BIG DATA

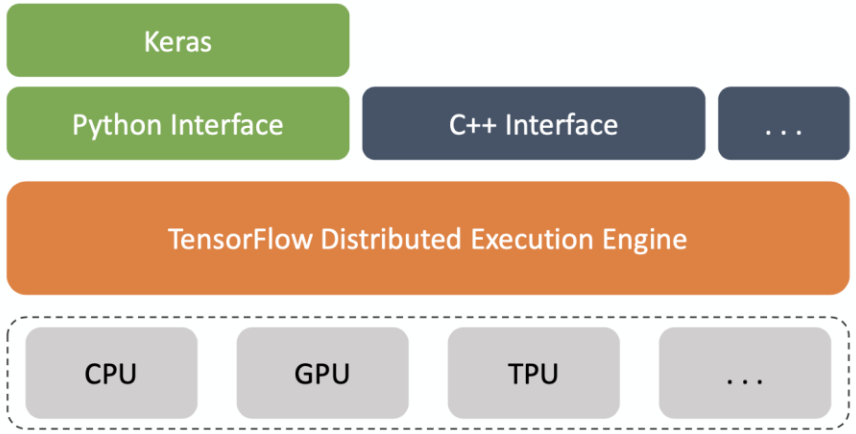
CURS 14

Introducere

Introducere în TensorFlow

Introducere în TensorFlow

- TensorFlow: bibliotecă open-source de *machine learning*. Este concepută în principal pentru task-uri orientate *deep learning*.
- Dezvoltată și menținută de Google
- Utilizată atât în cercetare cât și în producție
- Caracteristici:
  - Reprezintă datele sub formă de tensori
  - Reprezintă calculele sub formă de grafuri



Introducere în TensorFlow

Tensori

- TensorFlow utilizează o structură de date denumită Tensor, pentru reprezentarea oricărei date.
- Un Tensor este foarte similar unui tablou numpy
- Tensorii sunt entitățile ce parcurg grafurile de calcul
- Tensorii sunt obiecte imutabile (precum tuplurile din Python, DataFrame-urile din Spark SQL) – conținutul unui tensor nu se poate modifica.
  - În cazul în care este necesară o modificare, se creează un tensor nou.

Introducere în TensorFlow

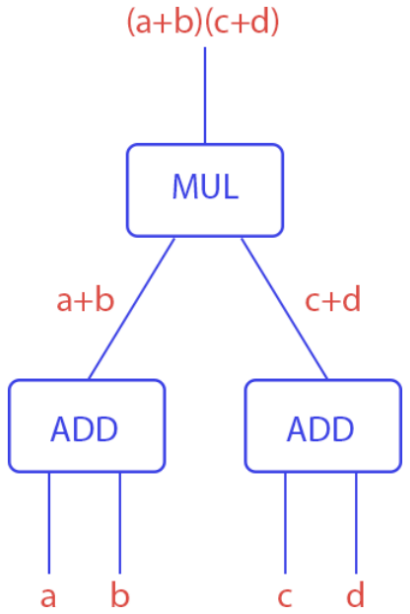
Crearea unui tensor

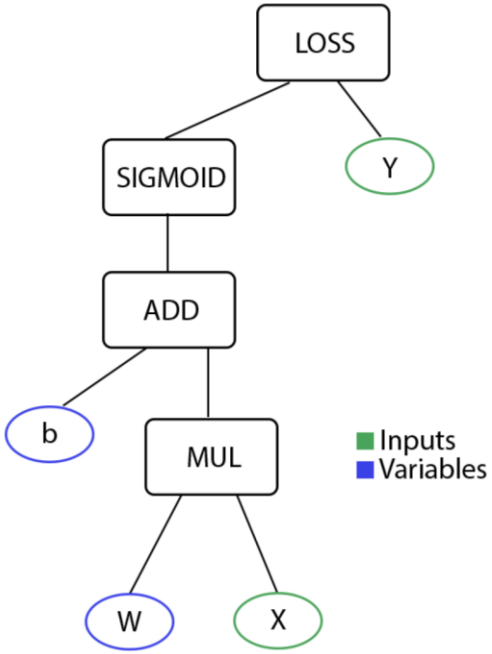
```
import tensorflow as tf
import numpy as np


t1 = tf.constant([1., 2., 3.]) # Tensor real cu valorile (1, 2, 3)
t2 = tf.zeros(shape=(3,3)) # Tensor real de dimensiune 3x3 cu toate valorile = 0
a = np.array([1., 2., 3.])
t3 = tf.constant(a)# Tensor real creat pornind de la un tablou numpy
t4 = tf.random.uniform(shape=(10,), 0, 11) # Tensor aleator, cu repartiția (distribuția) uniformă
```

Introducere în TensorFlow

Graf de calcul

- Graf computațional: reprezentare a expresiilor matematice. Totodată, este un fundament al TensorFlow
- Fie expresia h=(a+b)(c+d)
  - Expresia are 3 operații: 2 adunări și o înmulțire
- În graf, fiecare nod corespunde une intrări sau unei operații, iar fiecare arc este un tensor ce reprezintă fluxul de date între noduri.
- Majoritatea expresiilor matematice pot fi reprezentate în acest mod, prin urmare TensorFlow nu este limitat la deep learning.





Introducere în TensorFlow

- **Operații și proprietăți**

Dimensiune	`tensor.shape`
Modificare de tip	`tf.cast(tensor, dtype=…)`
Înmulțire matricială	`tf.matmul(tensor1, tensor2)`
Transpoziție de matrice	`tf.transpose(tensor)`
Modificarea dimensiunii	`tensor.reshape(tensor, shape=…)`
Concatearea tensorilor	`tf.concat([tensor1, tensor2], axis=…)`
Calculul sumei	`tf.reduce_sum(tensor)`
Inversarea unui tensor	`tf.linalg.pinv(tensor)`

Introducere în TensorFlow

- Demo1

Rețele neuronale

Rețele neuronale

(Prezentare din punct de vedere intuitiv)

- Neuron
- Funcție de activare
- Cum funcționează un ANN?
- Cum învață o rețea?
- Retropropagare

Rețele neuronale

Deep Learning – librării

Neuron

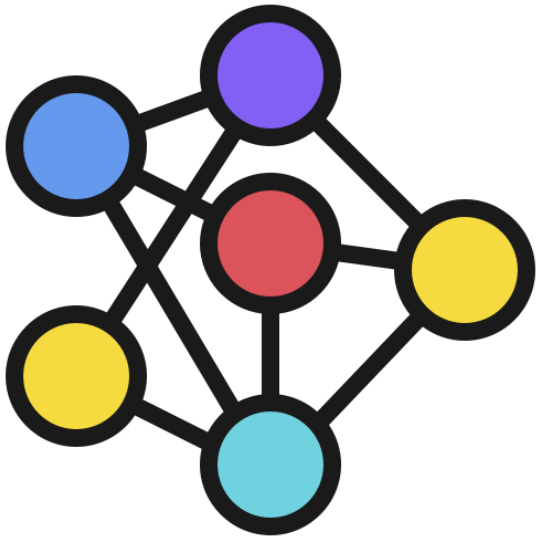
0
1
2
3
4
5
6
7
8
9

recognize

Cum recunoaște creierul o imagine?

Neuron

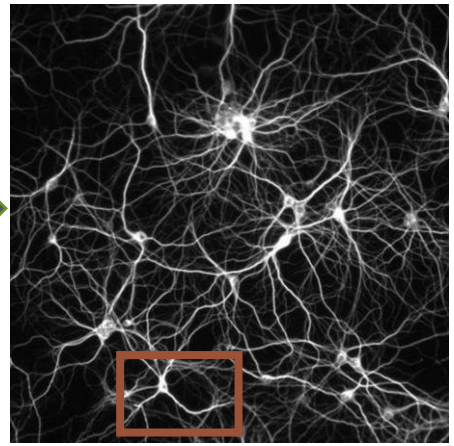
- Pentru a începe să înțelegem Deep Learning, vom construi modelele de abstractizare următoare:
  - Neuron biologic simplu
  - Modele de perceptroni
  - Perceptron cu mai multe straturi
  - Rețele de neuroni profunde

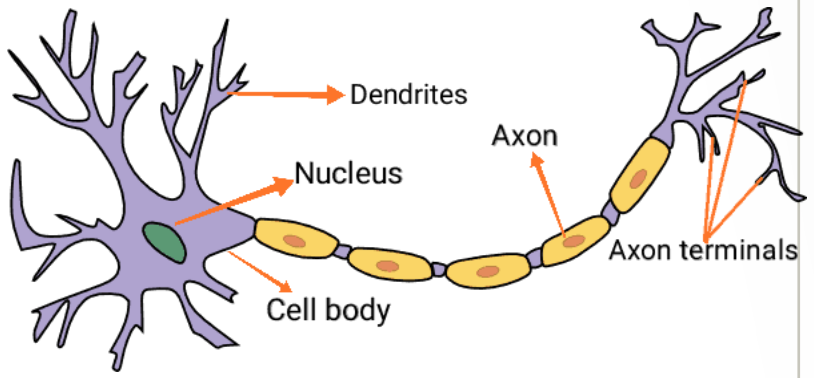


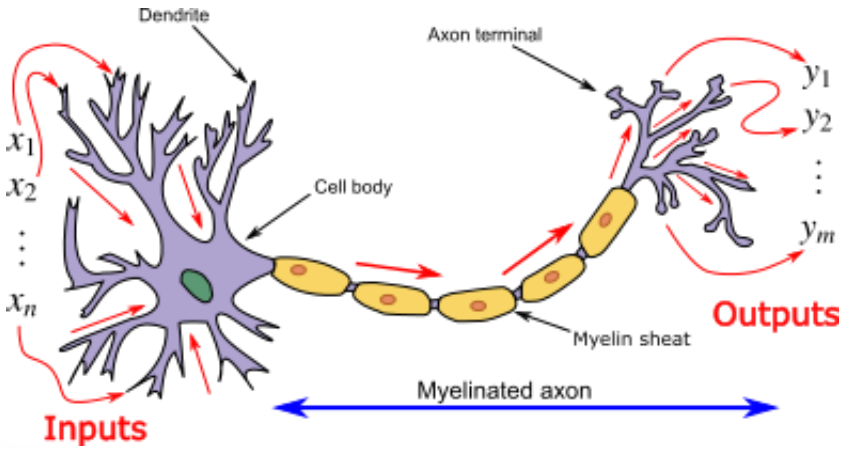
Neuron

- Ideea generală a Deep Learning este aceea de a imita artificial, prin intermediul calculatoarelor, inteligența neuronală biologică
- Care este modul general de funcționare a neuronilor biologici?

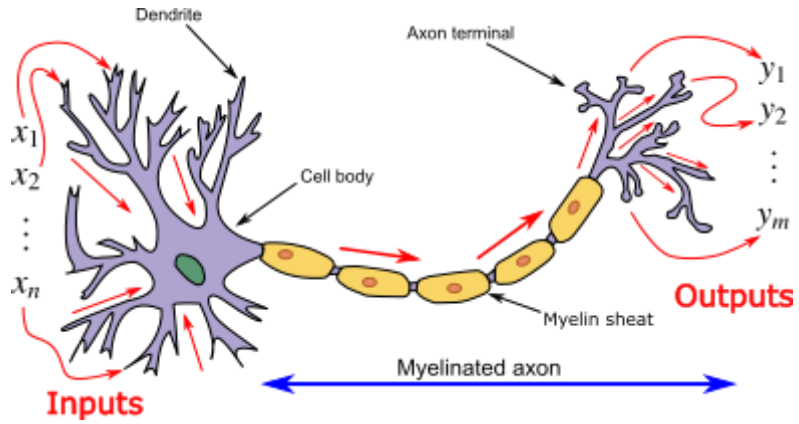
Neuron




Dendrites
Axon
Nucleus
Axon terminals
Cell body


Dendrite
Axon terminal
x_1
x_2
x_n
Cell body
y_1
y_2
y_m
Outputs
Myelin sheat
Myelinated axon
Inputs

Neuronul biologic



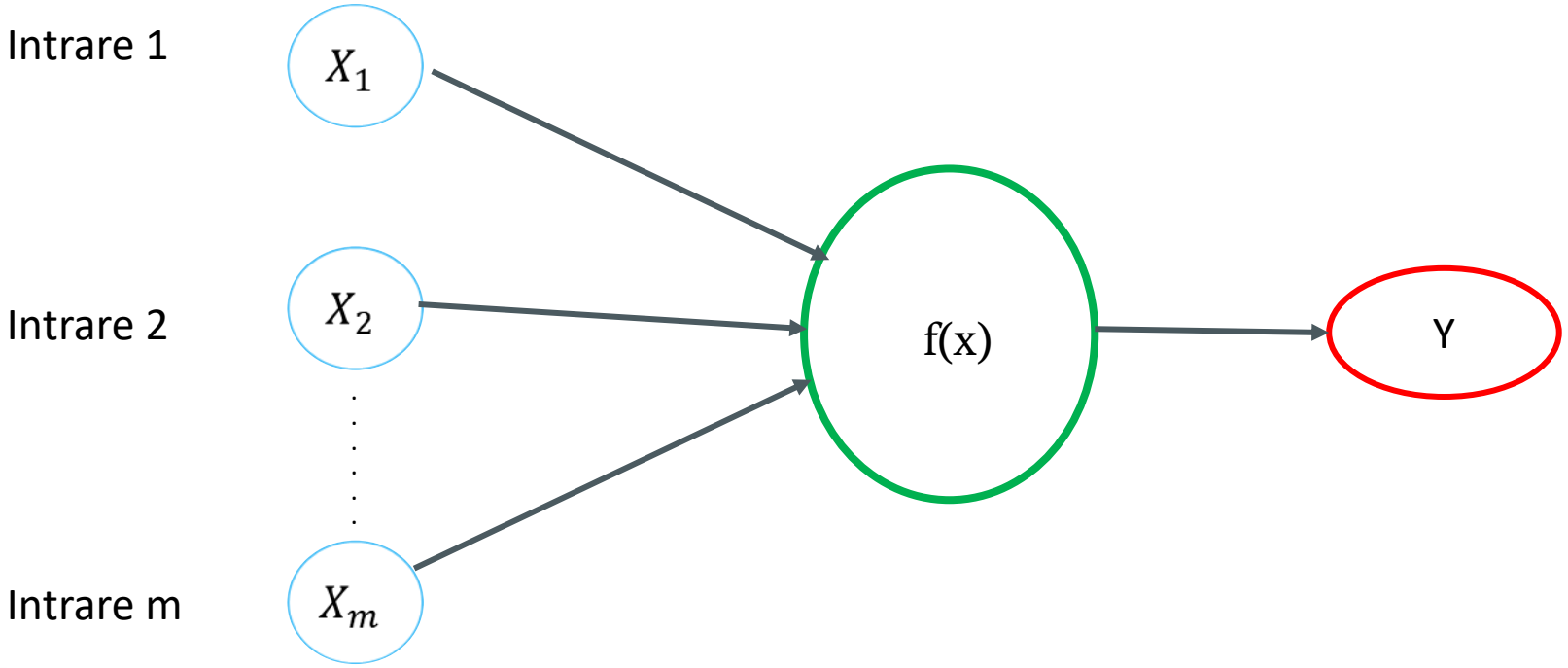
Cum se poate converti schema simplificată a unui neuron biologic într-un model matematic?

