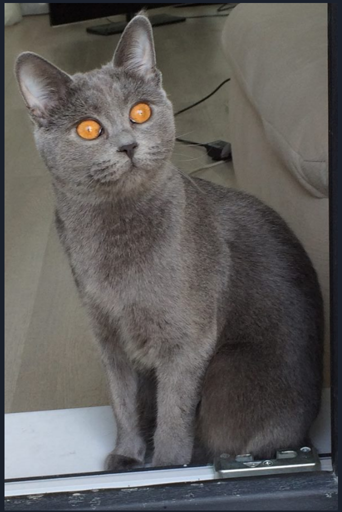
Invatare automata in arta vizuala

Clasificarea Imaginilor. Optimizare

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]
  ...,
  [135 140 134]
  [135 140 134]
  [135 140 134]]

 [[131 123 110]
  [125 117 104]
  [118 110  97]
  ...,
  [135 140 134]
  [134 139 133]
  [134 139 133]]]
```

Probabilitati peste clase discrete: catel, pisica, soarece ...

catel	0.2
tigru	0.3
pisica	0.4
soarece	0.1

Raspuns: pisica

Dificultati

Deformare

Ocluzie

Iluminare

Camuflaj

Varietate

- Nu avem o solutie programatica (if magic then cat else dog)
- Imaginile sunt matrici de numere (pixeli [0, 255])
- Orice schimbare de orientare schimba complet valorile acesteia

Abordarea parametrica

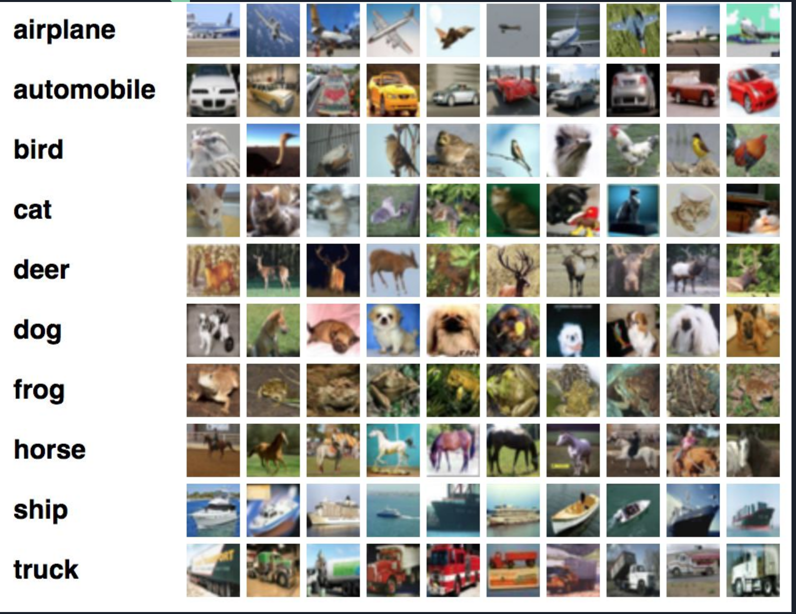
W sau θ = parametrii [parameters, weights]

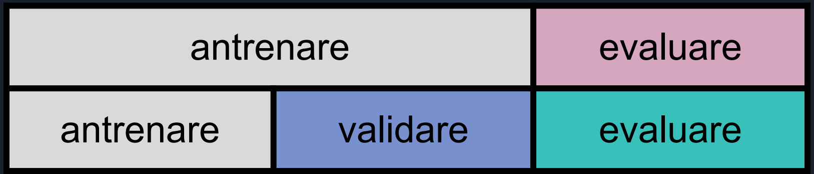
f(x; w)

Vector 32x32x3
(3072) elemente

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

Algoritmi de invatare din date





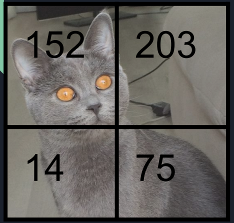
Dataset - perechi [imagine, eticheta]

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** pe imaginile de evaluare

Perceptronul



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

clasa	scor
pisica/not_pisica	123

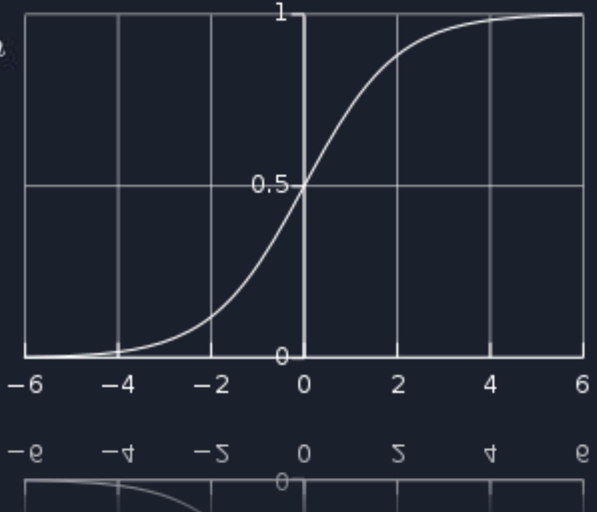
