

Introdução às Redes Neurais Artificiais

Cleber Zanchettin

Where we are...

$$s = f(x; W) = Wx$$

scores function

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

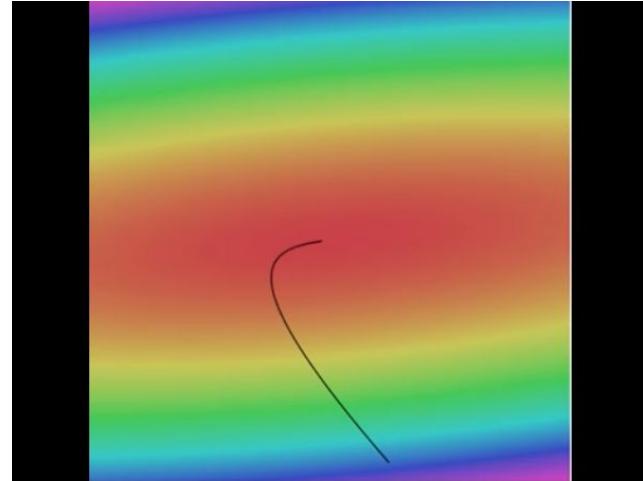
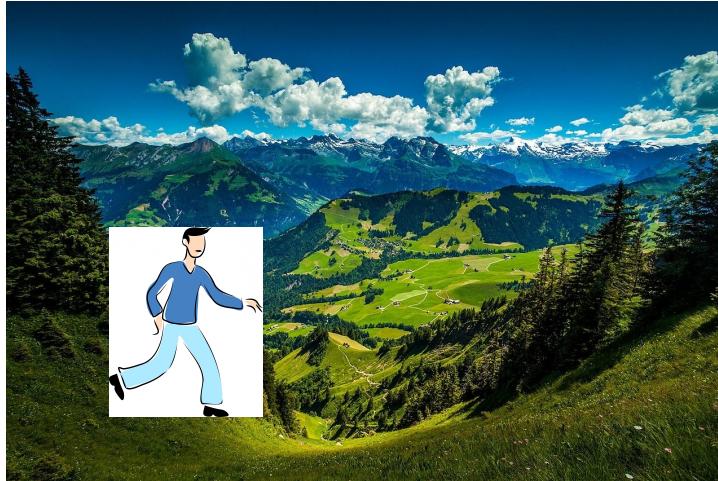
SVM loss

$$L = \frac{1}{N} \sum_{i=1}^N L_i + \sum_k W_k^2$$

data loss + regularization

want $\boxed{\nabla_W L}$

Optimization



```
# Vanilla Gradient Descent

while True:
    weights_grad = evaluate_gradient(loss_fun, data, weights)
    weights += - step_size * weights_grad # perform parameter update
```

Landscape image is [CC0 1.0](#) public domain
Walking man image is [CC0 1.0](#) public domain