Scurt istoric

- 1957: Frank Rosenblatt a inventat cea mai simplă rețea neuronală
- Neuron formal, cu o regulă de învățare ce determină automat ponderile sinaptice astfel încât să poată separa o problemă de învățare supervizată.
- Fapt surprinzător la acel moment: i se prevedea un potențial enorm
  - Perceptronul putea fi capabil să învețe, să ia decizii și să traducă.

Neuron

- Convertim modelul simplu de neuron biologic în model de perceptron „Neuron Artificial” al lui Rosenblatt

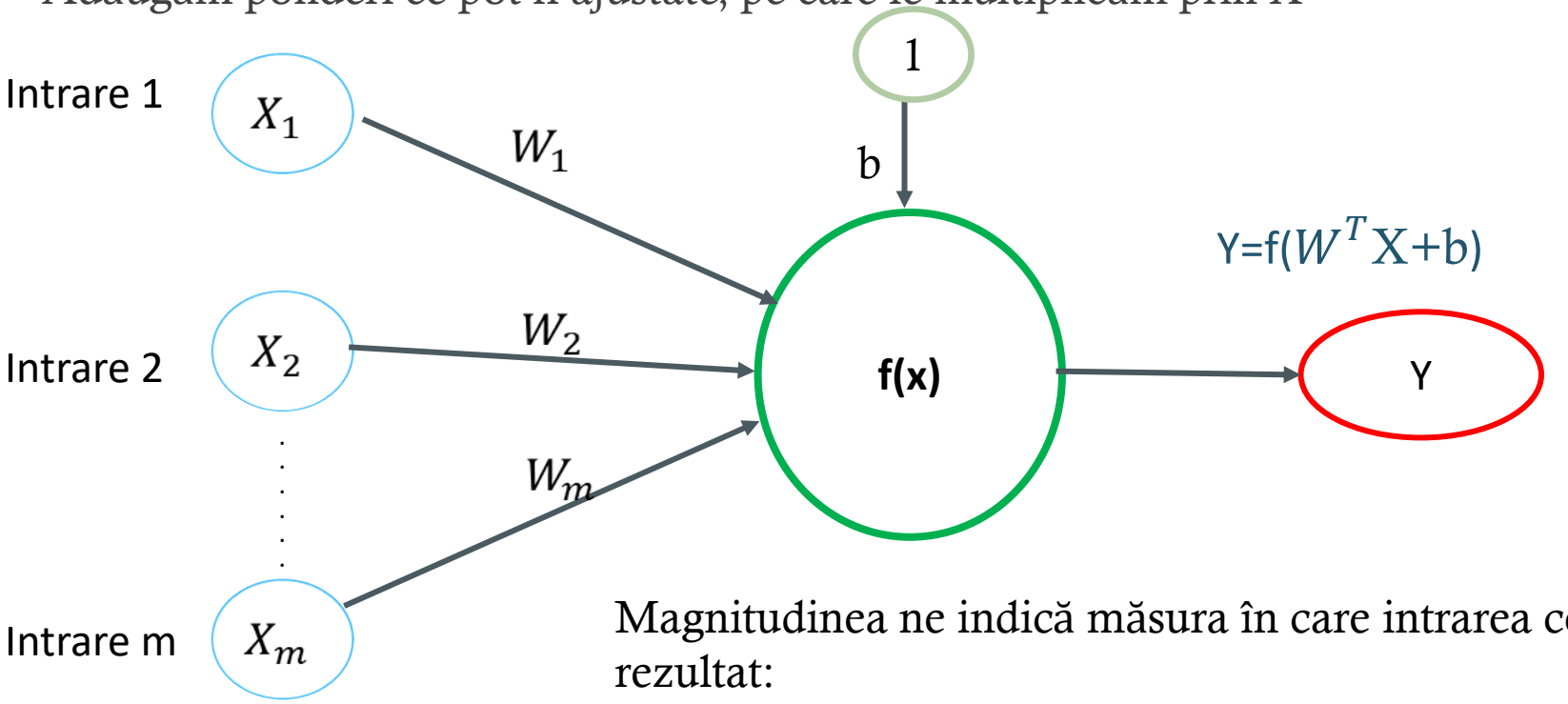


Una sau mai multe valori de ieșire, ce pot fi:

- Continue
- Binare

Neuron

- În mod realist, trebuie să ajustăm anumiţi parametri pentru a permite învăţarea
- Adăugăm ponderi ce pot fi ajustate, pe care le multiplicăm prin X



Scriere matricială 1

Y = W^TX+b, unde
W^T=(W_1, W_2, …., W_m)
X^T=(X_1, X_2, …., X_m)

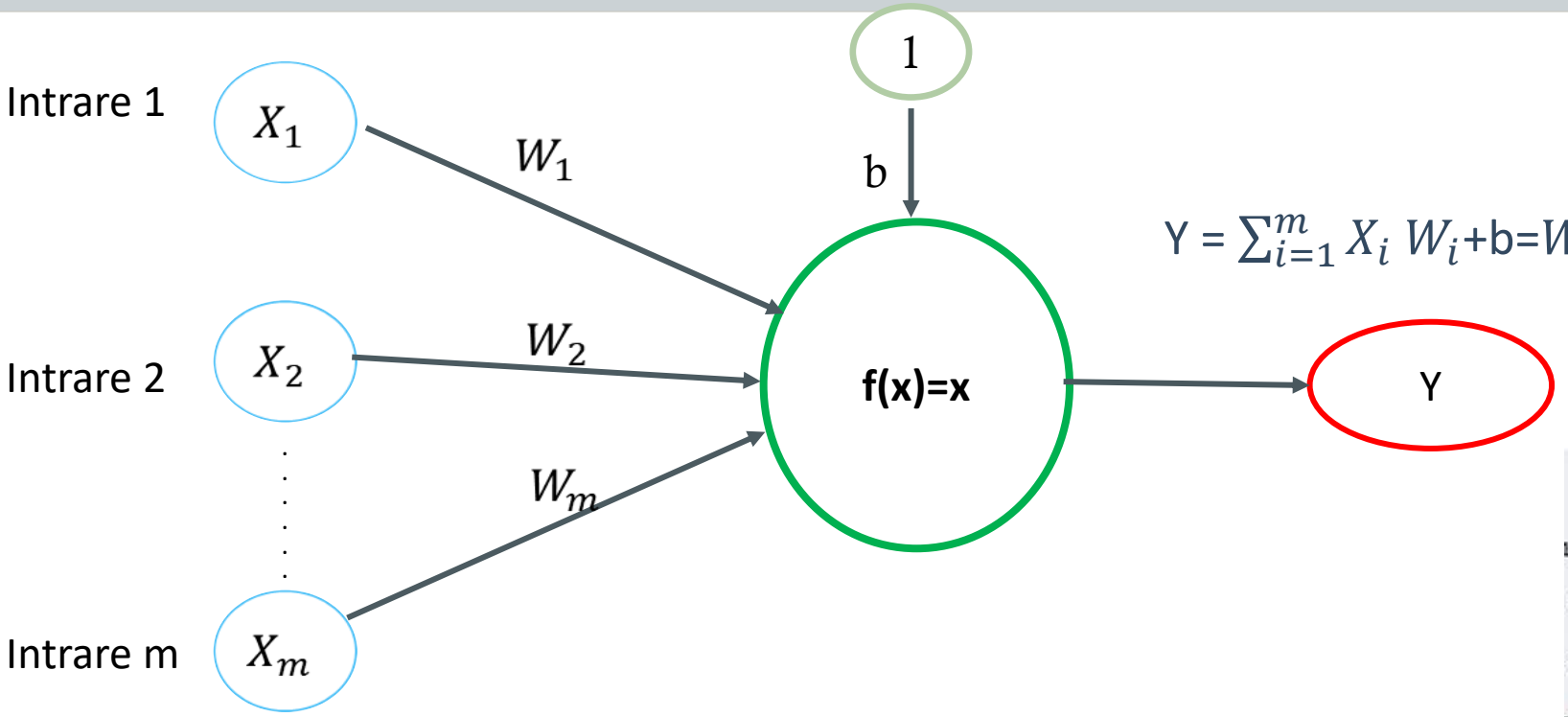
Scriere matricială 2

Y = W^TX, unde
W^T=(b,W_1, W_2, …., W_m)
X^T=(1,X_1, X_2, …., X_m)

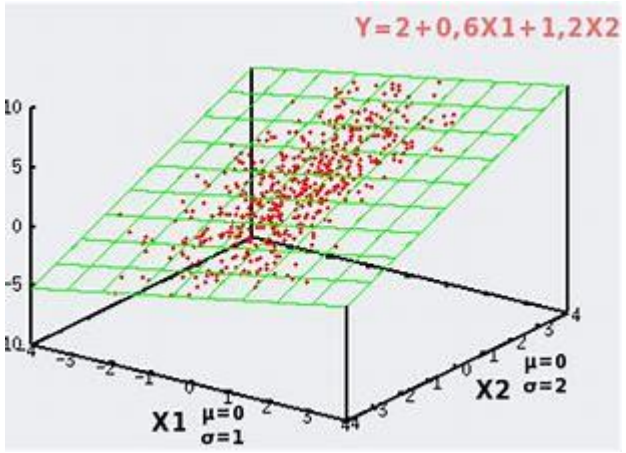
Magnitudinea ne indică măsura în care intrarea contribuie la rezultat:

- Dacă $W_i = 0$, atunci X_i nu are influenţă
- Dacă $|W_i|$ este mare, atunci intrarea respectivă are o influenţă mare asupra predicţiei

Neuron



$Y = \sum_{i=1}^{m} X_i W_i + b = W^T X + b$



Remarcăm asemănarea cu regresia.
Ponderile sunt coeficienții planului de regresie al lui Y în funcție de X=(X_i), i=1,….,M

Neuronul – Model liniar

- Cum se pot determina coeficienții ecuației de regresie?
  1. Se transmit toate observațiile (*P* observații de dimensiune *D*) rețelei => se va obține un vector de ieșiri
  2. Se caută să se minimizeze funcția cost între valorile reale ale variabilei țintă și ieșirile rețelei
  3. Funcția cost în acest caz este MSE (Mean Square Error):

$$C=\frac{1}{P}\sum_{i=1}^{P}(y_i-\hat{y}_i)^2=\frac{1}{P}\sum_{i=1}^{P}(y^i-W^T X^i)^2 \qquad C=\frac{1}{P}\left\|y^T-W^T X\right\|^2$$

$y=(y^1, y^2, \ldots, y^P)$ este un vector

$X=(X^1, X^2, \ldots, X^P)$ este o matrice

Neuronul – Model liniar

- Vom determina ponderile care minimizează funcția de cost C
- Vom căuta $\boldsymbol{W}= (\boldsymbol{W_1}, \boldsymbol{W_2}, \ldots . \boldsymbol{W_D})$ astfel ca: $\frac{\partial C}{\partial W} = 0$

$$\left(\frac{\partial C}{\partial W}\right)^T = \left(\frac{\partial C}{\partial W_1}, \frac{\partial C}{\partial W_2}, \ldots . , \frac{\partial C}{\partial W_D}\right)$$

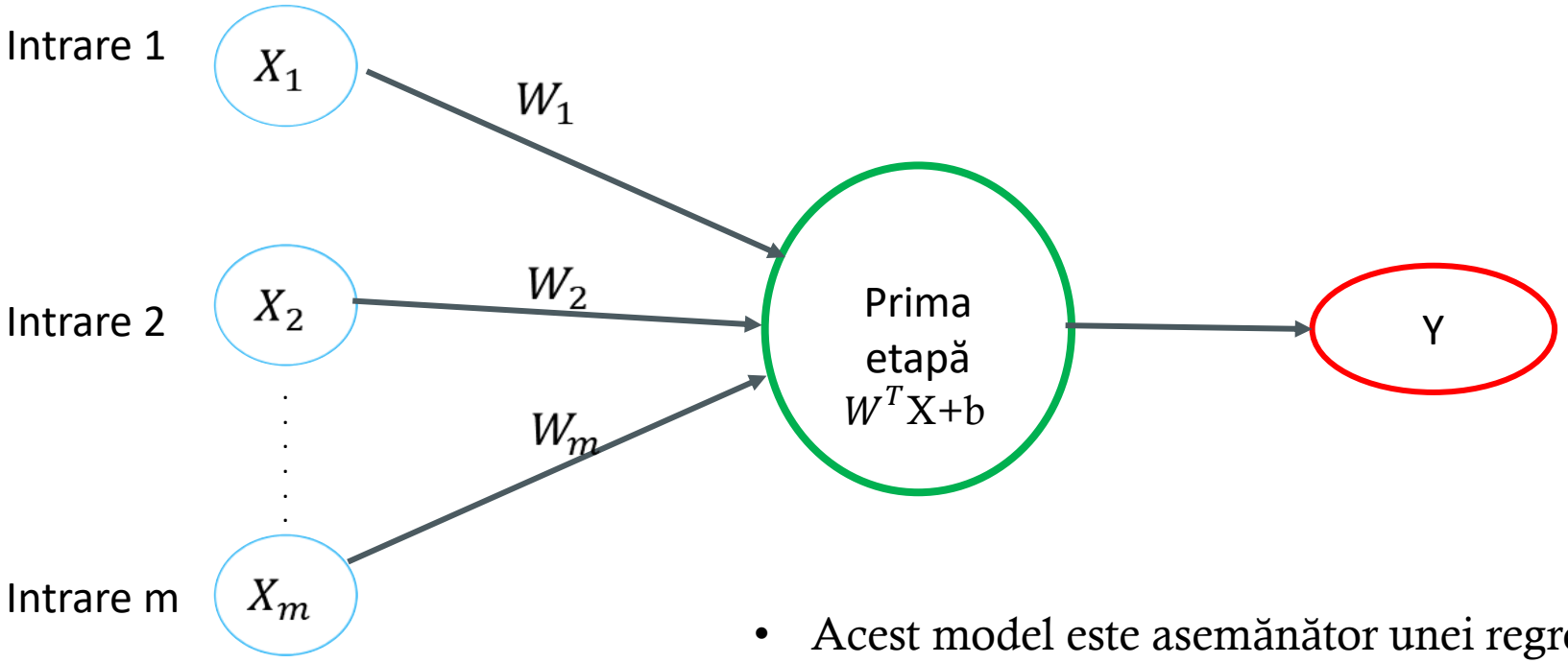
Atunci $$\left(\frac{\partial C}{\partial W}\right)^T = \frac{2}{P}\left(W^T X - y^T\right) X^T$$

$$\frac{\partial C}{\partial W_i} = \frac{1}{P}\frac{\partial\left(\|y^T - W^T X\|^2\right)}{\partial W_i}$$
$$= \frac{1}{P}\frac{\partial\left(y^T - W^T X\right)\left(X^T W - y\right)}{\partial W_i}$$
$$= \frac{2}{P}\left(W^T X - y^T\right) X^i$$