Se da o imagine x, vrem sa aflam $\hat{y} = P(y = 1|x), x \in \mathbf{R}^n$

$0 < \hat{y} \leq 1$

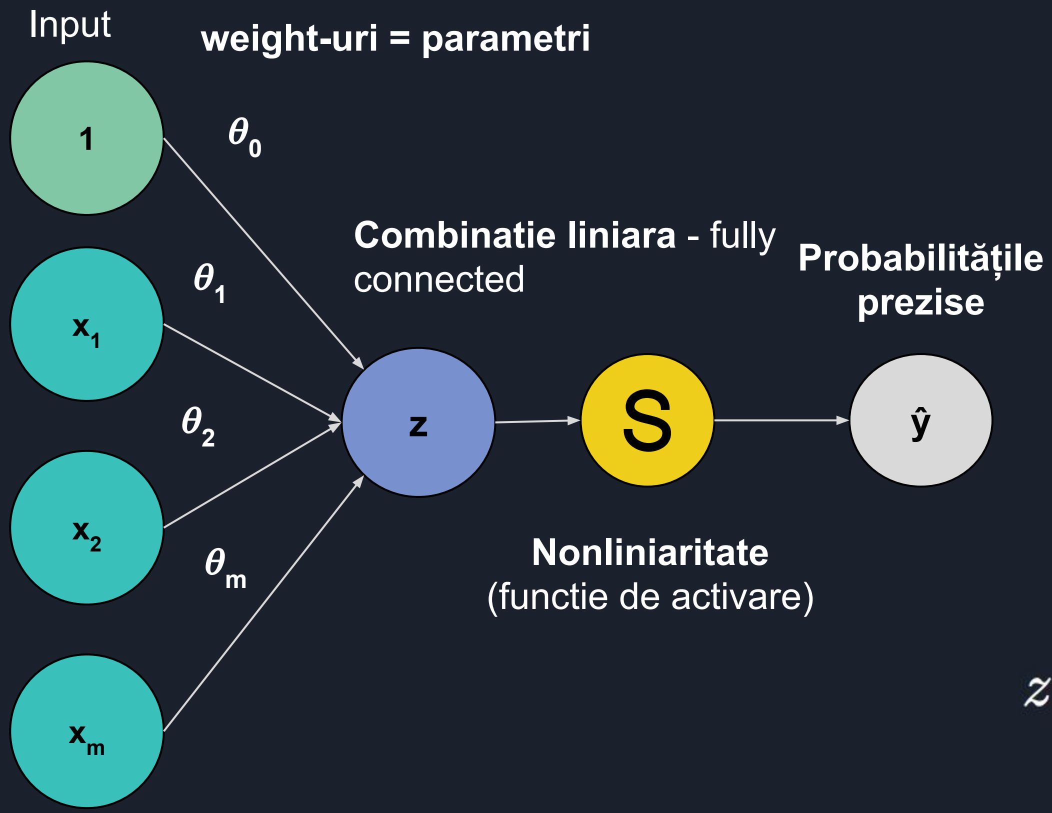
$Scor = W^T x + b$

$\hat{y} = \sigma(W^T x + b)$

Functia sigmoid $\sigma(z) = \frac{1}{1+e^{-z}}$

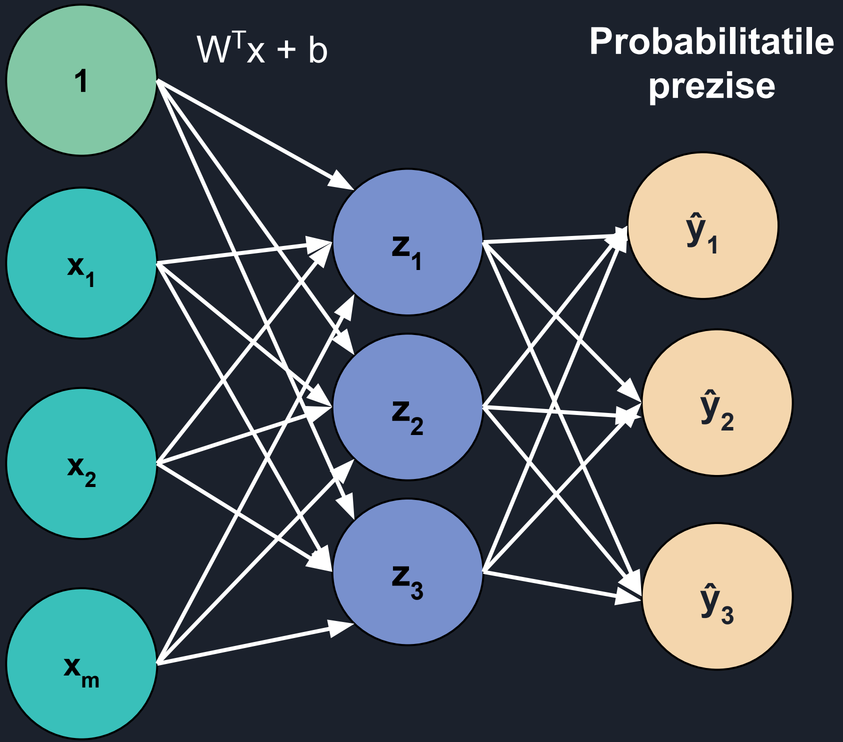


Perceptronul



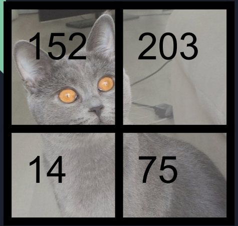
$$z = \theta_0 + \sum_{i=1}^{m} x_i \theta_i$$

Perceptron multi-iesire



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

Clasificare liniara



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

$$\begin{array}{ccccccc} W & & X & & b & & Scor \\ \begin{bmatrix} 0.2 & -0.5 & 0.1 & 0.2 \\ 0.4 & 0.3 & 0.1 & 0.0 \\ 0.0 & 0.25 & 0.2 & -0.3 \end{bmatrix} & * & \begin{bmatrix} 152 \\ 203 \\ 14 \\ 75 \end{bmatrix} & + & \begin{bmatrix} 0.01 \\ 0.03 \\ 0.09 \\ 0.02 \end{bmatrix} & = & \begin{bmatrix} -54.7 \\ 123.1 \\ 31.05 \end{bmatrix} \end{array}$$

Multinomial Logistic regression

- **Scorurile = 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_j = componenta i a scorurilor (scorul clasei j)

```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))
```

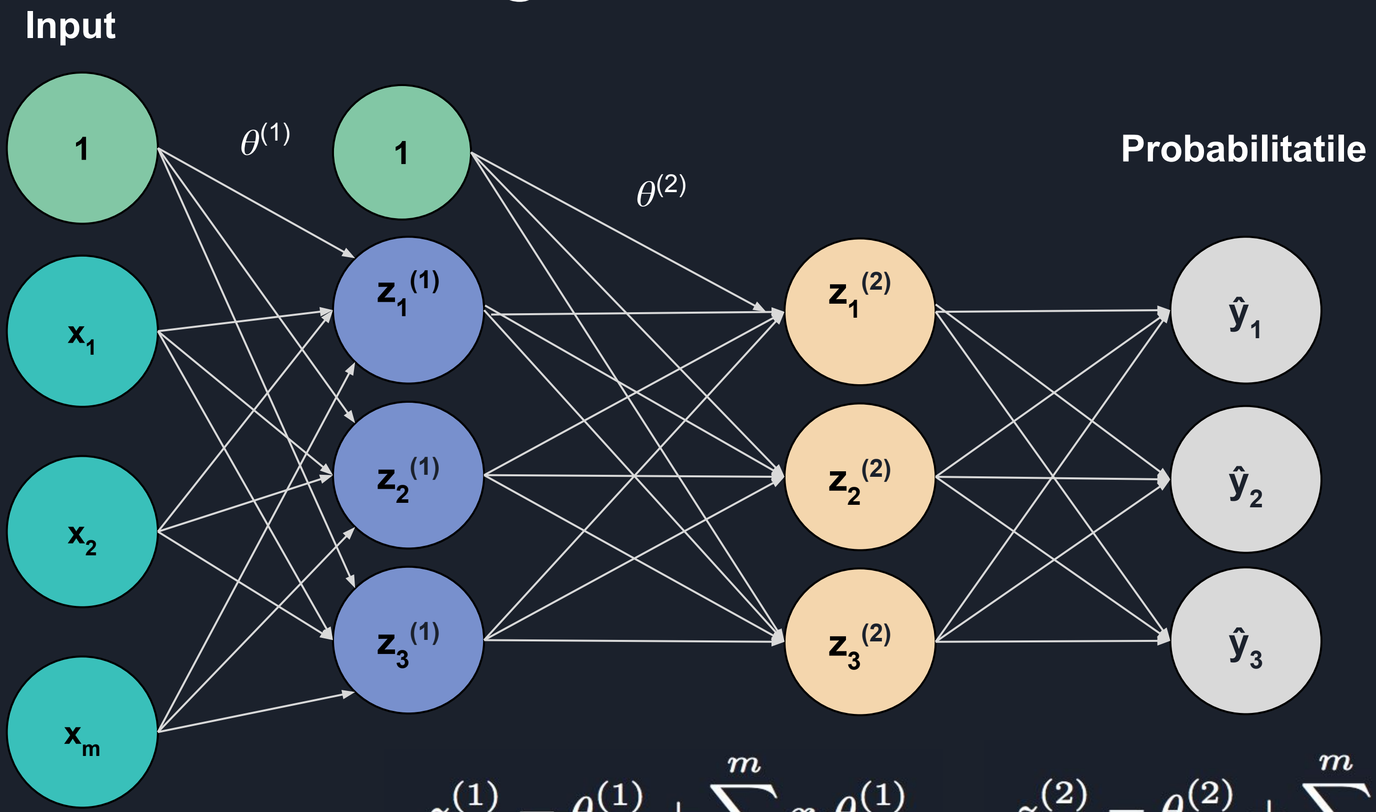
```
[ 0.8360188   0.11314284  0.05083836]
```

Numerically stable

```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."""
    e_x = np.exp(x - np.max(x))