Gradient descent

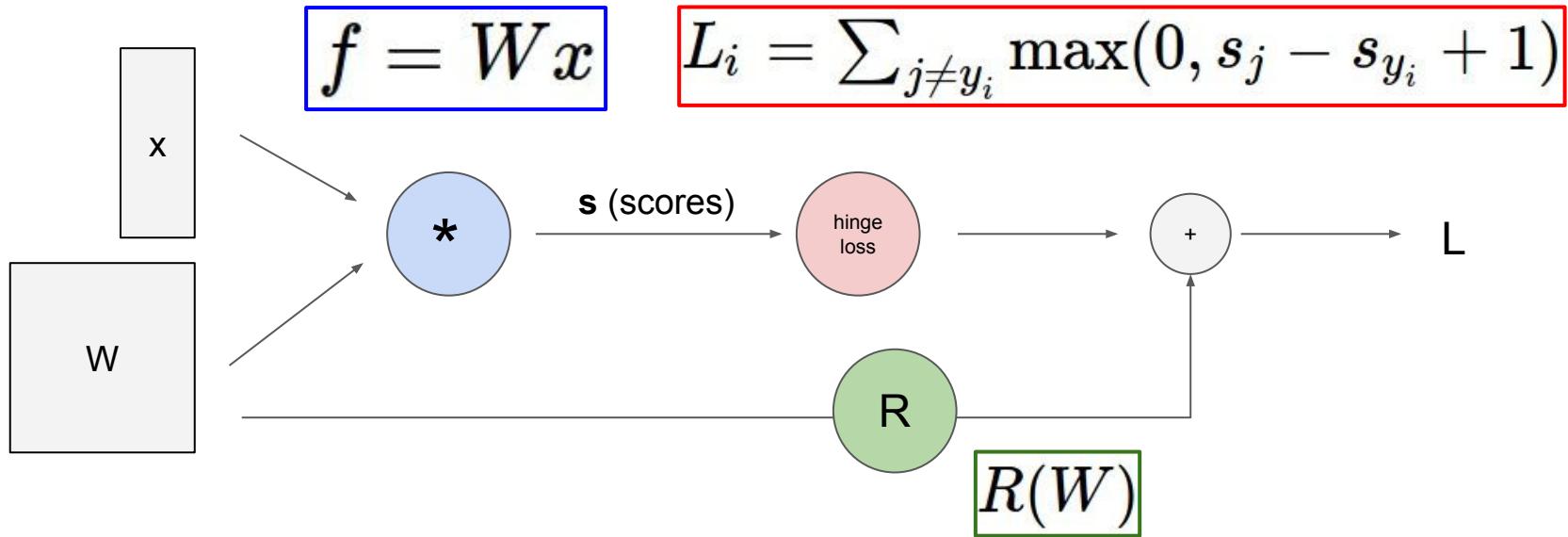
$$\frac{df(x)}{dx} = \lim_{h \rightarrow 0} \frac{f(x + h) - f(x)}{h}$$

Numerical gradient: slow :, approximate :, easy to write :)

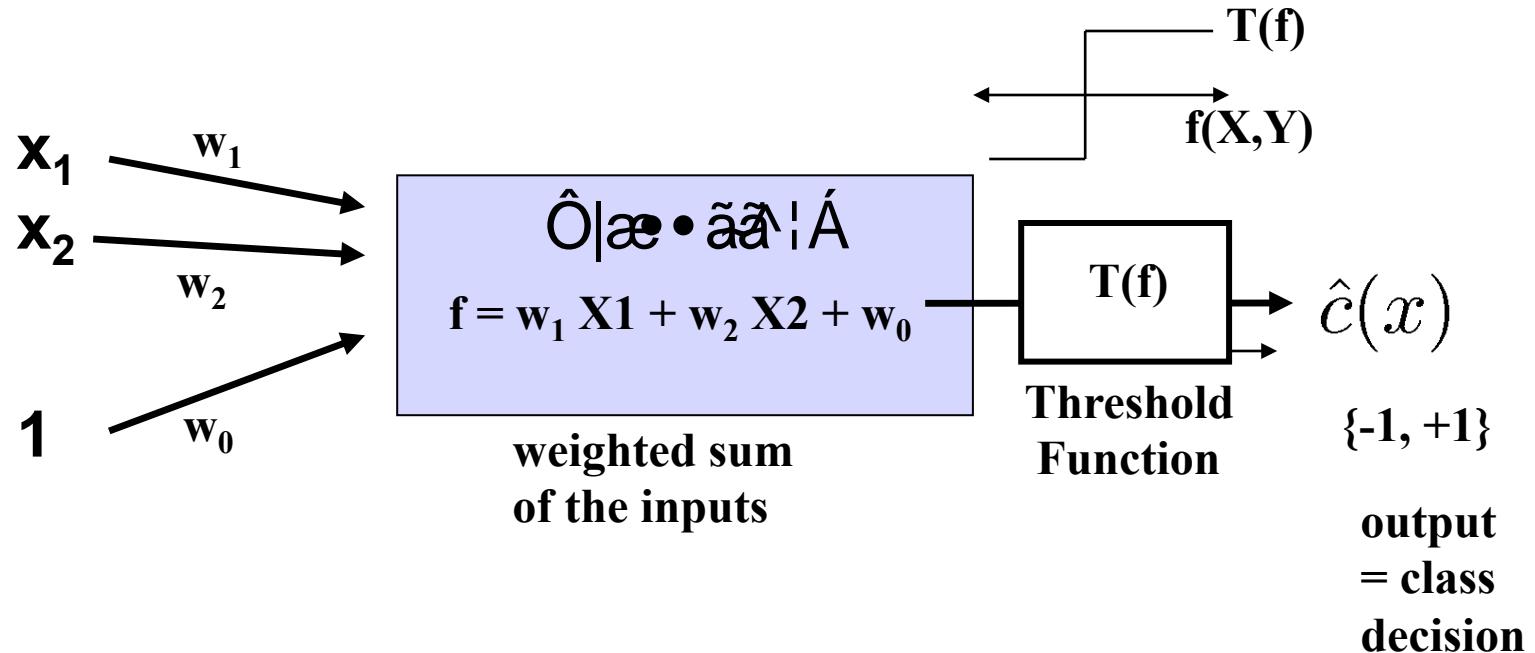
Analytic gradient: fast :), exact :), error-prone :(