Minimum este atins atunci când :

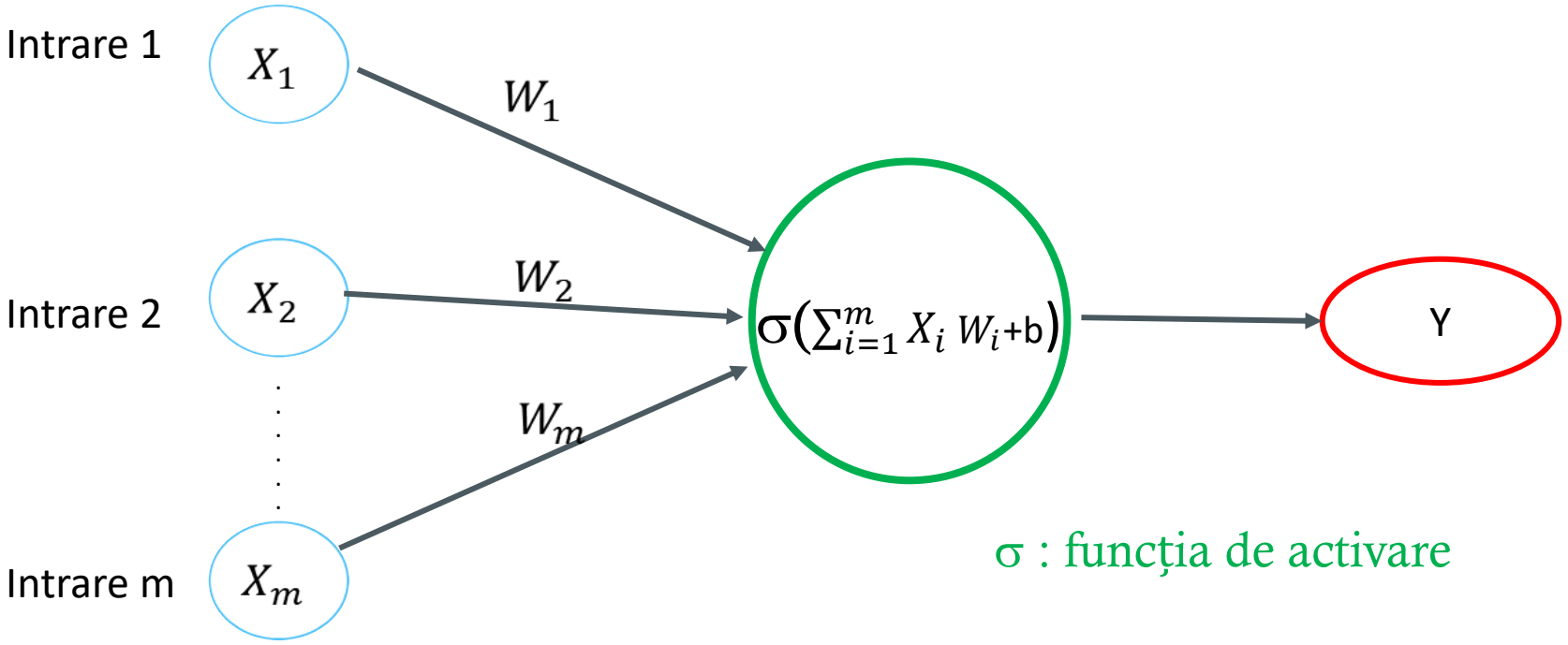
$$\left(\frac{\partial C}{\partial W}\right)^T = 0 \quad \Longrightarrow \quad W = \left(X^T X\right)^{-1} X y$$

Neuron



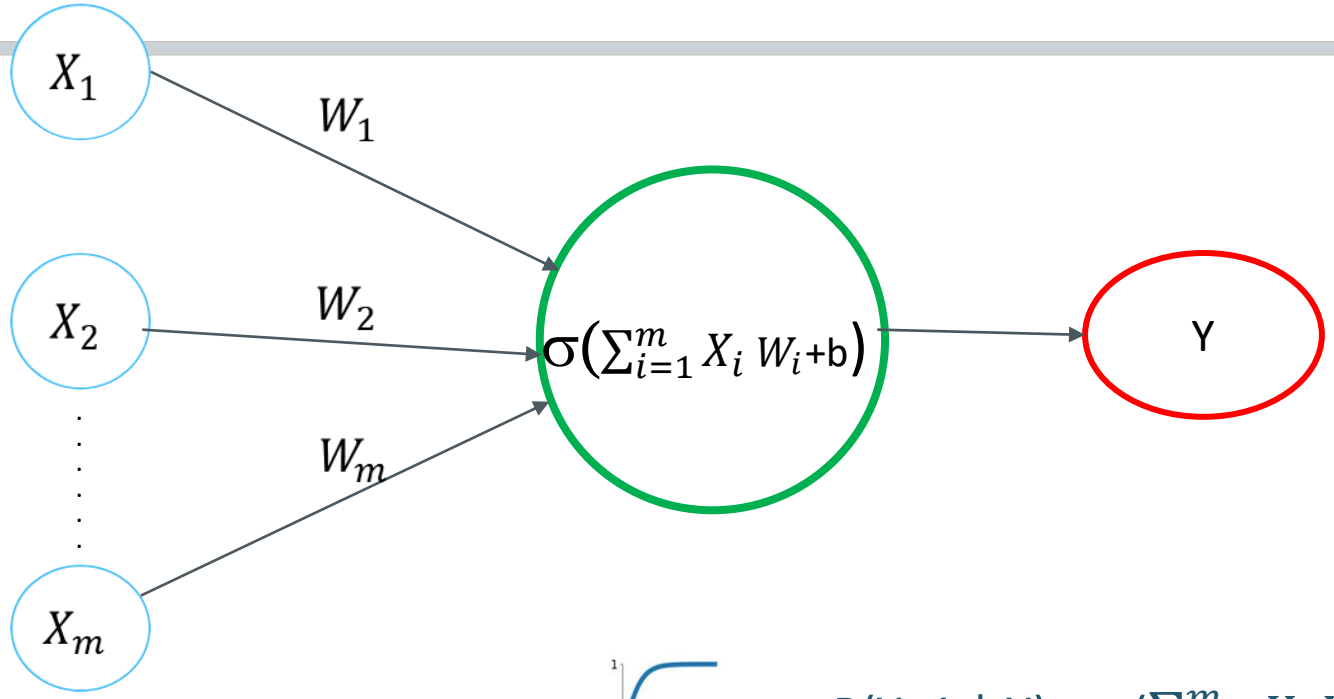
- Acest model este asemănător unei regresii liniare
  - Util în mare parte a cazurilor de utilizare
- Cu toate acestea, este necesar să modelăm funcţii mai complexe, deci trebuie introdus un operator neliniar

Neuron – Funcția de activare



- În modelele de clasificare, este util ca toate rezultatele să se afle între 0 și 1
- Aceste valori pot reprezenta probabilitatea de asociere pentru fiecare clasă

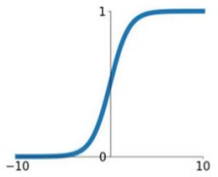
Neuron – Funcția de activare



b= deplasarea (*bias*)

Dacă $W^T X$ > b atunci modelul va furniza predicția 1
Dacă $W^T X$ < b atunci modelul va furniza predicția 0

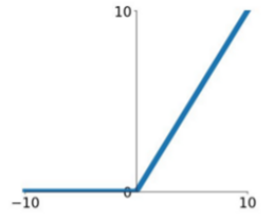
Dacă σ este funcția Sigmoid



$P(Y=1 \mid X) = \sigma(\sum_{i=1}^{m} X_i\ W_i+b)$

Se poate observa asemănarea cu regresia logistică.

Dacă σ este funcția ReLU

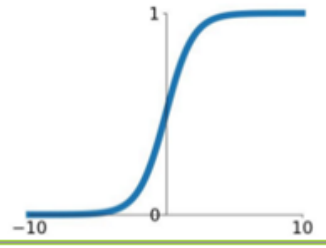


$Y= \sigma(\sum_{i=1}^{m} X_i\ W_i+b)$

Funcții de activare

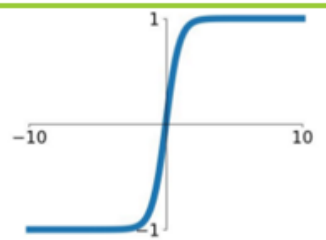
Sigmoid

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



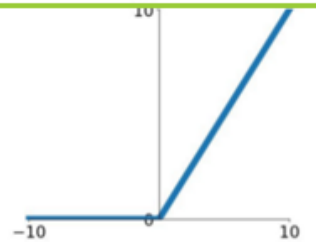
tanh

$$\tanh(x)$$



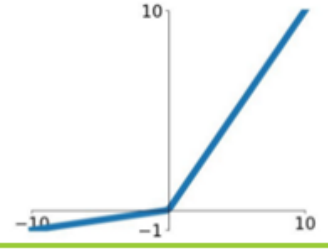
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

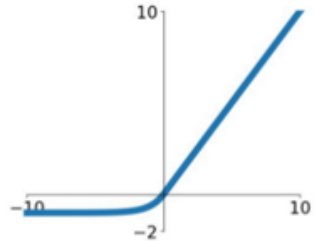


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

ELU

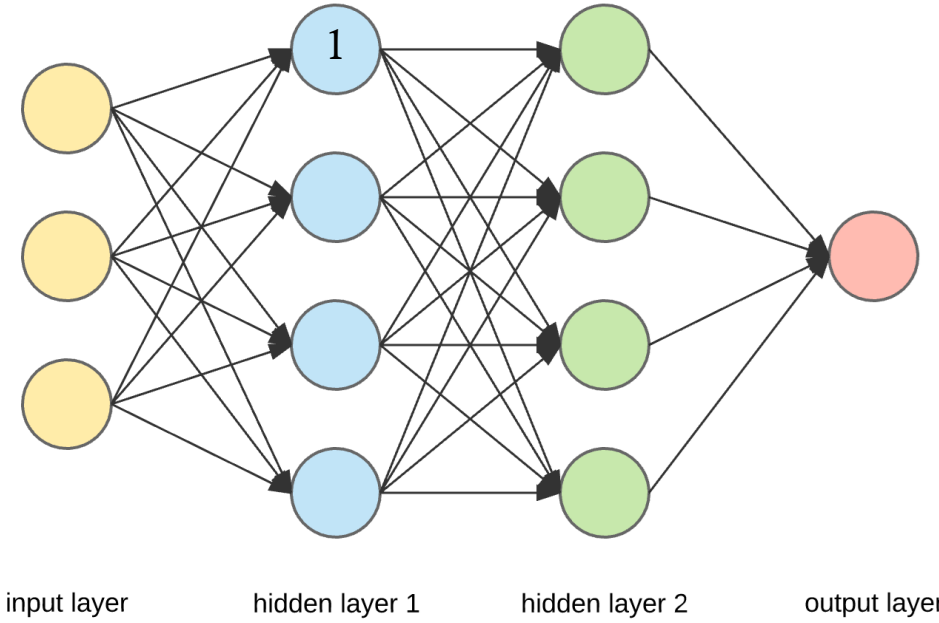
$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Rețele de neuroni

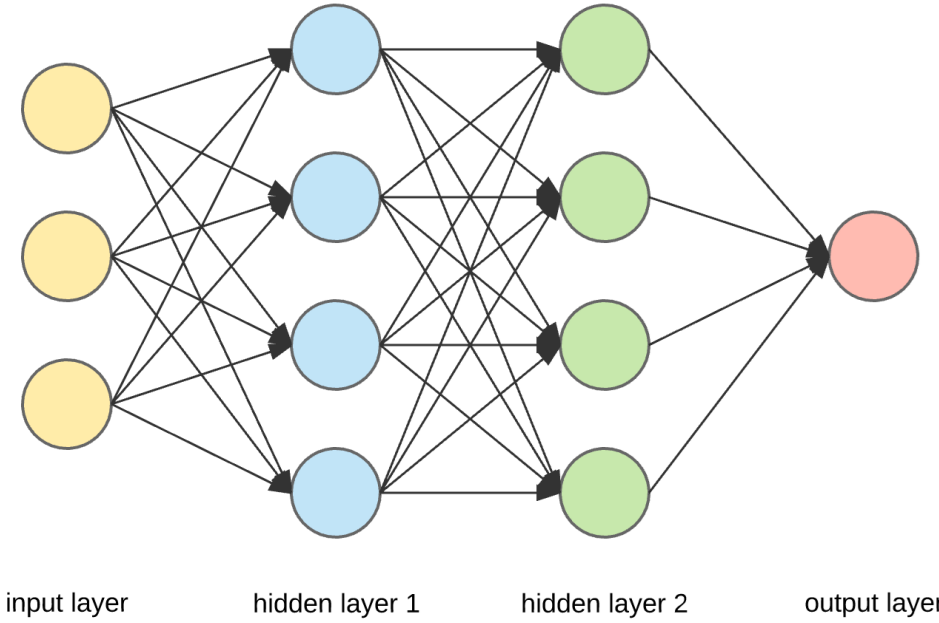
- Un singur neuron nu va fi suficient pentru a învăța sisteme complicate
- Se poate dezvolta ideea unui neuron unic, pentru crearea unei rețele neuronale multi-strat
- Pentru a construi o rețea de neuroni, putem lega straturi de neuroni simpli