    return e_x / e_x.sum()


print(softmax(scores))
```

```
[ 0.8360188   0.11314284  0.05083836]
```

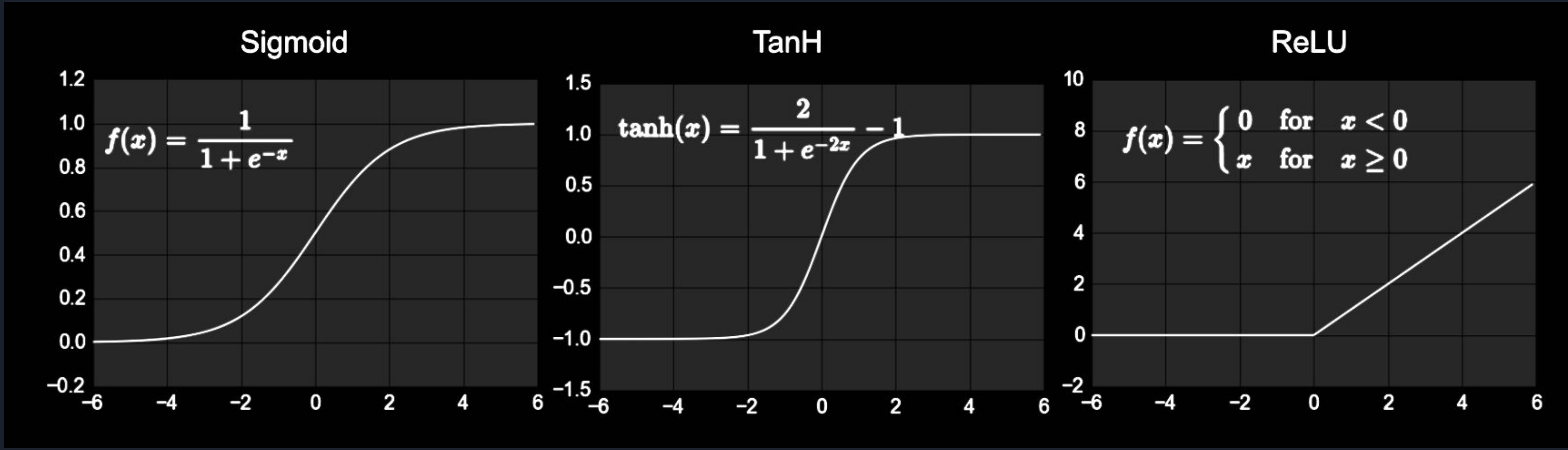
Retea cu un singur nivel



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

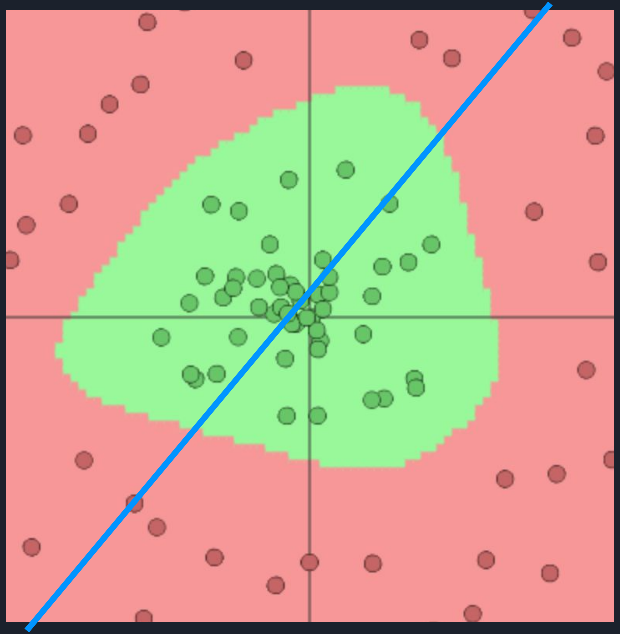
g = functie de activare(nonliniaritate)

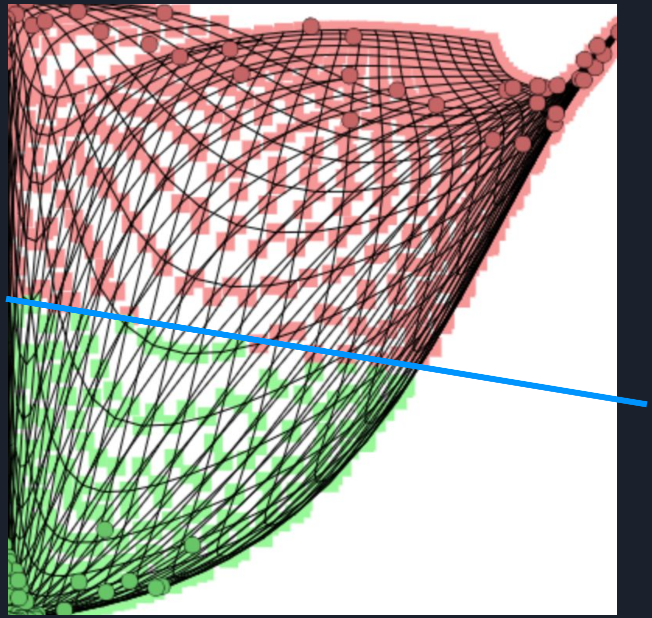
Functii de activare



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

Importanta functiilor de activare

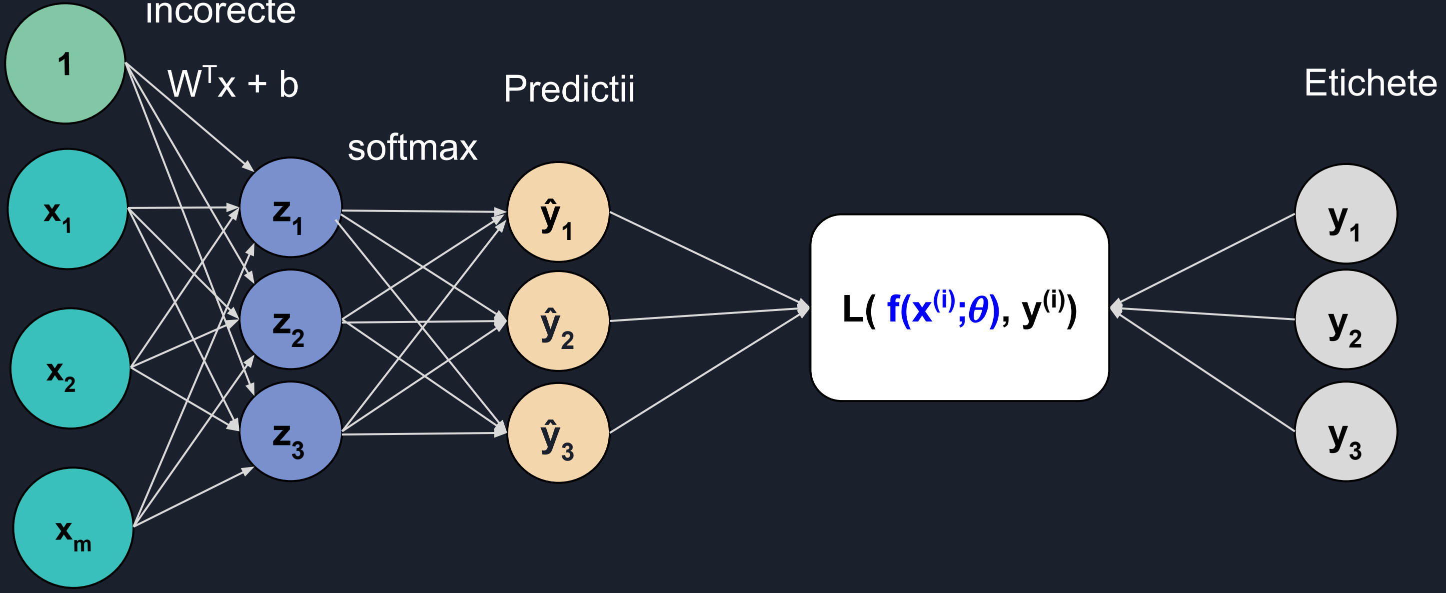




- Functiile de activare liniare gasesc un hiperplan(decision boundary) care imparte spatiul in 2 semiplane
- Cateodata feature-urile **nu sunt liniar separabile**
- Avem nevoie de o **remapare** intr-un alt spatiu in care acestea pot fi separate de un hiperplan

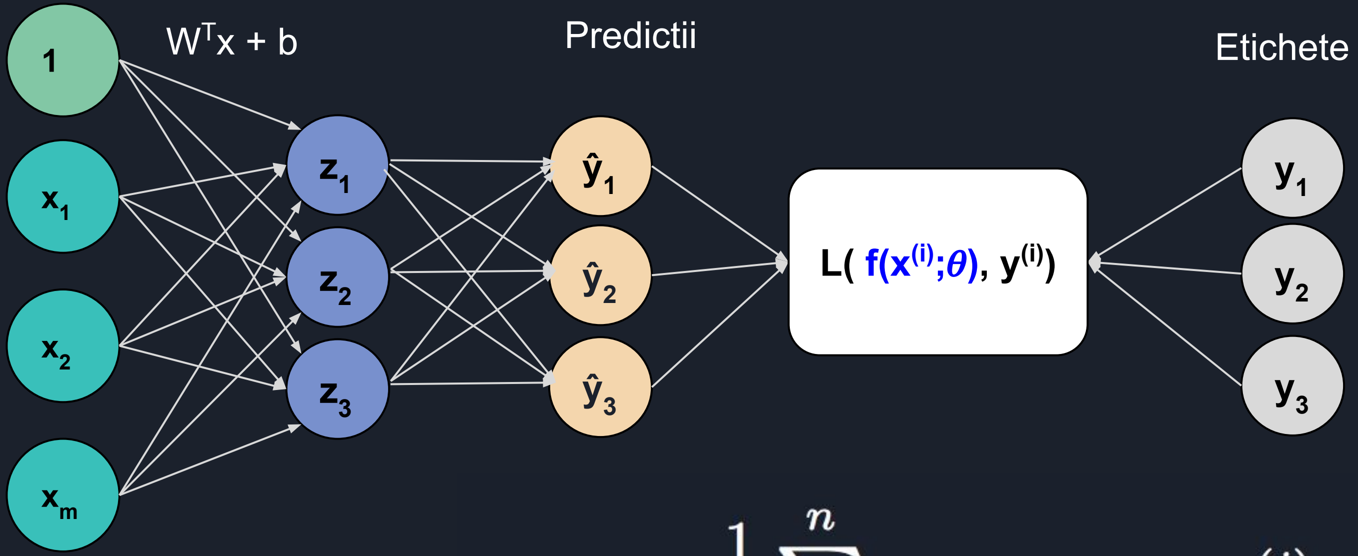
Masurarea costului

- Avem nevoie de o **masura pentru cat de bun este un clasificator**
- Avem un dataset de exemple $\{x_i, y_i\}_1^N$,vrem sa calculam $\hat{y}_i \cong yi$
- $\hat{y} = \sigma(w^*x + b)$
- **Functia de cost** masoara costul pe care trebuie sa-l platim pentru predictii incorecte



Costul empiric (costul total, functia obiectiv, functia de cost, riscul empiric)

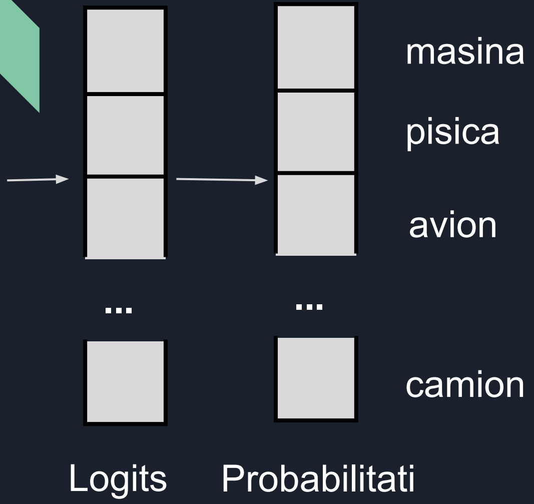
- Masoara costul total peste tot datasetul