In practice: Derive analytic gradient, check your implementation with numerical gradient

Computational graphs



Ú^{\&} d[} \hat{O}æ•ää^{\&} GÁ^æ^{\&} |^{\&} DÁ

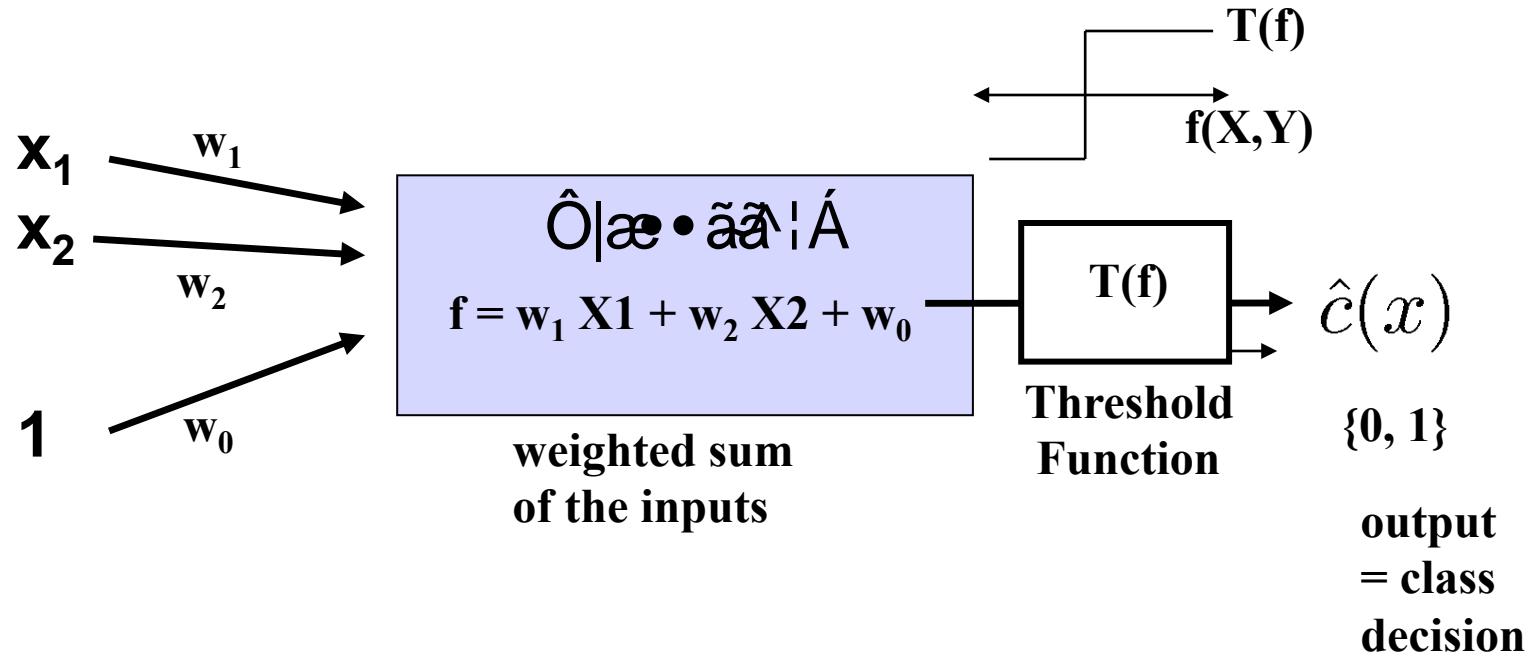


Decision Boundary at $f(x) = 0$

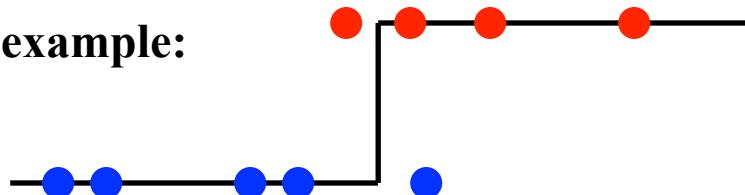
Solve: $X_2 = -w_1/w_2 X_1 - w_0/w_2$ (Line)

(c) Alexander Ihler

Ú^{\& } d[} Ášá ^æ&æ•ää^DÁ



1D example:



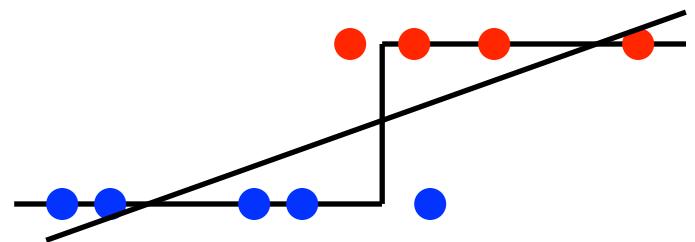
$$\begin{aligned} T(f) &= 0 \text{ if } f < 0 \\ T(f) &= 1 \text{ if } f > 0 \end{aligned}$$

Decision boundary = “ x such that $T(w_1 x + w_0)$ transitions”

| Ø^ǣ |^• Á̄ḡ åÁ̄ ^|&^] d[} • Á

Ü^ǣ Á@ Á[|^Á̄ Á^ǣ |^•