Rețele de neuroni



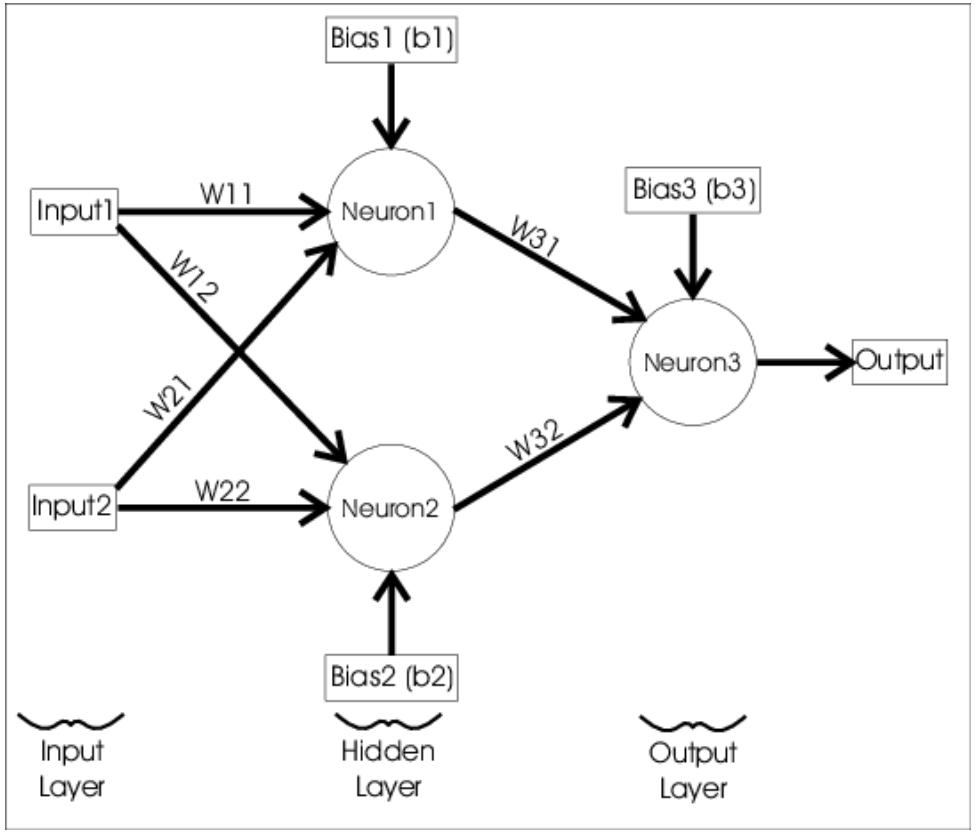
- Ieșirile unui perceptron sunt introduse direct în perceptronii de pe stratul următor, ca intrări ale acestuia

Rețele de neuroni



Rețelele de neuroni devin rețele de neuroni profunde (adânci) dacă acestea conțin două sau mai multe straturi ascunse

Rețele de neuroni superficiale (shallow)

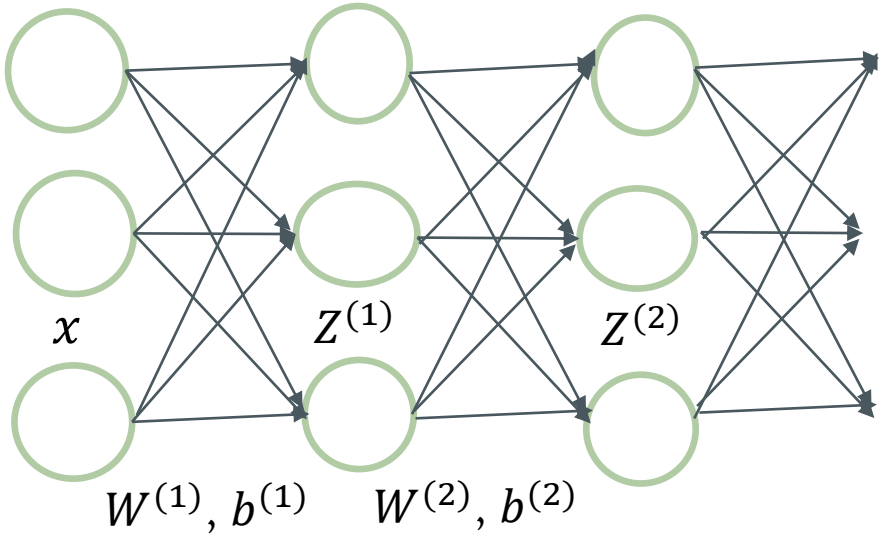


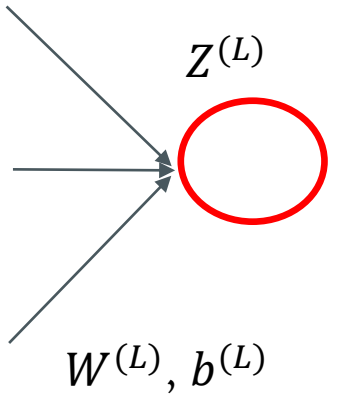
- Neuron 1= w11 x input1+w21 x input2+b1
- Neuron 2= w12 x input1+w22 x input2+b2
- Neuron 3 =w31xNeuron1+w32 x Neuron2+b3

$$\begin{pmatrix} Neuron\ 1 \\ Neuron\ 2 \end{pmatrix} = \begin{pmatrix} w11 & w21 \\ w12 & w22 \end{pmatrix} \times \begin{pmatrix} input\ 1 \\ input\ 2 \end{pmatrix} + \begin{pmatrix} b1 \\ b2 \end{pmatrix}$$

Rețele de neuroni – cum funcționează?

Fie $Z^{(i)}$, vectorul de ieșire al stratului ascuns i, de dimensiune M_i





- $Z^{(1)} = \sigma\left(W^{(1)T}x + b^{(1)}\right)$
- $Z^{(2)} = \sigma\left(W^{(2)T}Z^{(1)} + b^{(2)}\right)$
- $Z^{(3)} = \sigma\left(W^{(3)T}Z^{(2)} + b^{(3)}\right)$

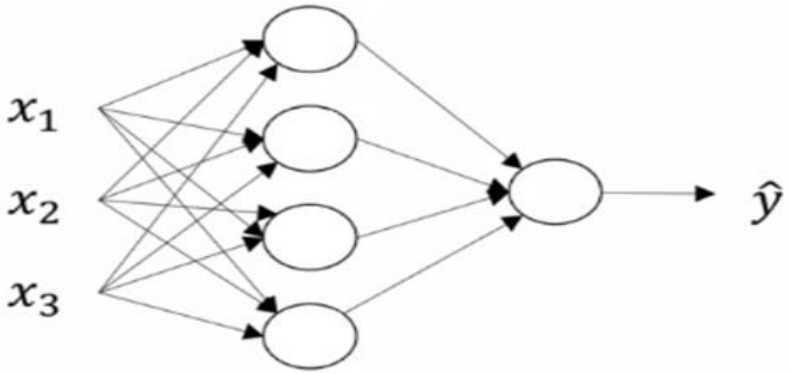
- $Z^{(L)} = \sigma\left(W^{(L)T}Z^{(L-1)} + b^{(L)}\right)$

- $b^{(i)}$ este un vector având dimensiunea lui $Z^{(i)}$
- $W^{(1)T}$ este o matrice de dimensiune M_i * dimensiunea lui x
- $W^{(i)T}$este o matrice de dimensiune $M_i * M_{i-1}$, i >2

Rețele de neuroni – cum învață?

- Cum învață rețelele, cum actualizează ponderile și deplasarea pentru a îmbunătăți performanța?
- Înțelegem că rețelele neuronale iau intrările, le multiplică prin ponderi și adaugă deplasarea, apoi rezultatul este transmis unei funcții de activare care, la finalul parcurgerii straturilor, conduce la ieșire.

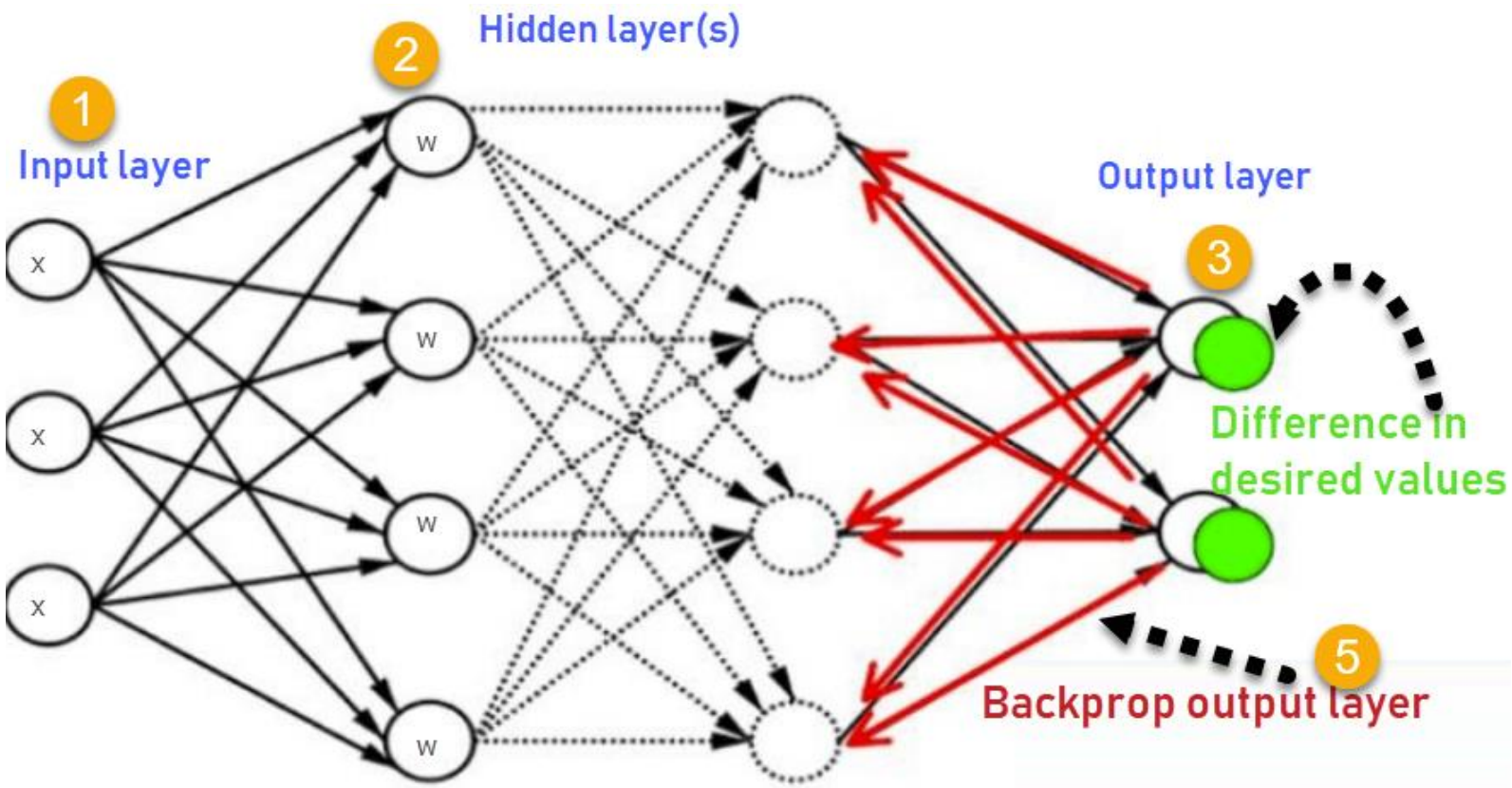
$$z_1^{[1]} = w_1^{[1]T} x + b_1^{[1]}, a_1^{[1]} = \sigma\left(z_1^{[1]}\right)$$
$$z_2^{[1]} = w_2^{[1]T} x + b_2^{[1]}, a_2^{[1]} = \sigma\left(z_2^{[1]}\right)$$
$$z_3^{[1]} = w_3^{[1]T} x + b_3^{[1]}, a_3^{[1]} = \sigma\left(z_3^{[1]}\right)$$
$$z_4^{[1]} = w_4^{[1]T} x + b_4^{[1]}, a_4^{[1]} = \sigma\left(z_4^{[1]}\right)$$



Rețele de neuroni – cum învață?

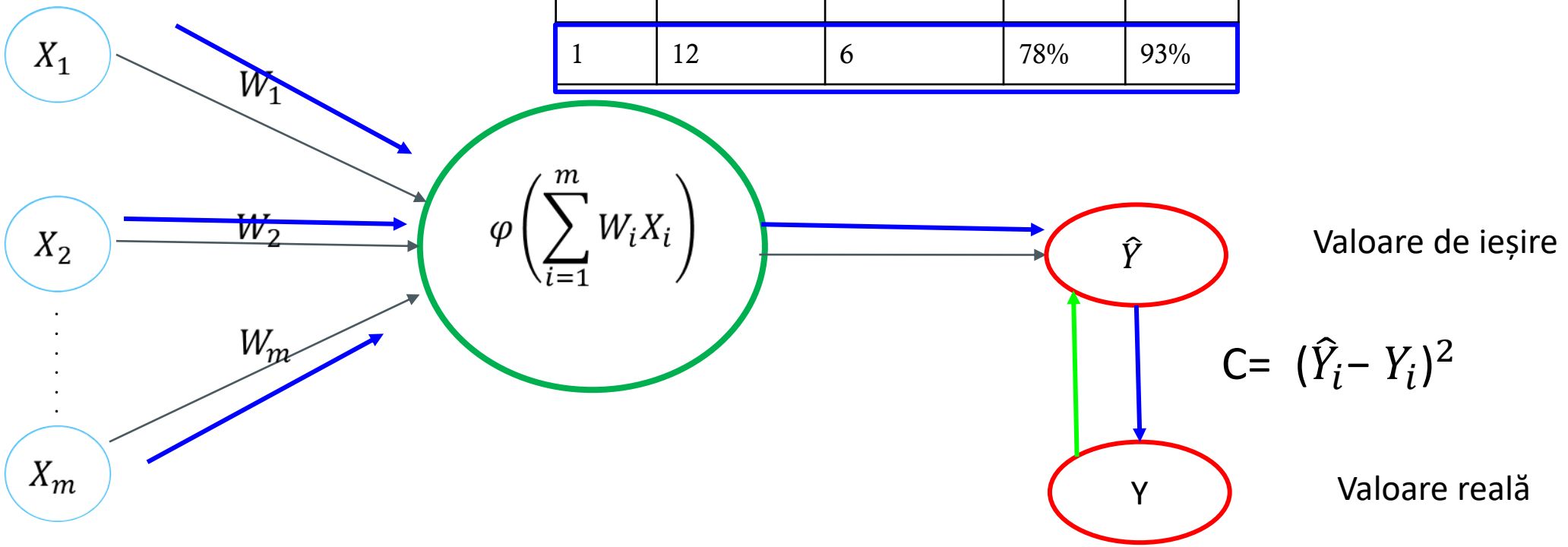
- Trebuie să luăm rezultatele estimate de către rețele și să le comparăm cu valorile reale (ale țintei) cu scopul de a ajusta ponderile.
- De reținut că ajustarea modelului constă în utilizarea ansamblului de date.