pred	actual
0.2	0
0.5	1
0.3	0

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

Functia cost pentru clasificare



- Reteaua modeleaza **distributia claselor conditionata de imaginea data ca input**

$$\hat{y}_k(x) = p(C_k|x) \qquad \hat{y}_i = softmax(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}}$$

- Pornim de la presupunerea ca datasetul este amestecat si datele sunt **i.i.d.**
- Probabilitatea de observare a datasetului este data de:

$$p(Y|X) = \prod_{n=0}^{N} p(y^{(n)}|x^{(n)}) = \prod_{n=0}^{N}\prod_{k=0}^{K}(\hat{y}_k(x^{(n)}))^{y_k^{(n)}}$$

- Maximizarea probabilitatii de a observa datasetul **(likelihood)** este echivalenta cu o functie de cost pe minimizarea probabilitatii logaritmate de observare a datasetului **(negative log-likelihood)**

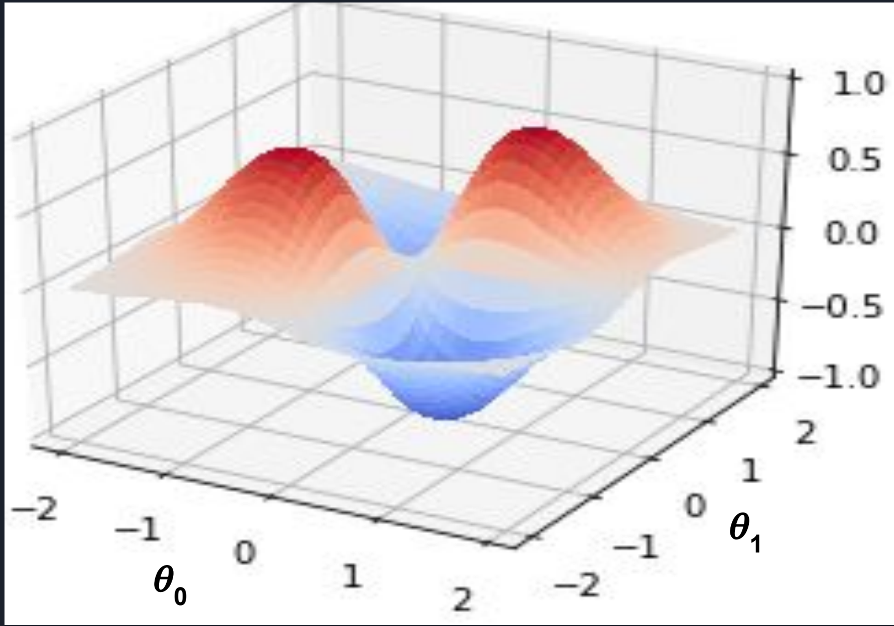
$$L(\theta) = -\sum_{n=0}^{N}\sum_{k=0}^{K} y_k^{(n)} log(\hat{y}_k^{(n)})$$

Optimizare

- **Vrem sa gasim parametrii care ne dau cel mai mic cost**

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

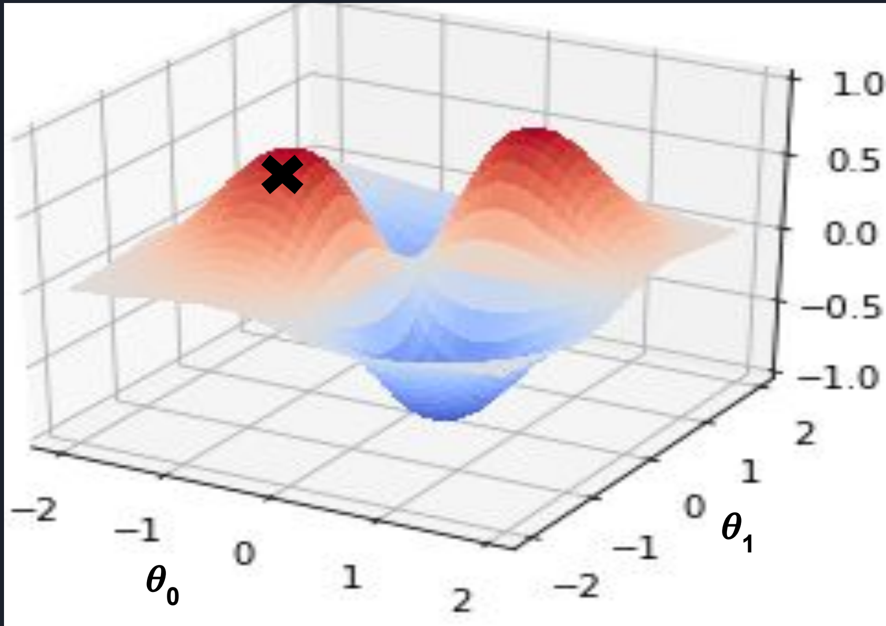
$L(\theta_{0,} \theta_1)$



Stochastic Gradient Descent

- **Alegem parametrii initiali random** $\theta_0^{(0)}\,\theta_1^{(0)}$

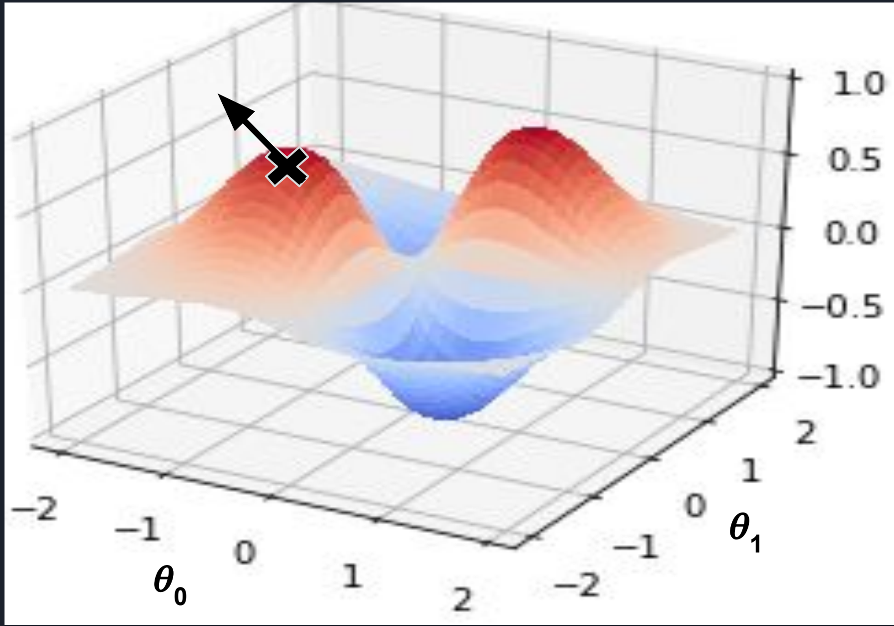
$L(\theta_{0,}\theta_1)$



Stochastic Gradient Descent