- . Y ^Á̄ & Á!^ǣ Ácdǣ |^• Á@ǣ [, Á [|^Á̄ { } |^c Á^ǣ }
- . à[` } åǣ •
- . Š̄ ^ǣ & ǣ • ã̄ |•
- . Ø^ǣ |^• Á̄ F̄cá
 - „ Ö^ǣ } Á |^KÁ̄ Ḡ EàD ÁM̄c Ē ÁND ÁE
 - „ Ó[` } åǣ Á̄ Eà ÁM̄ ÁN̄ N̄ [ã̄ c



$\emptyset \wedge \text{æ} \vee \text{^• Á} \wedge \text{å Á} \wedge \text{^! & }] \text{ d[} \} \bullet \text{ Á}$

Ü⁸ & æ | Á @ Á [| ^ Á | Á ^ æ | ^ • Á

- Y ^ Á & æ Á | ^ æ ^ Á cd æ ^ æ | ^ • Á @ Á [| , Á [| ^ Á { } | ^ c Á ^ & å Á } Á
à [` } å æ à • Á

- Š å ^ æ & æ • å å | • Á

- Ø ^ æ | ^ • Á F E Á

 - Ö ^ & å Á } Á | ^ Á V G E à D M Á E Á Á N D Á E Á

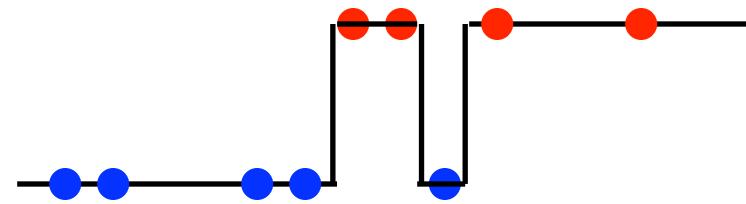
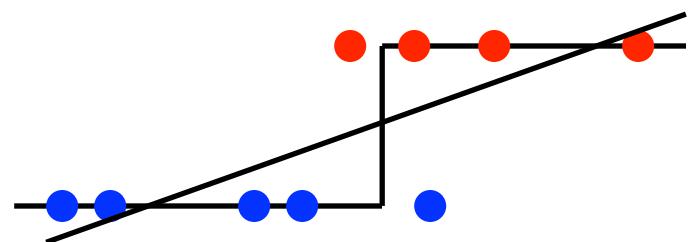
 - Ó [` } å æ ^ Á E à Á M Á N Á [ä o Á

