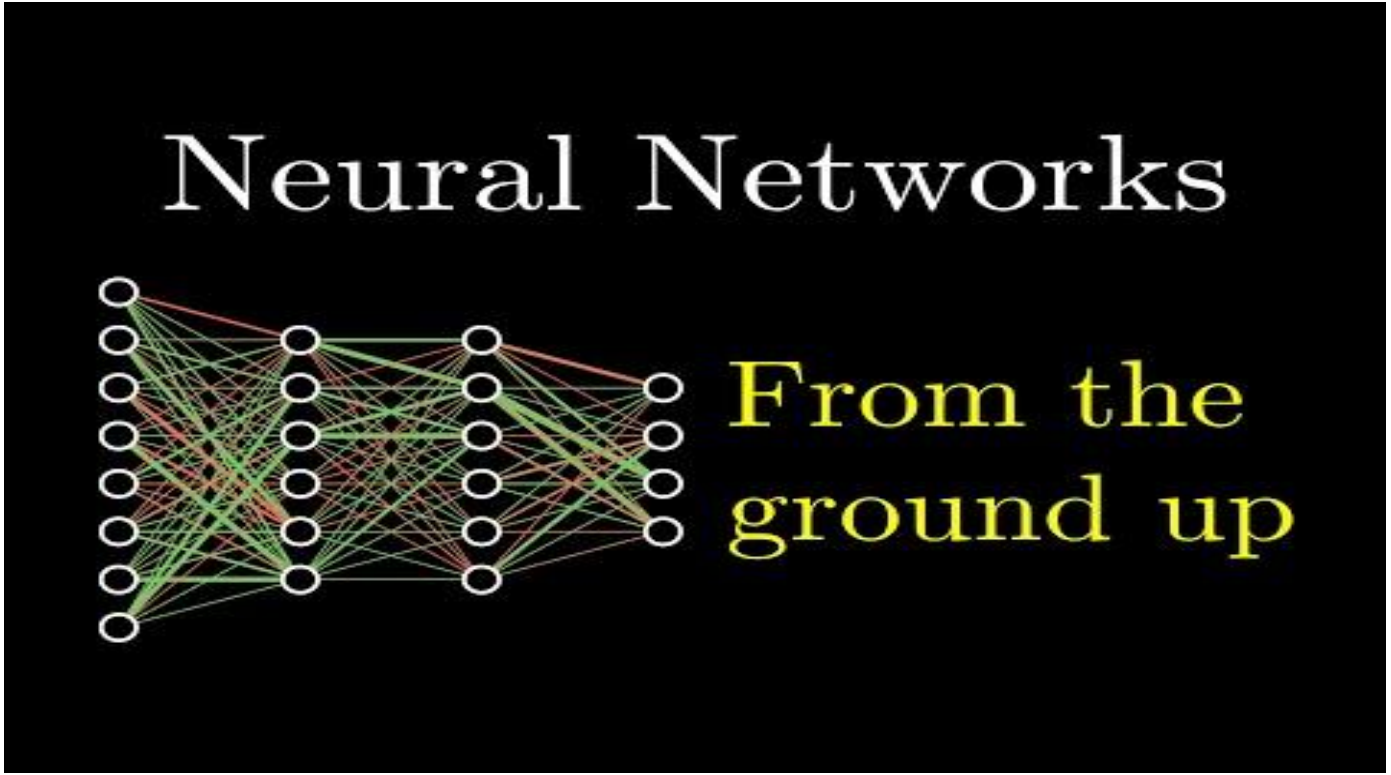
Rețele cu un singur strat

- ★ Suprapunerea perceptronilor pentru a construi rețele
- ★ Funcții de cost

Antrenare

- ★ Mini-batch Stochastic gradient descent
- ★ Rate de invatare
- ★ Regularizare

Resurse - 1



Resurse - 2