- Alegem parametrii initiali random $\theta_0^{(0)}\,\theta_1^{(0)}$
- **Calculam gradientul (derivata)** $\dfrac{\partial L(\theta)}{\partial \theta}$

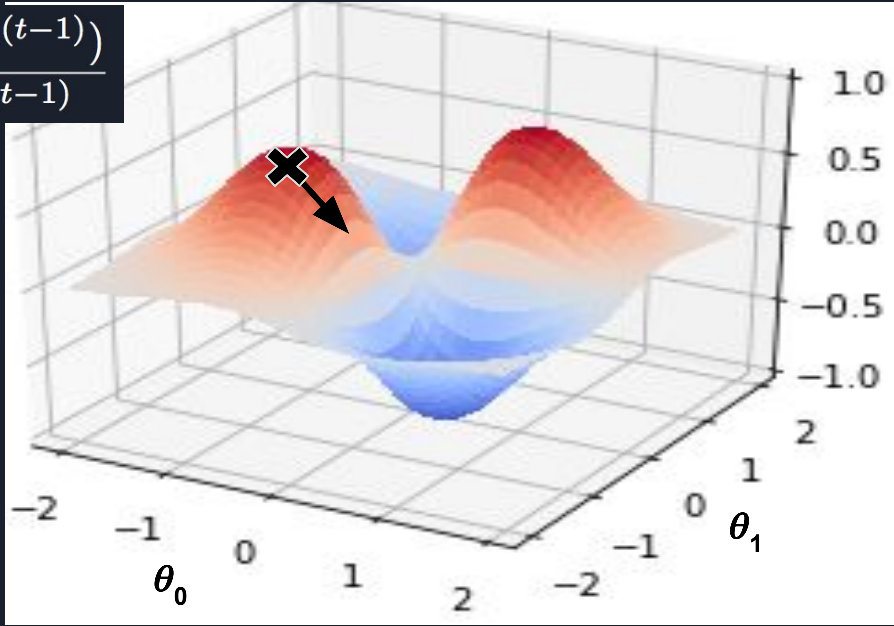
$L(\theta_{0,}\theta_1)$



Stochastic Gradient Descent

- Alegem parametrii initiali random $\theta_0^{(0)}\,\theta_1^{(0)}$
- Calculam gradientul (derivata) $\dfrac{\partial L(\theta)}{\partial \theta}$
- **Facem un pas mic in directia opusa gradientului:** $\theta^{(t)} \leftarrow \theta^{(t-1)} - \alpha \dfrac{\partial L(\theta^{(t-1)})}{\partial \theta^{(t-1)}}$

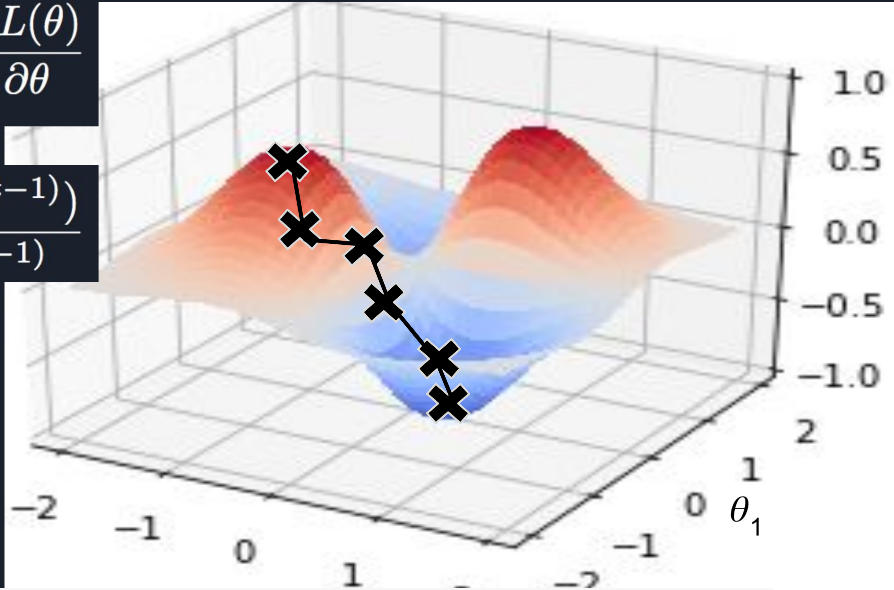
$L(\theta_{0,}\theta_1)$



Stochastic Gradient Descent

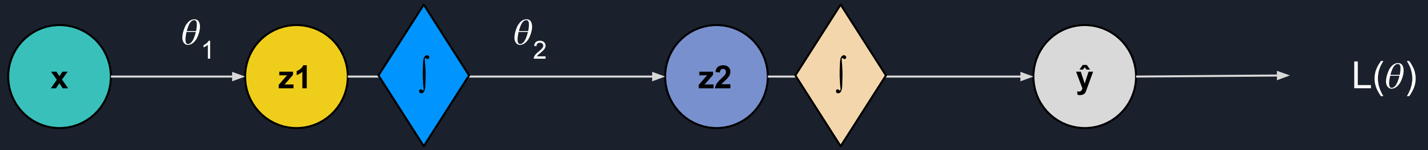
$L(\theta_0, \theta_1)$

- Alegem parametrii initiali random $\theta_0^{(0)}\ \theta_1^{(0)} \sim N(0, \sigma^2)$
- Calculam gradientul (derivata) $\frac{\partial L(\theta)}{\partial \theta}$
- Facem un pas mic in directia opusa gradientului: $\theta^{(t)} \leftarrow \theta^{(t-1)} - \alpha \frac{\partial L(\theta^{(t-1)})}{\partial \theta^{(t-1)}}$
- **Repetam pana la convergenta**




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

Calcularea gradientului - Backpropagation



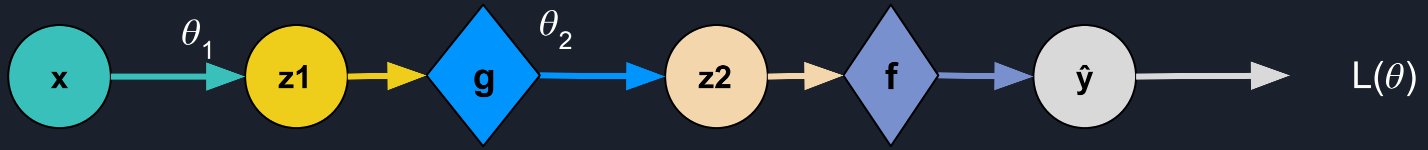
- **Gradientul ne spune cum afecteaza o schimbare mica in parametrii θ costul final L**