Retropropagare



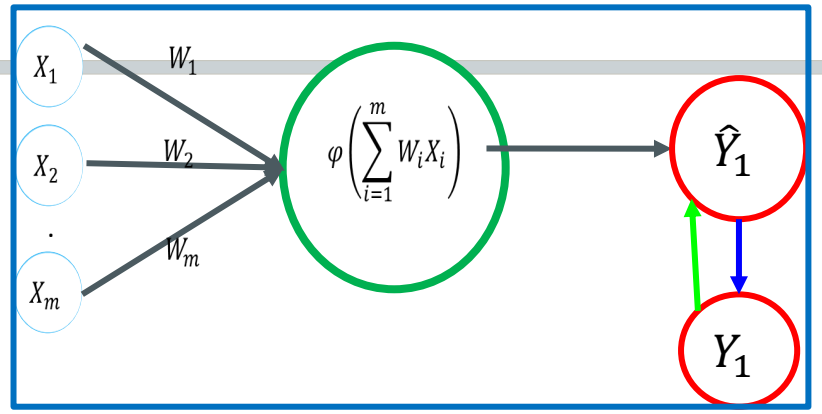
Funcția cost

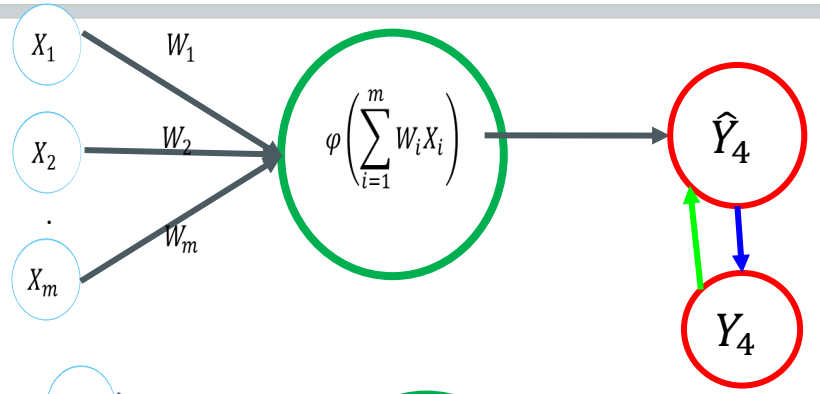
id	study hrs	sleep hrs	QuiZ	Exam
1	12	6	78%	93%

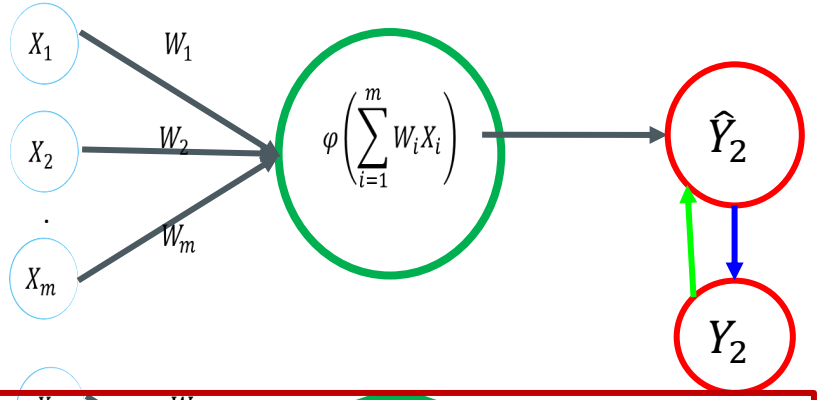


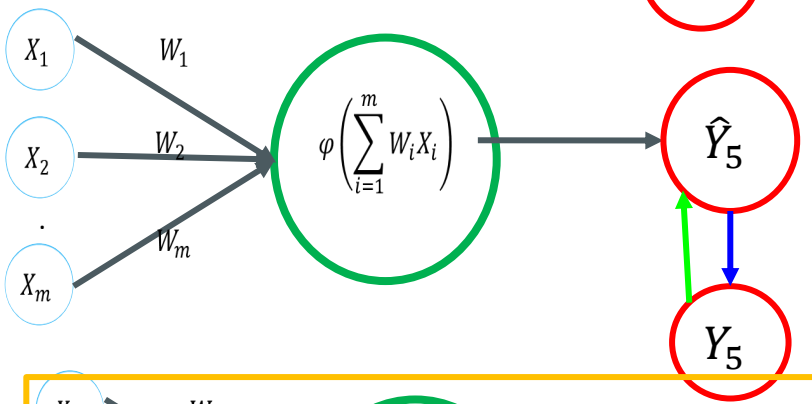
O funcție de cost foarte comună este funcția cuadratică.

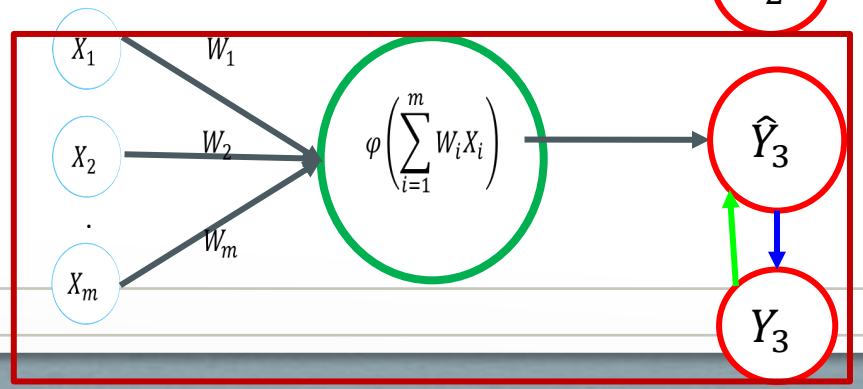
Învățare

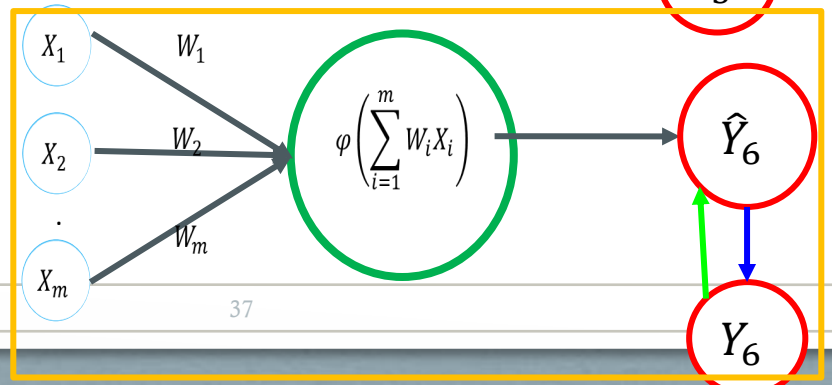












row id	study hrs	slee hrs	Quiz	Exam
1	12	6	73%	93%
2	22	6.5	24%	68%
3	115	4	10%	95%
4	31	9	67%	75%
5	0	10	58%	51%
6	57	8	91%	97%

Retropropagare (*Backpropagation*)

$$C = \frac{1}{n} \sum_{i=1}^{n} (\hat{Y}_i - Y_i)^2$$

Y	85	95	66	56	95	Cost
$\widehat{Y}$	55	97	99	63	61	37

Ajustăm W_1, W_2 , W_3.....

Actualizarea ponderilor minimizând funcția de cost C

- În timpul procesului de învățare dorim să minimizăm funcția de cost
- Pentru aceasta, ne bazăm pe algoritmul de coborâre a gradientului

Învățare – funcțiile de cost

- Pentru modele de clasificare:
  - *Categorical Cross Entropy*
  - *Binary Cross Entropy*

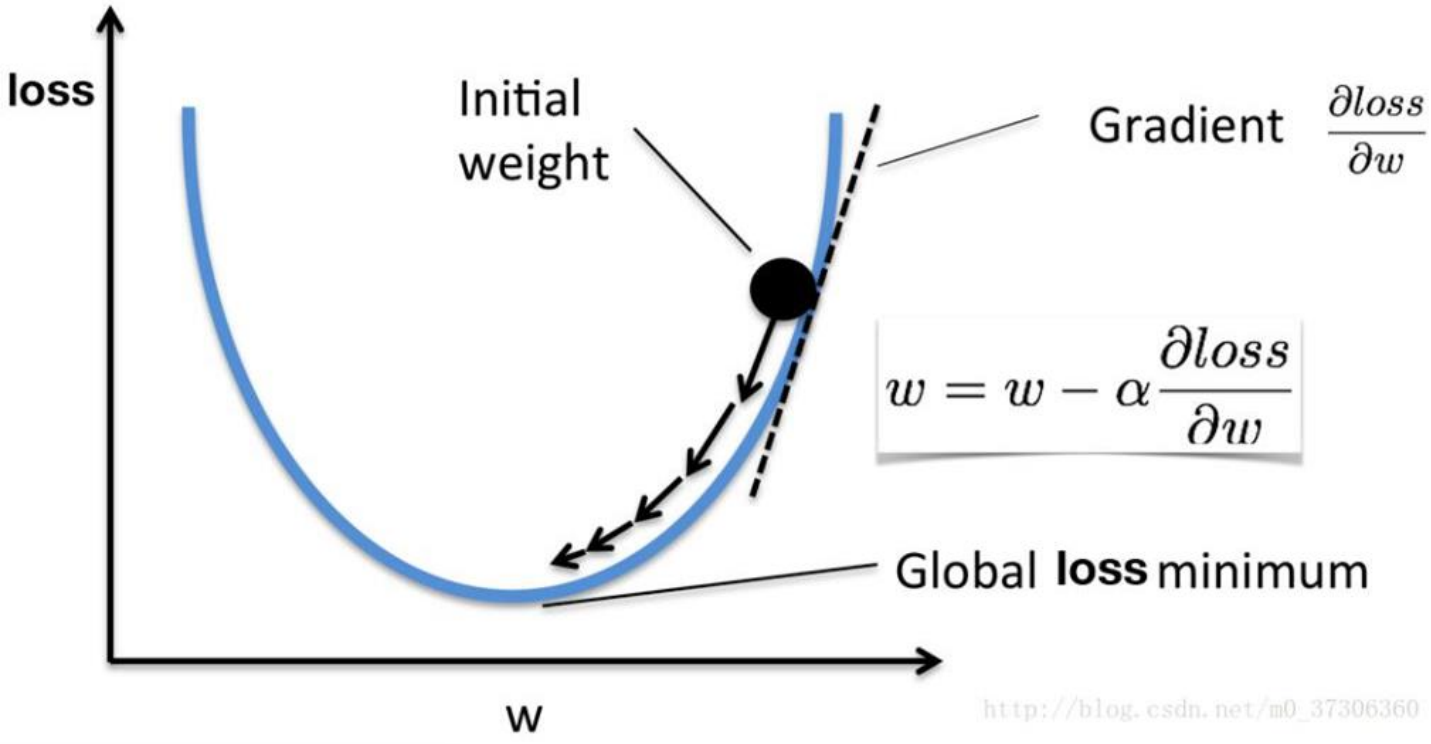
$$H(p, q) = -\sum_{x \in \mathcal{X}} p(x) \log q(x)$$

$$H_p(q) = -\frac{1}{N} \sum_{i=1}^{N} y_i \cdot log(p(y_i)) + (1 - y_i) \cdot log(1 - p(y_i))$$

- Pentru modele de regresie:
  - *Mean Error* (ME)
  - *Mean Absolute Error* (MAE)
  - *Mean square Error* (MSE)
  - *Root Mean Square Error* (RMSE)

Algoritmul de coborâre a gradientului

- *Gradient Descent*: algoritm de optimizare iterativ pentru găsirea minimului local al unei funcții