- In 1D, derivata unei functii L: $$\frac{\partial L(\theta)}{\partial \theta} = \lim_{h \to 0} \frac{L(\theta + h) - L(\theta)}{h}$$

- In mai multe dimensiuni gradientul este un **vector de derivate partiale** pentru fiecare dimensiune

- Functia obiectiv este parametrizata de θ => putem folosi reguli pentru a calcula **gradientul analitic**

Calcularea gradientului - Backpropagation



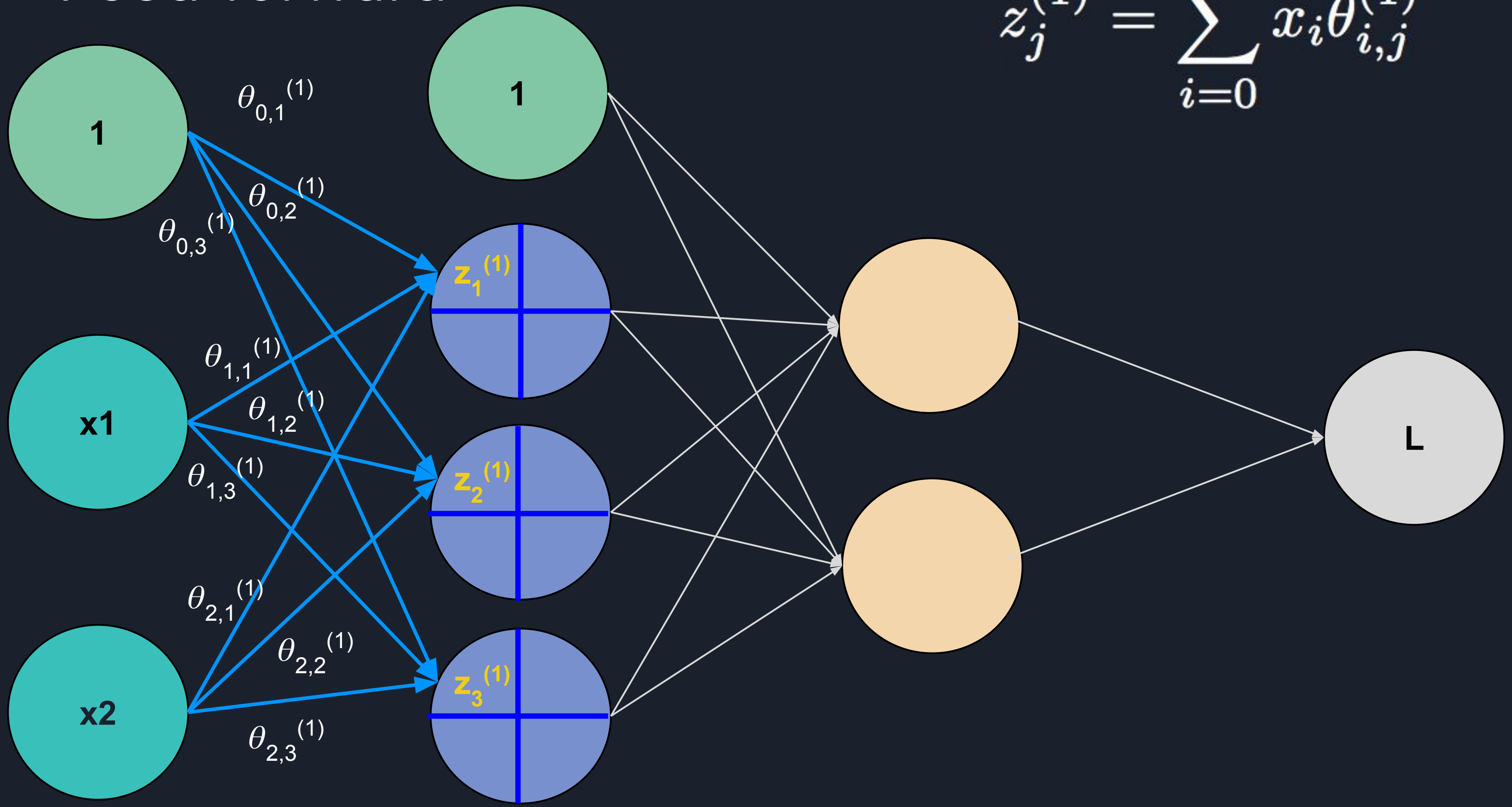
- **The chain rule**

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

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

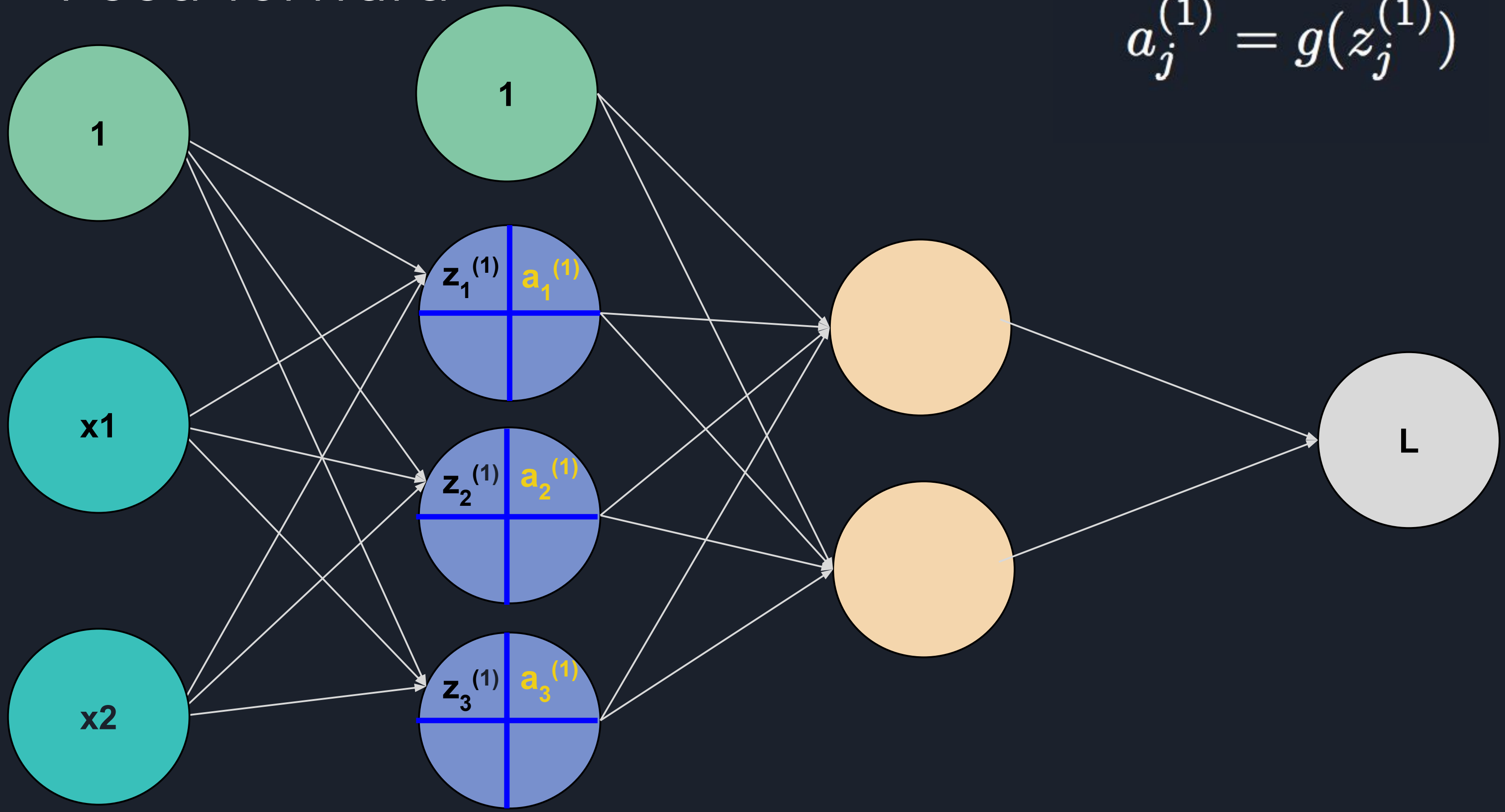
Feed-forward

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



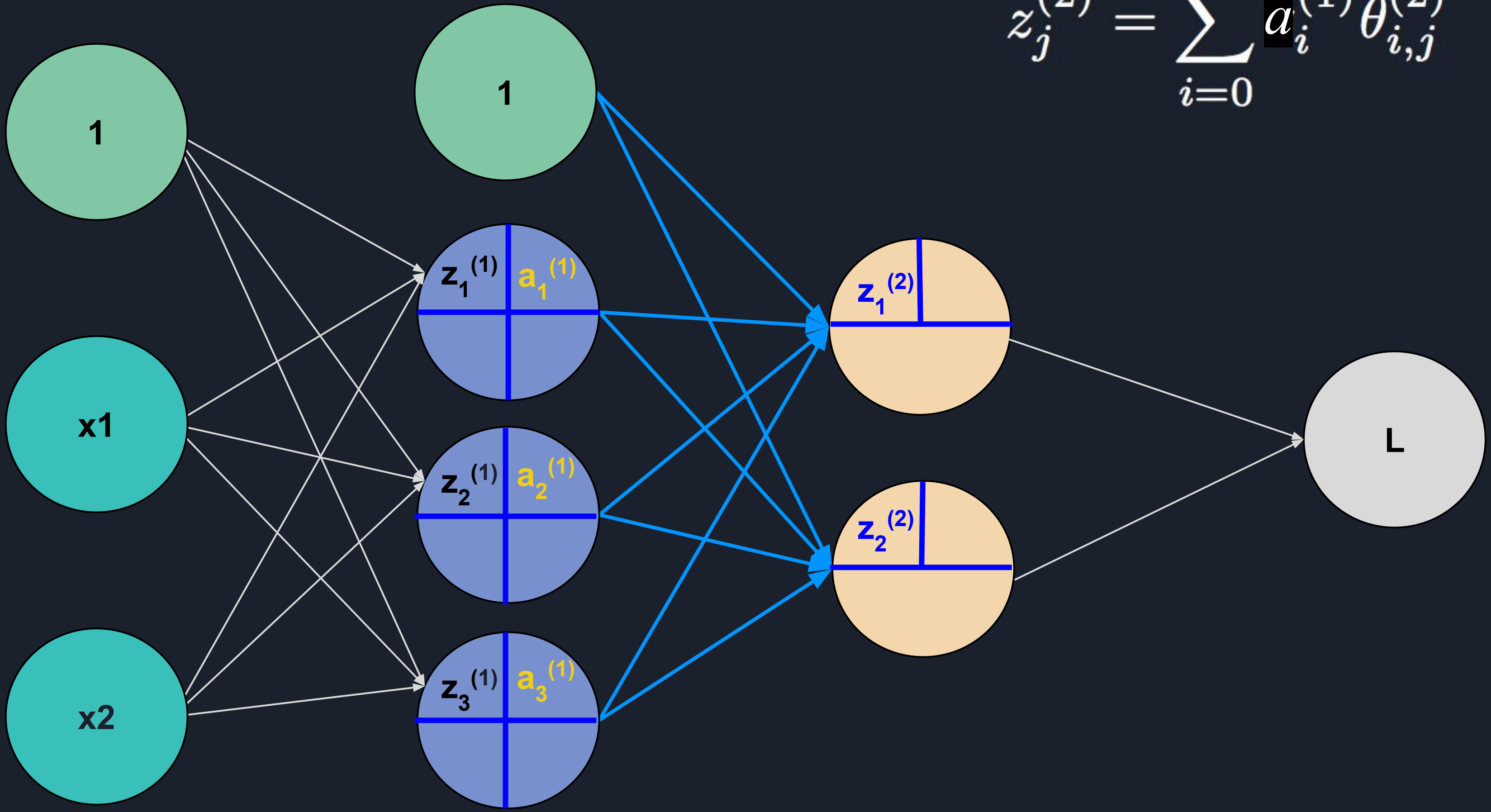
Feed-forward

$$a_j^{(1)} = g(z_j^{(1)})$$



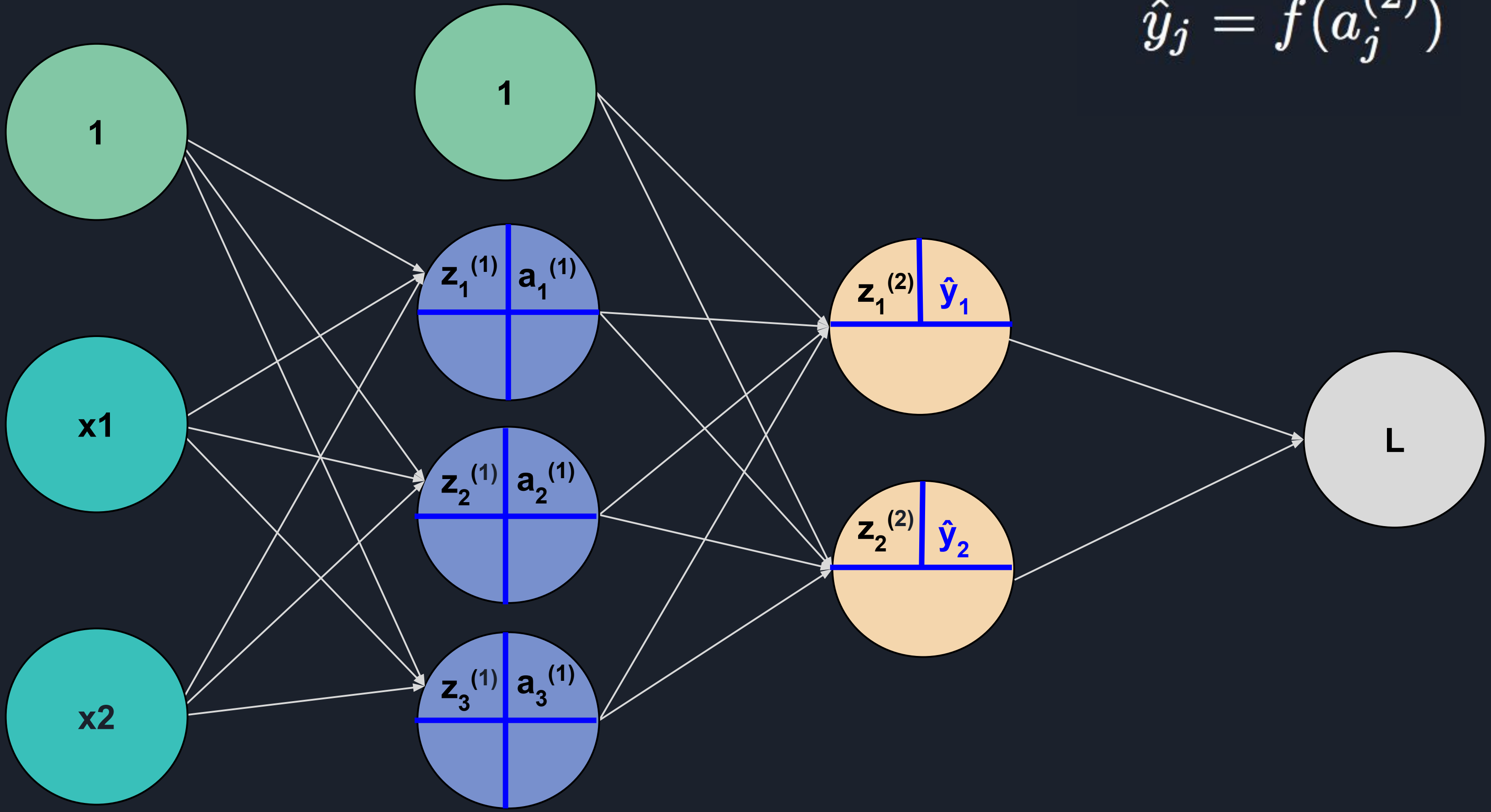
Feed-forward

$$z_j^{(2)} = \sum_{i=0} a_i^{(1)} \theta_{i,j}^{(2)}$$



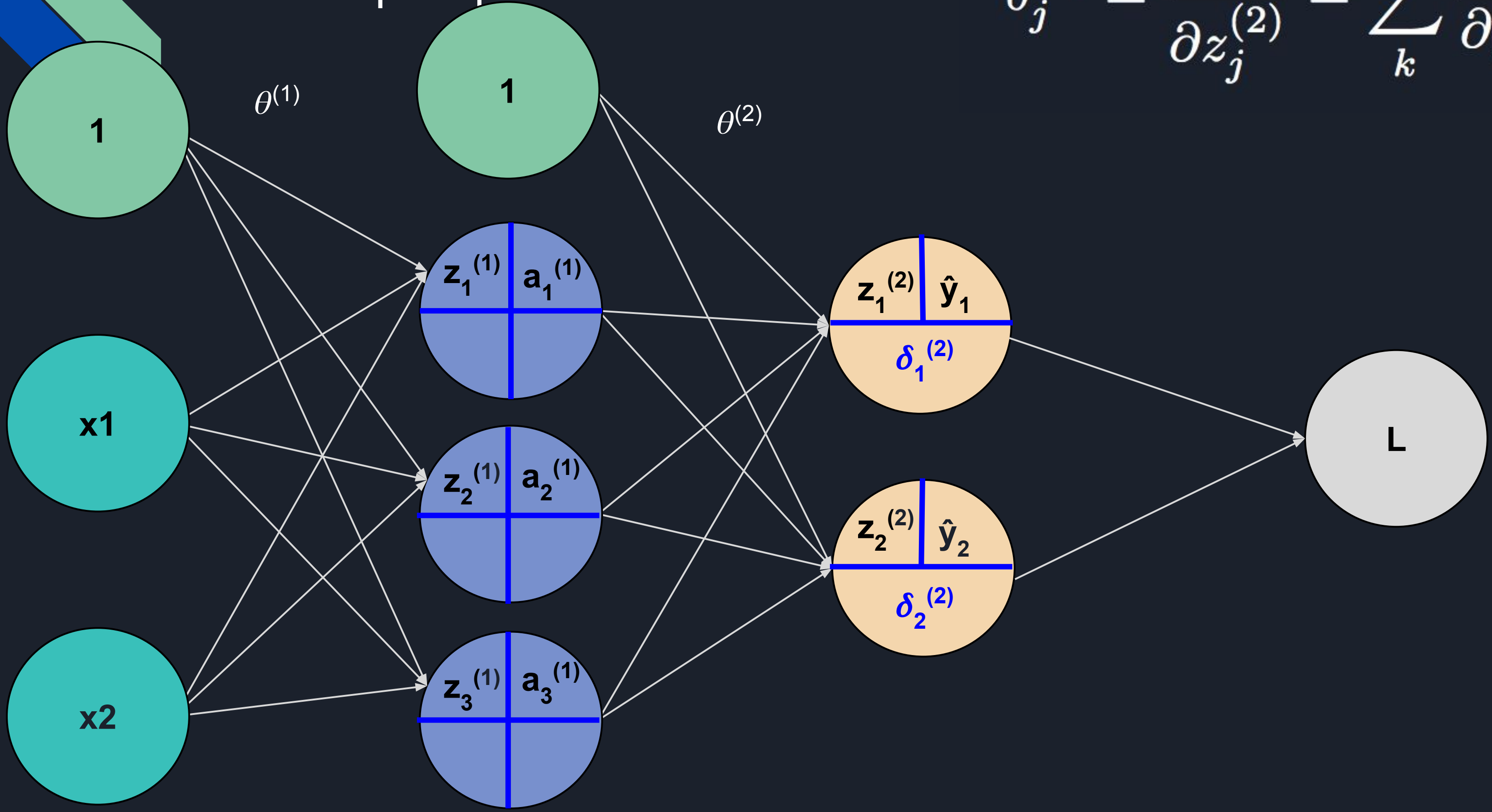
Feed-forward

$$\hat{y}_j = f(a_j^{(2)})$$



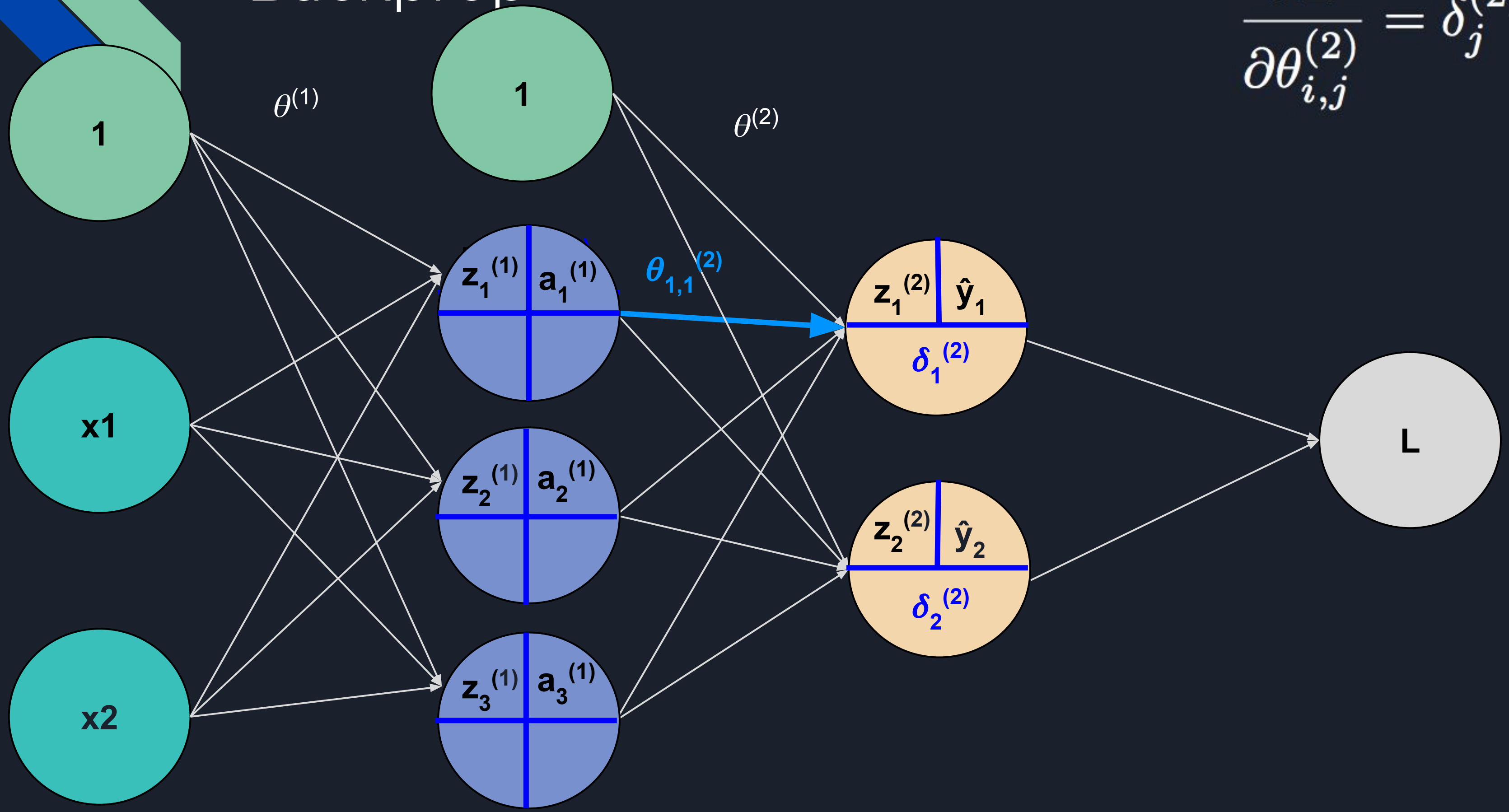
Backprop

$$\delta_j^{(2)} = \frac{\partial L}{\partial z_j^{(2)}} = \sum_k \frac{\partial L}{\partial \hat{y}_k} \frac{\partial \hat{y}_k}{\partial z_j^{(2)}}$$



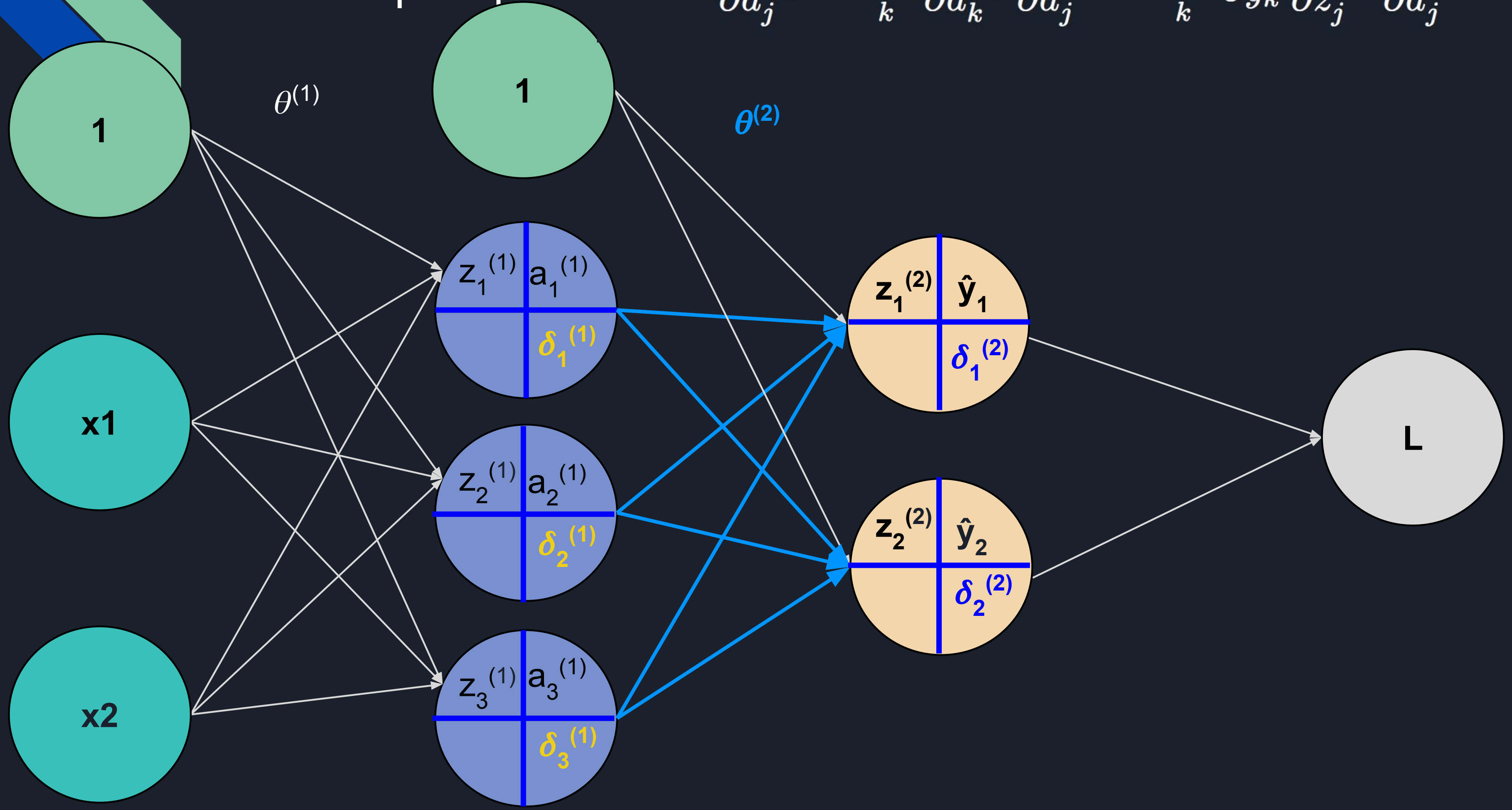
Backprop

$$\frac{\partial L}{\partial \theta_{i,j}^{(2)}} = \delta_j^{(2)} a_i^{(1)}$$



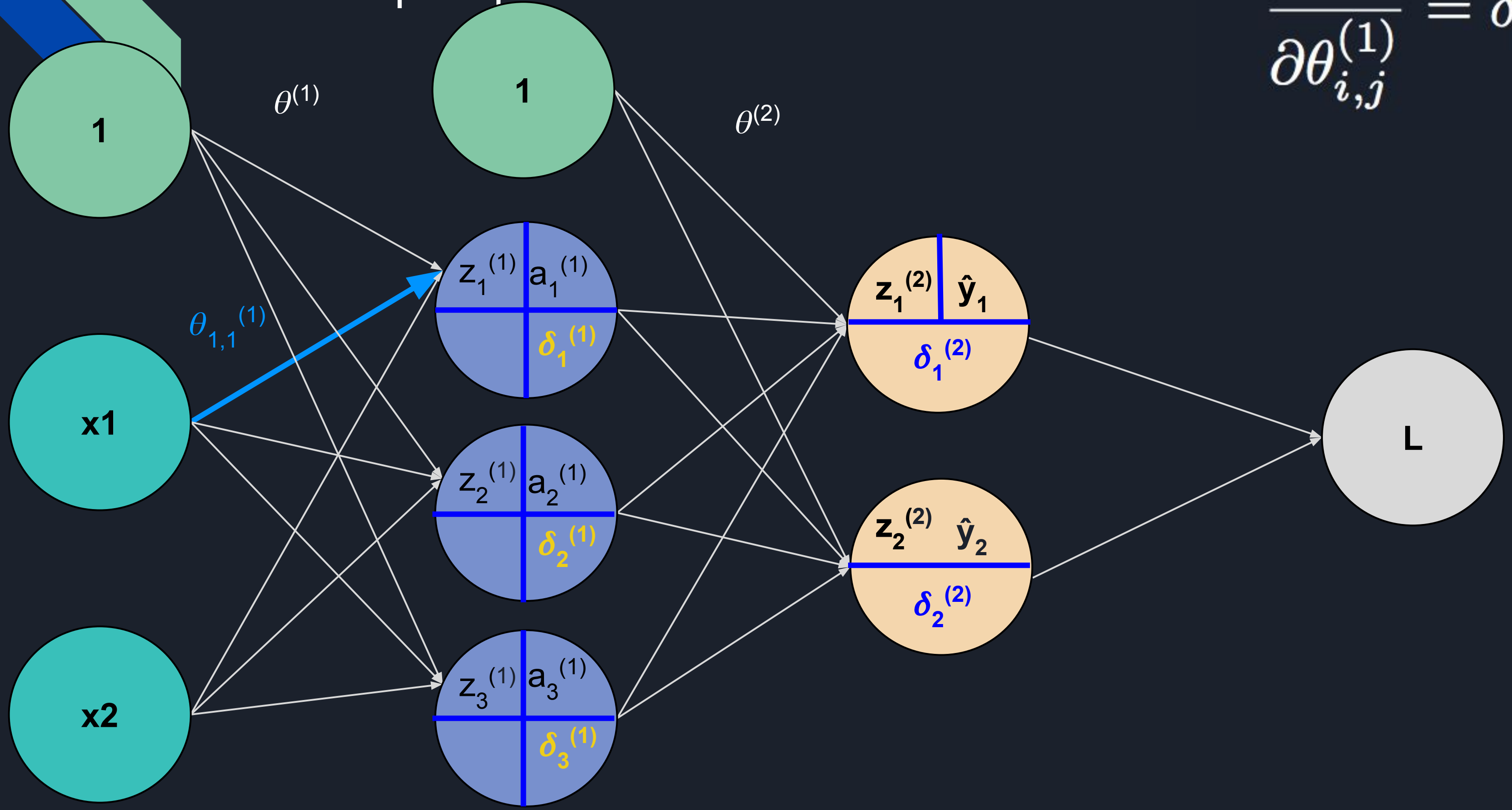
Backprop

$$\delta_j^{(1)} = \frac{\partial L}{\partial a_j^{(1)}} = \sum_k \frac{\partial L}{\partial a_k^{(2)}} \frac{\partial a_k^{(2)}}{\partial a_j^{(1)}} = \sum_k \frac{\partial L}{\partial \hat{y}_k} \frac{\partial \hat{y}_k}{\partial z_j^{(2)}} \frac{\partial z_j^{(2)}}{\partial a_j^{(1)}} = \sum_k \delta_k^{(2)} \theta_{j,k}^{(2)}$$

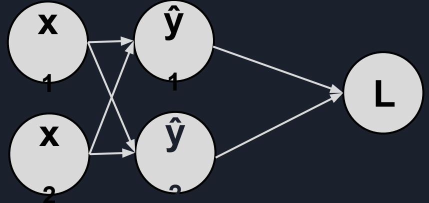


Backprop

$$\frac{\partial L}{\partial \theta_{i,j}^{(1)}} = \delta_j^{(1)} x_1$$



Gradientul cross-entropiei



$$\hat{y}_k(x) = p(C_k|x)$$

$$p(Y|X) = \prod_{n=0}^{N} p(y^{(n)}|x^{(n)}) = \prod_{n=0}^{N}\prod_{k=0}^{K}(\hat{y}_k(x^{(n)}))^{y_k^{(n)}}$$

= take neg log likelihood

$$L(\theta) = -\sum_{n=0}^{N}\sum_{k=0}^{K} y_k^{(n)} log(\hat{y}_k^{(n)})$$ = negative log likelihood

$$\boxed{\frac{\partial L(x_i)}{\partial x_i}} = \sum_k \boxed{\frac{\partial L}{\partial \hat{y}_k}} * \boxed{\frac{\partial \hat{y}_k}{\partial x_i}}$$

$$\boxed{\frac{\partial L(x_i)}{\partial x_i}} = \hat{y}_i - y_i$$

$$\frac{\partial \frac{f(x)}{g(x)}}{\partial x} = \frac{\frac{\partial f(x)}{\partial x} * g(x) - f(x) * \frac{\partial g(x)}{\partial x}}{g(x)^2} \qquad \delta_{ki} = \begin{cases} 1, & k == i \\ 0, & \text{otherwise} \end{cases}$$

Reminder !