Gradient descent algorithm

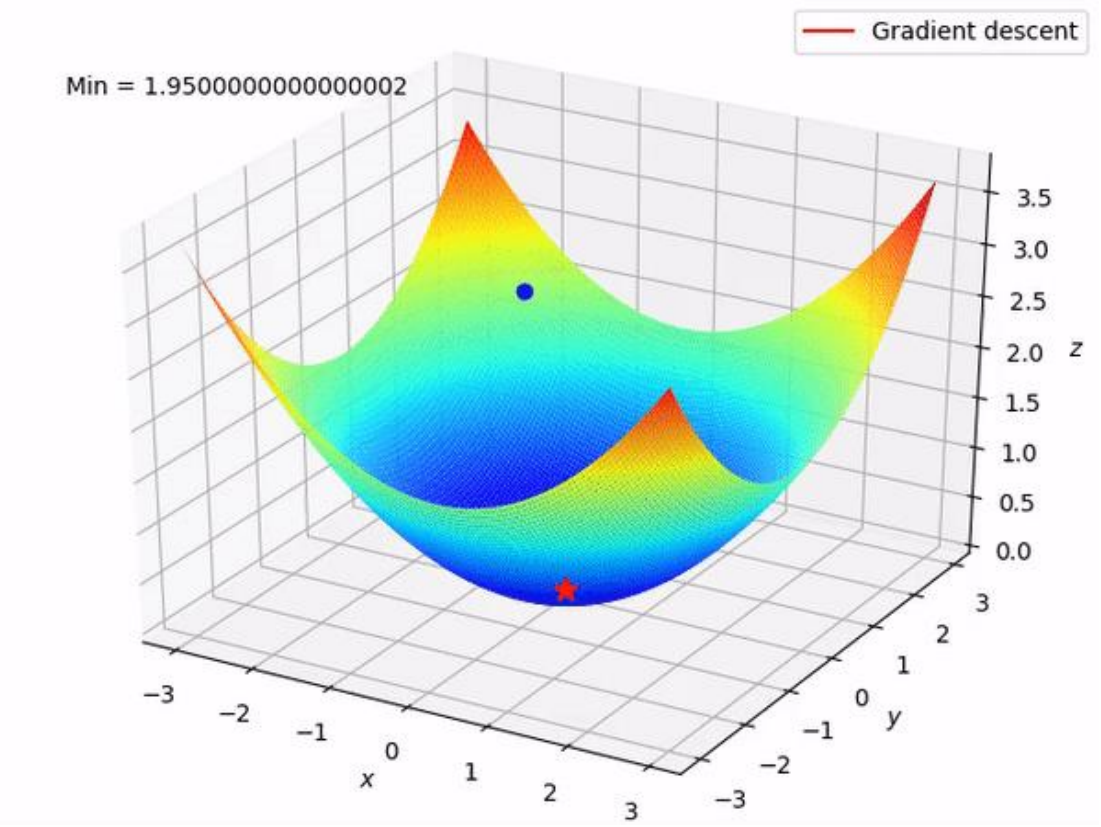
repeat until convergence {

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$$

(for $j = 1$ and $j = 0$)

}

Gradient Descent pentru modelul liniar



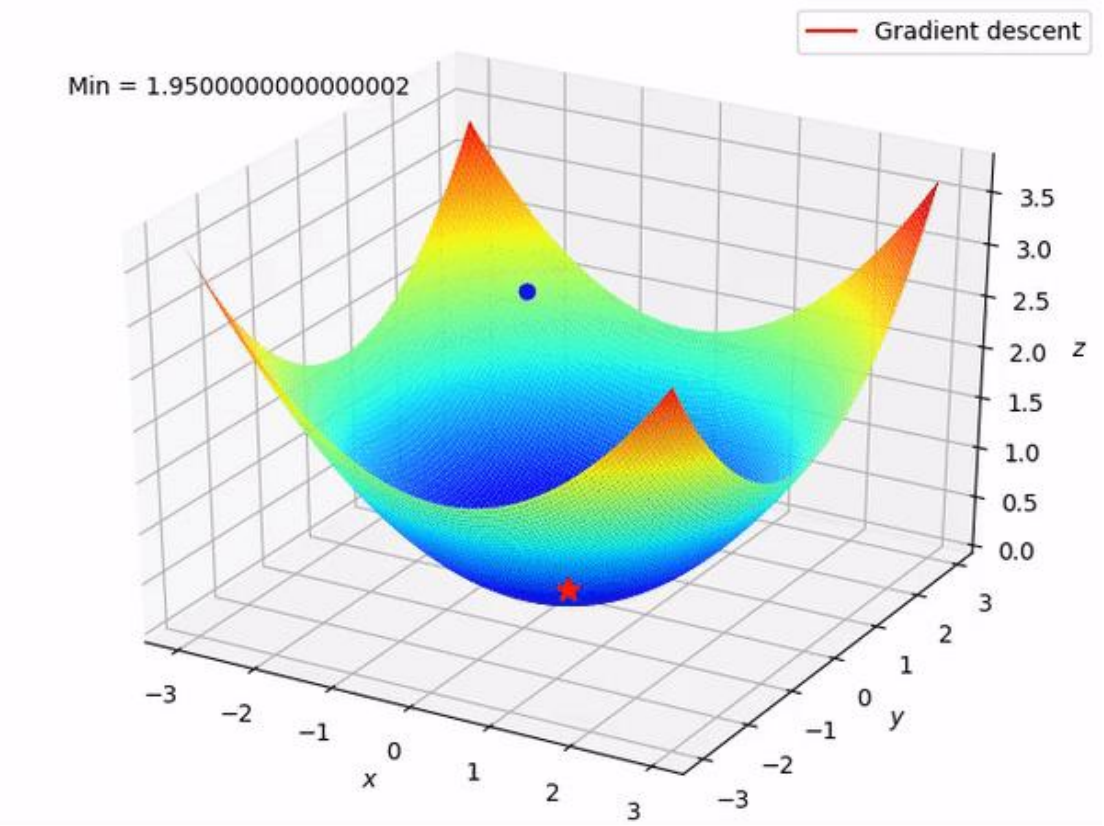
Hypothesis: $h_\theta(x) = \theta_0 + \theta_1 x$

Parameters: θ_0, θ_1

Cost Function: $J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2$

Goal: $\underset{\theta_0, \theta_1}{\text{minimize}}\ J(\theta_0, \theta_1)$

Gradient Descent



Gradient descent algorithm

repeat until convergence {

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$$

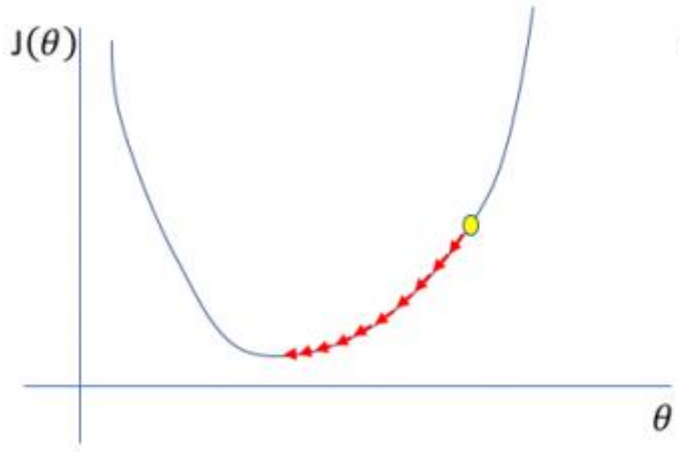
$$(\text{for } j = 1 \text{ and } j = 0)$$

}

α = Rata de învățare

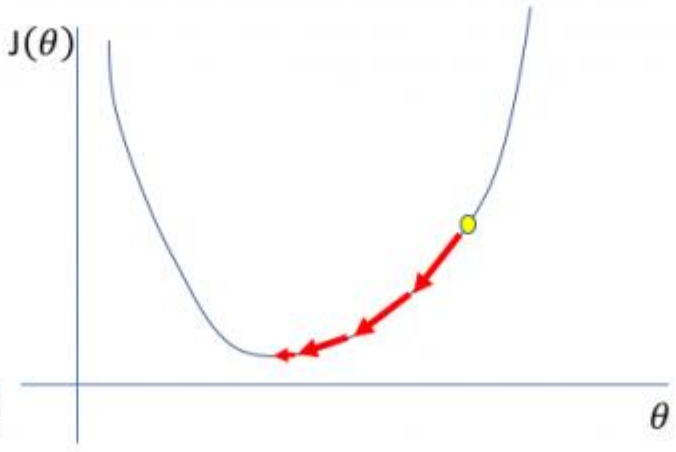
Rata de învățare

rată mică



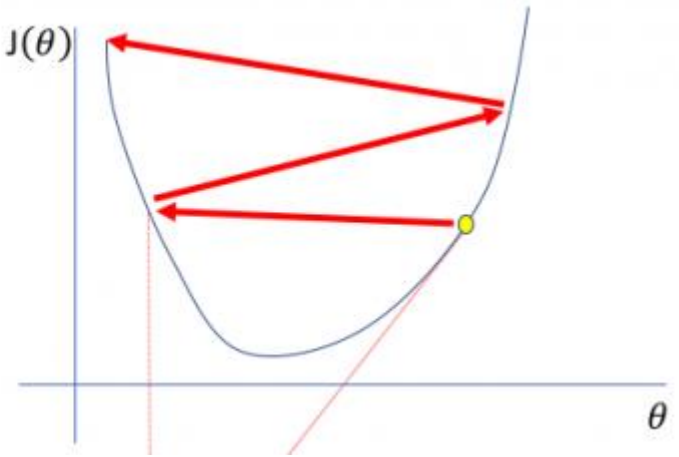
O rată mică de învățare necesită actualizări numeroase înainte de a ajunge la punctul de minim

rată optimă optimal



Rata de învățare optimă ajunge rapid la minimum

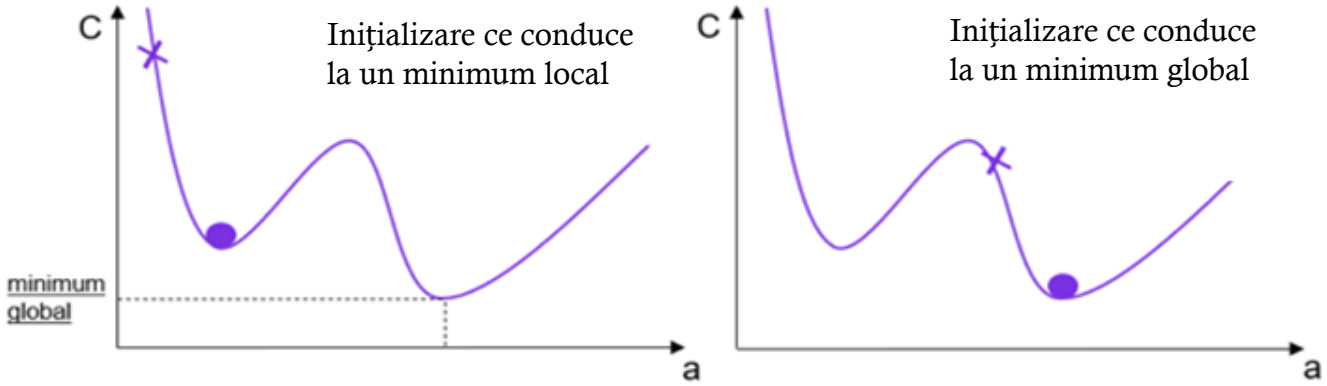
rată foarte mare



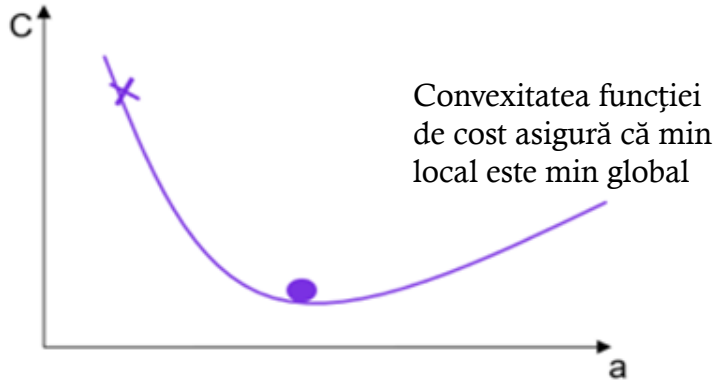
O rată de învățare foarte mare conduce la actualizări drastice, ce conduc la comportamente divergente

Gradient Descent

- Problemă: Gradient Descend nu garantează un minim global dacă funcția de cost nu este convexă
  - Cazul unei funcţii de cost non-convexe



  - Cazul unei funcţii de cost convexe



Gradient Descent Stochastic (SGD)

- Pentru un set mare de date, calculul gradientului funcţiei devine foarte costisitor

Algoritmul GD:

$$\theta_{j+1} = \theta_j - \alpha \frac{\partial}{\partial \theta_j} (\sum_{i=1}^{n} (h_\theta(x^{(i)}) - y^{(i)})^2)$$

$$\text{for } j = 0, 1, 2, ...n\}$$

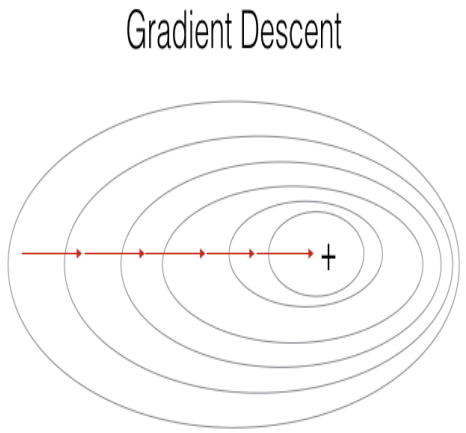
- SGD pentru accelerarea învăţării:

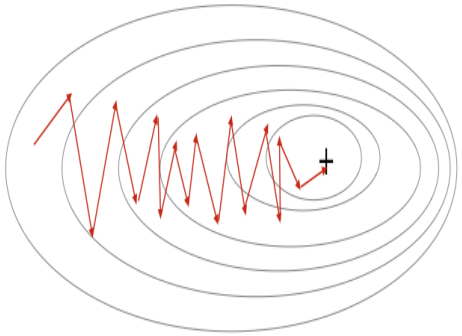
Algoritmul SGD:

$$\{ \text{ for arbitrary i=0,1, ..., m\{}$$

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} (h_\theta(x^{(i)}) - y^{(i)}))^2$$

$$\text{for } j = 0, 1, 2, ...n\}$$



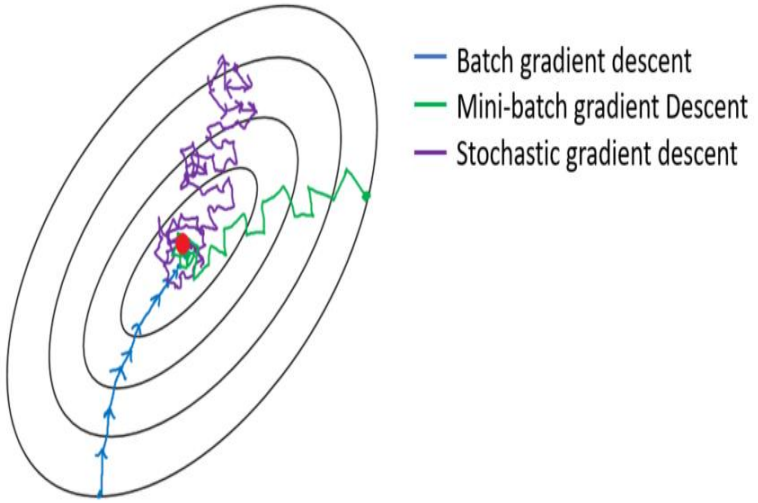


Mini-batch GD

- Gradient Descent: Parametrii sunt actualizați după calculul gradientului de eroare în funcție de setul de date