$$\boxed{\frac{\partial L}{\partial \hat{y}_k}} = -\frac{y_k}{\hat{y}_k}$$

$$\boxed{\frac{\partial \hat{y}_i}{\partial x_i}} = \frac{\partial \frac{e^{x_i}}{\sum_j e^{x_j}}}{\partial x_i} = \hat{y}_i * (1 - \hat{y}_i)$$

$$\boxed{\frac{\partial \hat{y}_k}{\partial x_i}} = \frac{\partial \frac{e^{x_k}}{\sum_j e^{x_j}}}{\partial x_i} = -\hat{y}_k * \hat{y}_i$$

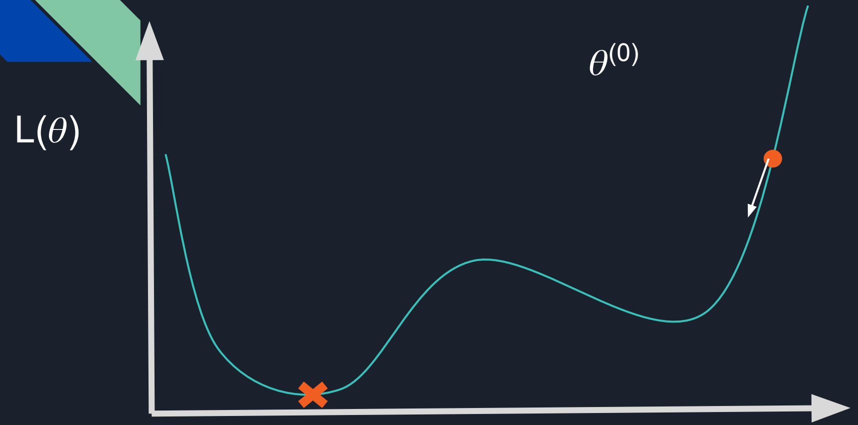
$$\boxed{\frac{\partial \hat{y}_k}{\partial x_i}} = \hat{y}_k * (\delta_{ki} - \hat{y}_i)$$

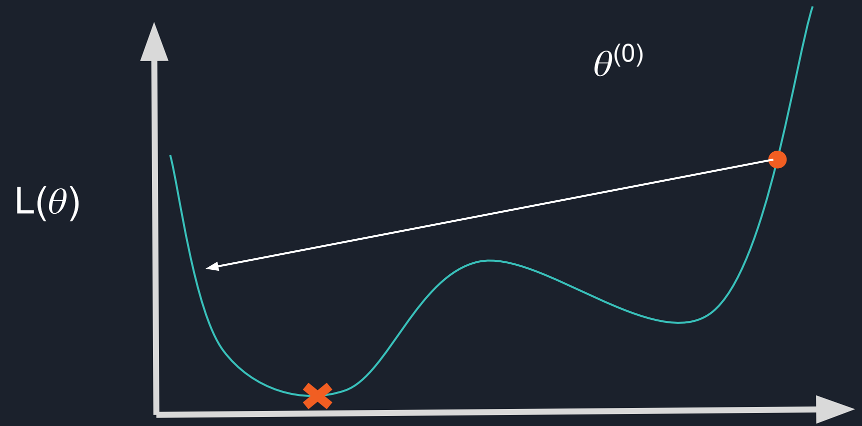
Updatarea parametrilor

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

- **Stochastic vs batch gradient descent**
  - **Stochastic Gradient Descent**
    - Modificam parametrii dupa fiecare exemplu
    - Exemple On-line, dataseturi redundante foarte mari
  - **Batch Gradient Descent**
    - Modificam parametrii dupa ce calculam eroarea (L) peste toate exemplele din dataset
    - Poate fi paralelizat, eroarea (L) este estimata foarte bine
  - **Mini-batch Gradient Descent (cateodata denumit si SGD - stochastic gradient descent)**
    - Modificam parametrii dupa ce calculam eroarea (L) peste un mini-batch de exemple din dataset (32, 64, 128, 256, 512, 1024)

Setarea ratei de invatare





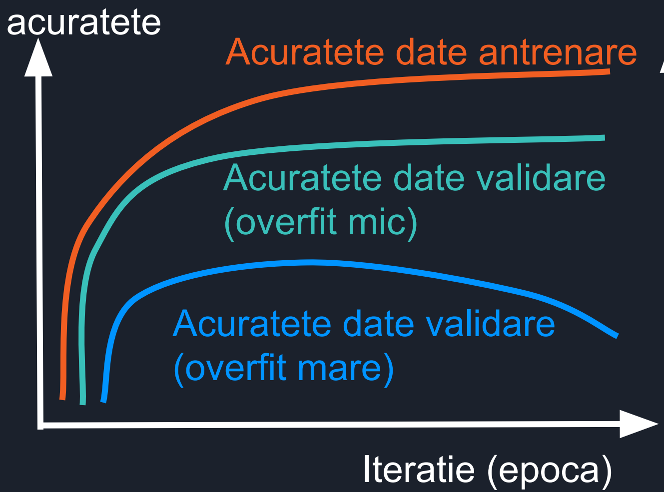
- **Rata de invatare mica**
  - Converge foarte lent
  - Poate ramane blocata intr-un minim local
- **Rata de invatare mare**
  - Face un pas prea mare - sare peste goal
  - Devine instabila, diverge
- **Rata de invatare adaptabila**
  - In episodul urmator...

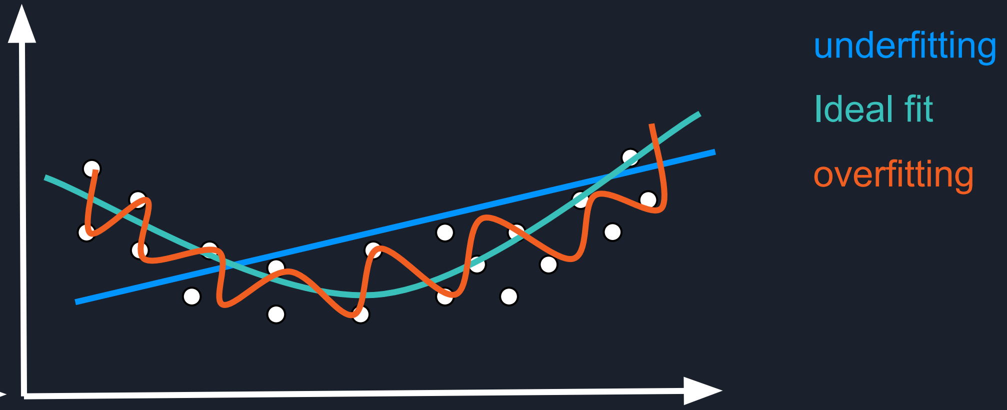
Overfitting

- **Generalizare**
  - Proprietatea unui estimator functional 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





Initializarea parametrilor

- **Initializare cu zero ?**
    - Fiecare neuron calculeaza acelasi output, acelasi gradient si executa acelasi update
- **Initializare cu numere mici random**
    - Fiecare neuron este unic si calculeaza update-uri distincte
    - Initializare dintr-o gaussiana centrata in 0 cu varianta = sqrt(0.01)

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

    - Daca parametrii sunt prea mici, gradientul (care e proporțional pe valoarea parametrilor) va fi mic
- **Calibrarea variantei**

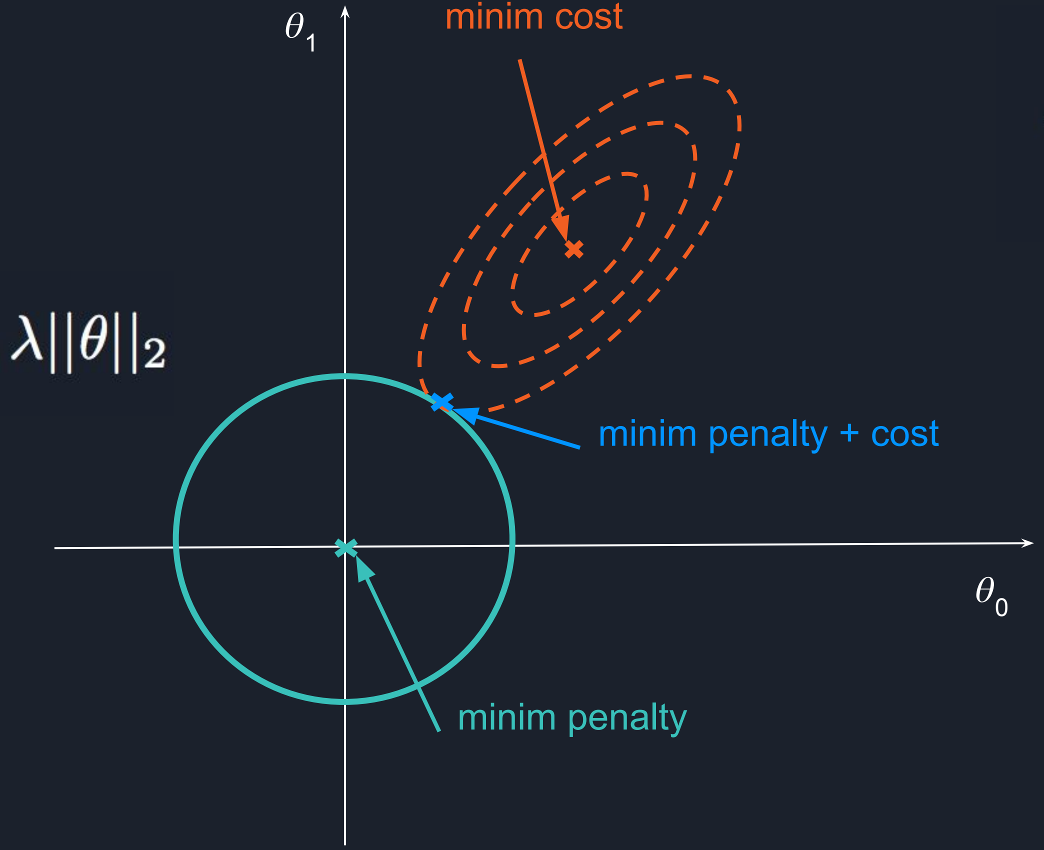
```
w = np.random.randn(n) / sqrt(n)
```

    - Distributia activarilor unui neuron initializat random creste cu numarul de intrari
    - Putem normaliza varianta fiecarui neuron pentru ca output-ul sa aiba varianta 1

Overfitting

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

L2 norm



$$\lambda||\theta||_2 = \lambda * \sqrt{\sum_j \theta_j^2}$$

Sumar concepte fundamentale

Perceptronul

- ★ Unitatea de baza
- ★ Clasificare
- ★ Functii de activare nonliniare

Retele cu un singur strat

- ★ Suprapunerea perceptronilor pentru a construi retele
- ★ Functii de cost
- ★ Optimizare cu backpropagation

Antrenare

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