- Stochastic Gradient Descent: Parametrii sunt actualizați după calculul gradientului de eroare relativ la un singur exemplu din dataset

- Mini-Batch Gardient Descent: Parametrii sunt actualizați după calculul gradientului de eroare relativ la o submulțime a dataset-ului



Algoritm Mini-batch GD:

{ for arbitrary i=1, B+1, 2B ....., m{

$$\{\theta_{j+1} = \theta_j - \alpha \frac{\partial}{\partial \theta_j} (\sum_{k=i}^{i+B-1} (h_\theta(x^{(i)}) - y^{(i)})^2)$$

for $j = 0, 1, 2, ...n\}$

Gradient Descent

- Pentru accelerarea învățării și pentru îmbunătățirea performanțelor se pot utiliza optimizori foarte puternici:

❖ Gradient Descent

❖ SGD

❖ SGD + Momentum

❖ NAG (Nestrov Accelerate Gradient)

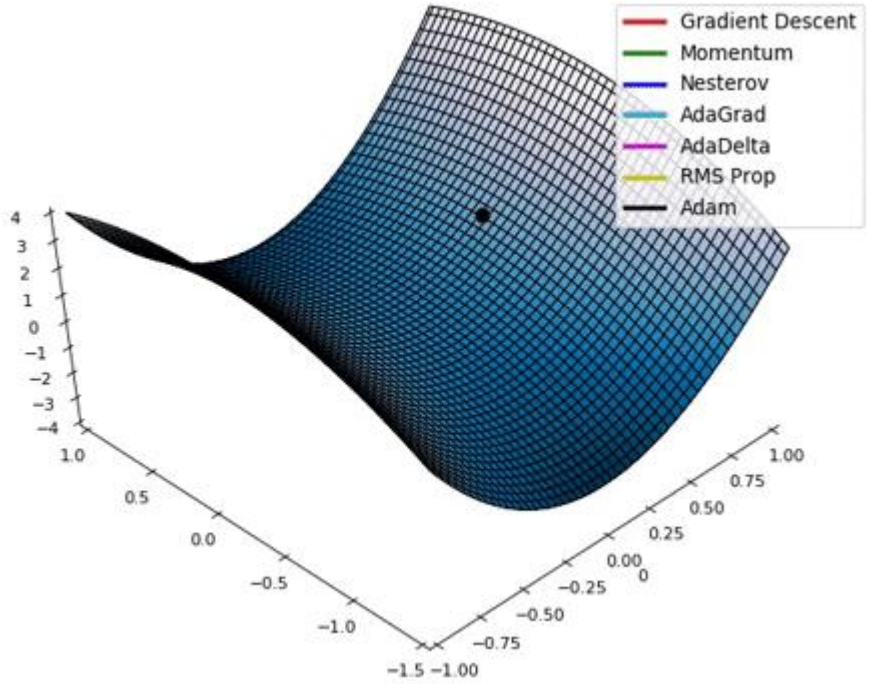
❖ Rprop (Resilient Propagation)

❖ AdaGrad (Adaptive Gradient)

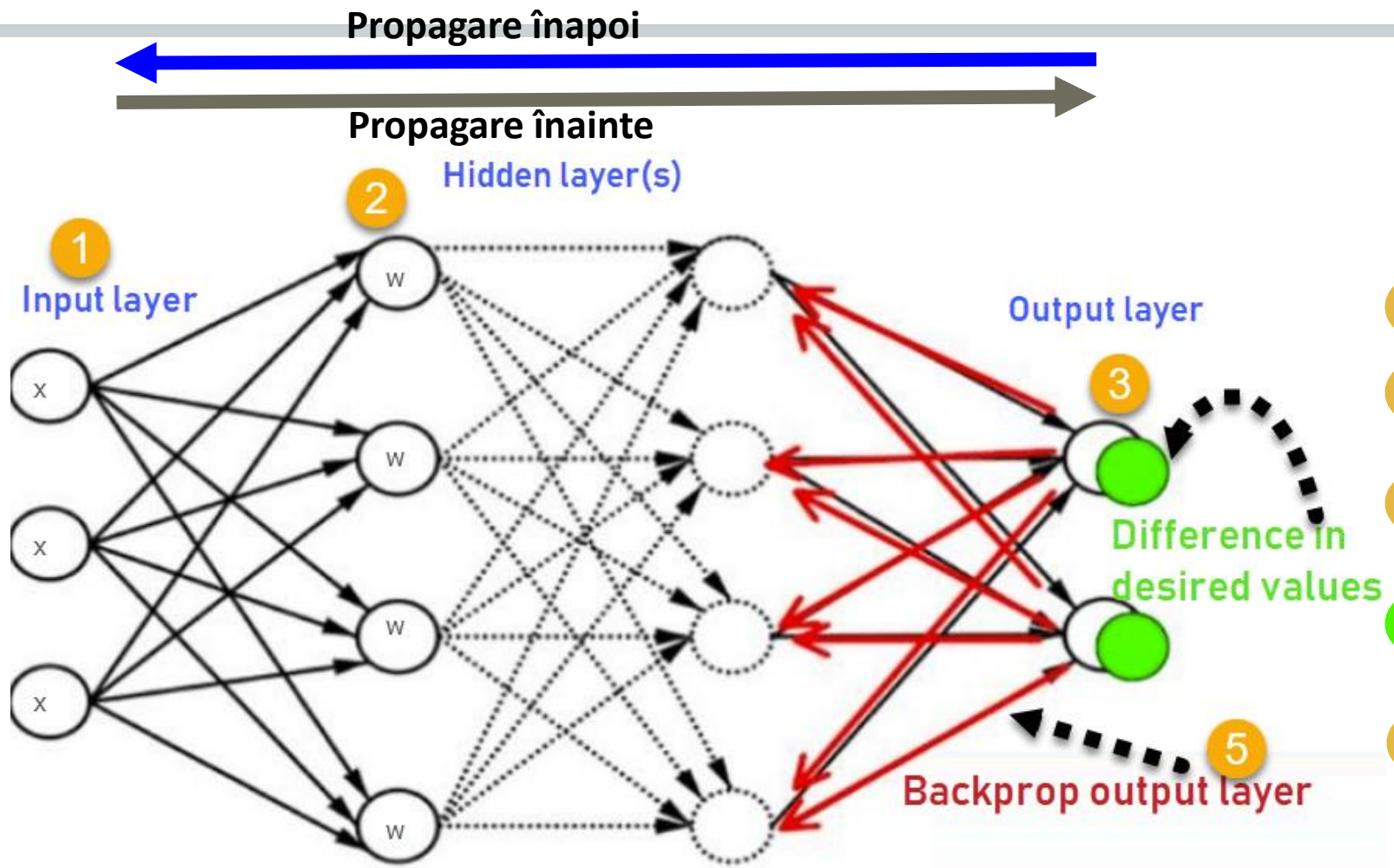
❖ RMSProp = Rprop+SGD

❖ AdaDelta

❖ Adam (Adaptive moment estimation)



Rezumat



1. Date de intrare
2. Combinări și activare
3. Date de ieșire
4.
5. Retropropagare și minimizare

Gradient Descent

- Demo GD

Construcția unui ANN

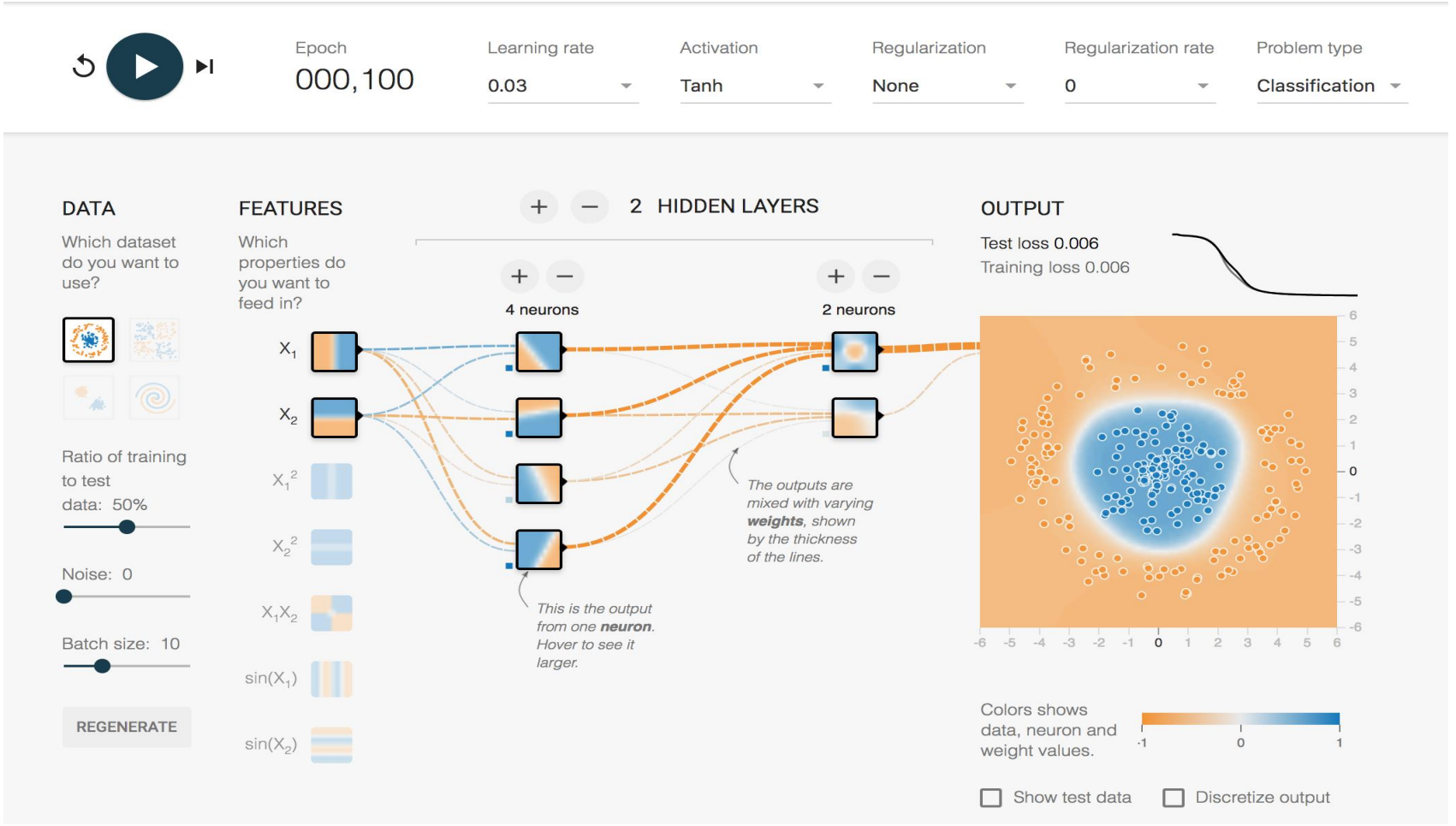
Construcția unui ANN - Hiperparametri

- ANN = Artificial Neural Network
- Ce funcție de activare vom folosi?
- Ce rată de învățare?
- Câte straturi ascunse?
- Care este numărul de epoci (*epochs*)?

Hiperparametri

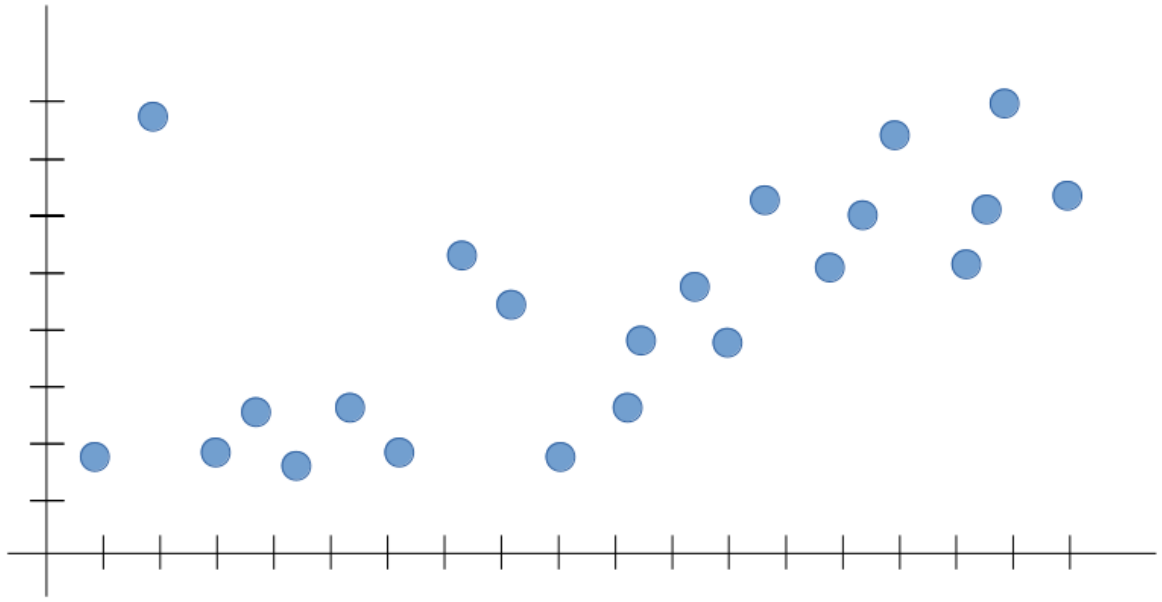
- Cu ajutorul PlaygroundTesorflow se poate descoperi efectul diferitelor opțiuni:
  - Funcția de activare
  - Rata de învățare
  - Numărul de straturi ascunse

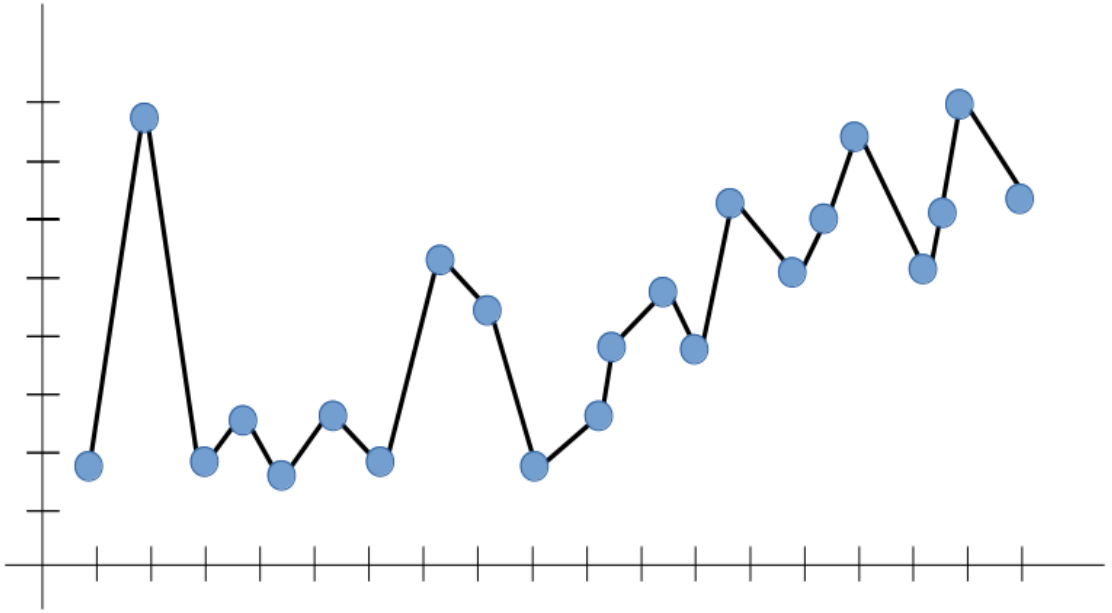
Playground Tensorflow



Overfitting

- Modelul se adaptează prea mult zgomotului ansamblului de date
- Se găsesc adesea erori slabe pe datele de antrenare, dar erori mari pe datele de test



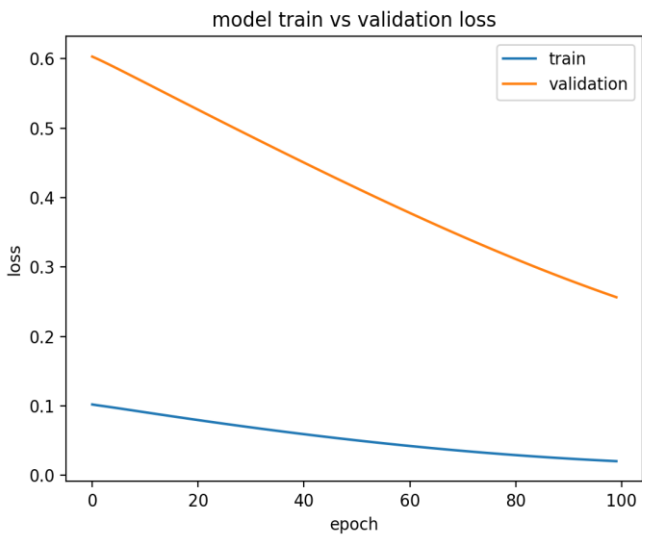


Underfitting

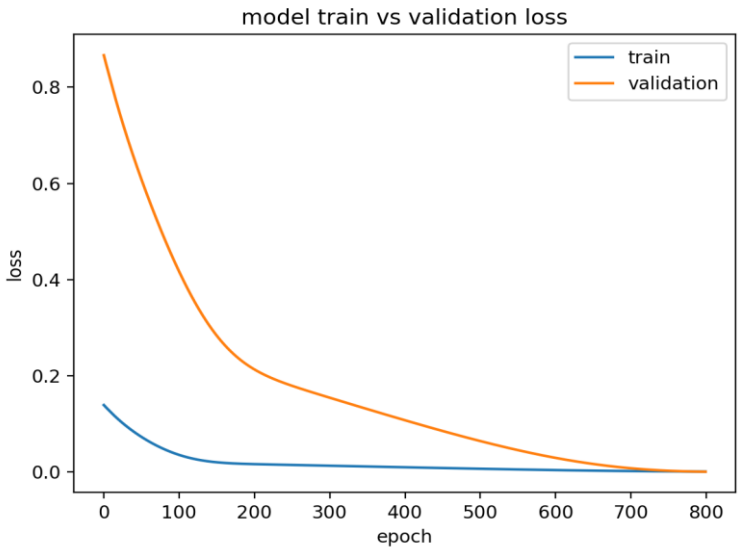
- Modelul nu sesizează tendința subiacentă a datelor și nu corespunde prea bine datelor
- Varianță slabă și deplasare mare
- Underfitting-ul este adesea rezultatul unui model excesiv de simplu.

Funcția de cost

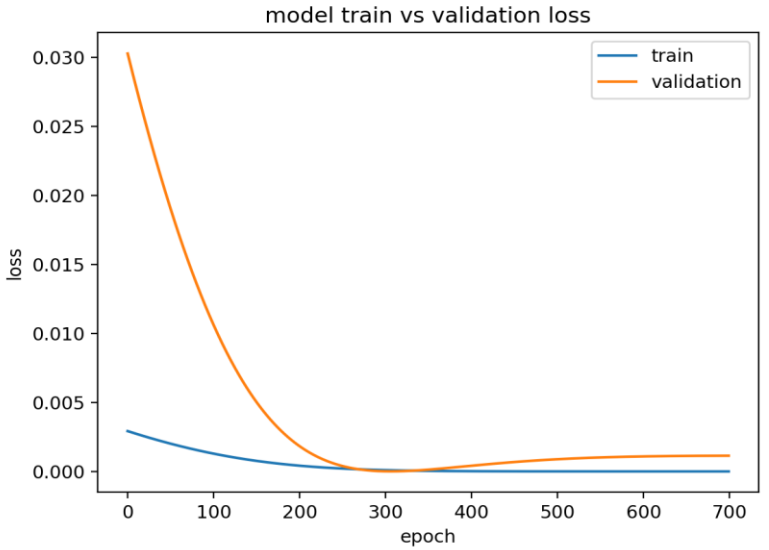
- În exemplul precedent, datele sunt ușor de vizualizat , dar cum putem observa overfitting-ul sau underfitting-ul atunci când avem mulțimi de date multidimensionale?
- Presupunem că am format un model și i-am măsurat eroarea pe durata antrenamentului:



Underfitting



Model bun



Overfitting

Overfitting/Underfitting – Soluții

- Overfitting:
  - Renunțarea la o parte a datelor (*dropout*) pentru a evita ca modelul să depindă prea mult de o singură intrare, deoarece aceasta va putea dispărea
  - Augmentarea datelor pentru a diversifica mulțimea datelor de antrenament
  - *Early Stopping*

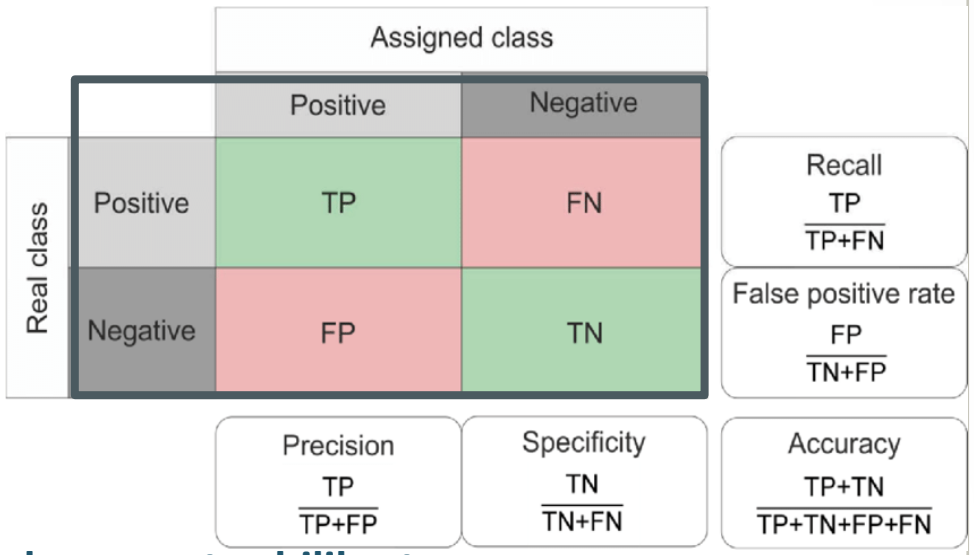
- Underfitting:
  - Adăugarea de straturi ascunse
  - Augmentarea numărului de neuroni poate ajuta în depășirea problemei
  - Augmentarea timpului de învățare, fiind posibil ca un model neperformant să nu fi găsit încă valorile optimale ale parametrilor săi.

Evaluare – Modele de clasificare

- În mod obișnuit, după procesul de învățare se vor utiliza metrici de performanță pentru evaluarea performanțelor modelului.

Confusion Matrix

Acuratețea: Raportul dintre numărul de predicții corecte realizate de către model și numărul total de predicții



Acuratețea nu este o alegere bună, ca metrică, atunci când datele nu sunt echilibrate

Exemplu: Pentru o bază de date cu 99 de imagini de pisici și un singur câine, dacă toate predicțiile modelului sunt pisici vom avea o acuratețe de 99%, care de fapt nu corespunde performanței modelului.

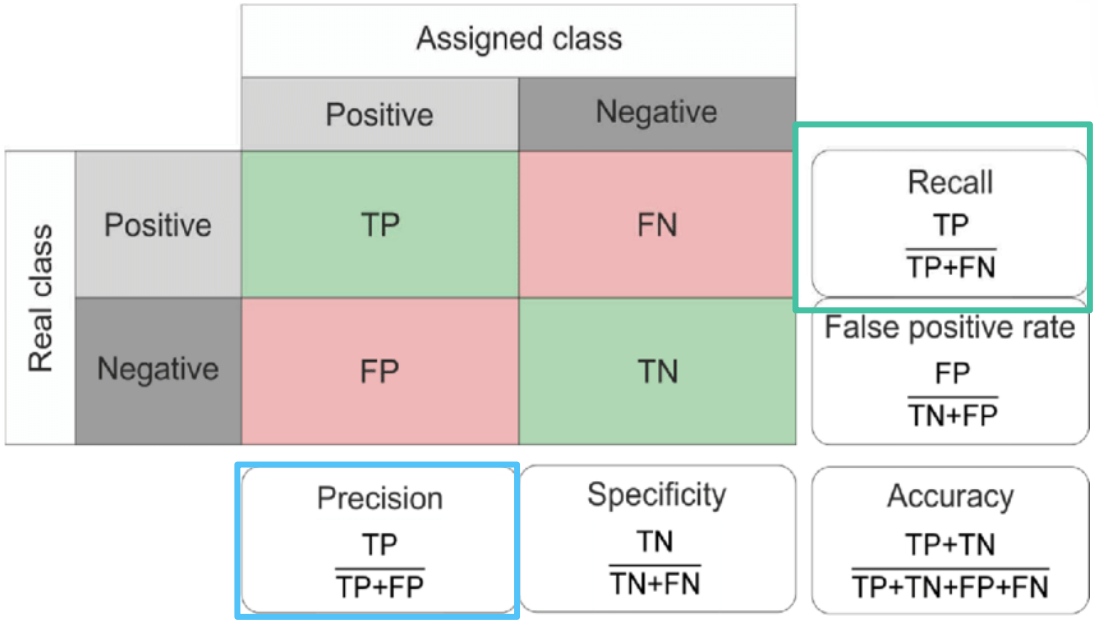
Evaluare – Modele de clasificare

***Recall* :** numărul de *true positives*/(numărul de *true positives* + numărul de *false negatives*)

Precizie: numărul de *true positives* / (numărul de *true positives* + numărul de *false positives*)

F1-Score: Media armonică dintre recall și precizie:

F1-Score: 2*(Precision*Recall)/(Precision+Recall)



Evaluare – Modele de regresie

- Metricile cele mai populare pentru regresie sunt:
  - MAE : Mean Absolute Error : : $\frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$
  - MSE : Mean Squared Error : $\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$
  - RMSE: Root Mean Squared Error : $\sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$

Evaluare – Modele de regresie

- Care este cea mai bună metrică?
  - Depinde de situaţie şi de domeniu!
- Compararea măsurii de eroare cu media etichetelor (valorilor căutate) în mulţimea de date pentru a obține o intuiție asupra performanței globale
- Cunoașterea domeniului joacă un rol important.

Evaluare – Învățarea nesupervizată

- În învățarea nesupervizată nu avem date istorice etichetate
  - Clustering
  - Detectarea anomaliilor
  - Reducerea dimensionalității

- În aceste situații nu avem răspunsul corect pentru datele istorice, ceea ce înseamnă că evaluarea este mult mai dificilă și nuanțată

Construirea unui ANN: TensorFlow și Keras

- TensorFlow posedă un ecosistem vast de componente conexe, inclusiv biblioteci precum tensorBoard, API-uri de deployment și de producție, disponibile în mai multe limbaje
- Keras: Bibliotecă open-source pentru rețele neuronale scrisă în Python, ce funcționează sub Theano sau Tensorflow
  - Este concepută să fie modulară, rapidă și ușor de utilizat
  - Utilă pentru a construi orice algoritm de deep learning
  - Este separată de TF1
  - Odată cu lansarea TF2, TF a adoptat Keras ca API oficial pentru TF (nu este nevoie de o instalare separată)

Construirea unui ANN – Demers

ETAPA 1 : Pregătirea datelor : Pandas, Seaborn, sk-learn……

ETAPE 2 : Crearea modelului : Sequential(), Dense()

ETAPE 3 : Compilarea modelului: compile( ), optimizer, loss

ETPAE 4 : Antrenarea modelului: fit(), epochs,.....

ETAPE 5 : Evaluarea modelului : model.history.history (Overfitting), model.evaluation()

Implementare cu TensorFlow 2

Import de librării	`from tensorflow.keras.models import Sequential` `from tensorflow.keras.layers import Dense, Activation`
Inițializare ponderi	`model = Sequential()`

Adăugare de straturi ascunse	`model.add(Dense(4,activation='relu'))`
	`model.add(Dense(4,activation='relu'))`
	`model.add(Dense(4,activation='relu'))`
	`## 4 neuroni în fiecare strat`

Ultimul strat (Output)	`model.add(Dense(1))`
Compilarea modelului	`model.compile(optimizer='rmsprop',loss='mse')`
Antrenarea modelului	`model.fit(X_train,y_train,epochs=250)`

Construirea unui ANN

- Demo