- Ø ^ æ | ^ • Á F E E G Á

 - Ö ^ & å Á } Á | ^ Á V G E à c E & D M Á

 - Ó [` } å æ ^ Á E à c E & Á M Á N Á

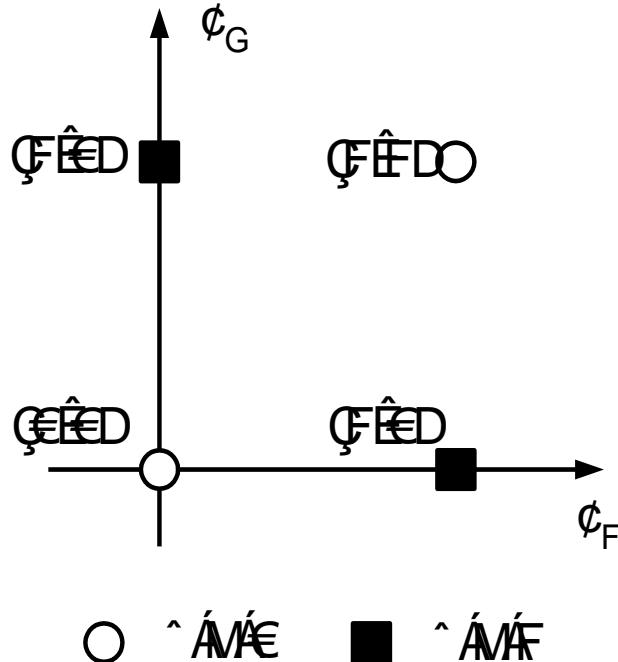
- Y @ Á ^ æ | ^ • Á & æ Á | [å ^ & Á @ Á ^ & å Á } Á | ^ Ñ Á



Problema do XOR (ou-exclusivo)

Portanto, é necessário que a saída seja 1 se e somente se exatamente um dos dois entradas for 1.

Função XOR



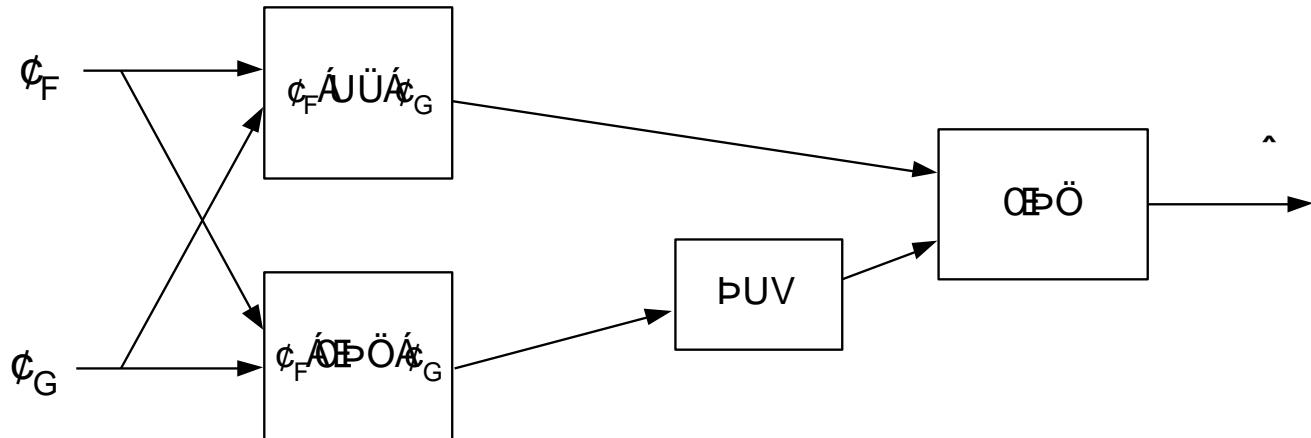
O neurônio do tipo Perceptron é de tipo $\text{f}(x) = \sum w_i x_i + b$, onde w_i são os pesos, x_i são os valores das entradas e b é o termo de bias. A função ou-exclusivo é:

Features and perceptrons

Oé } 8é[ÁUÜÁ• a Á. { Áa&á a&a^Á{ ÁU&^] d[} Áá] |^•É
Ô[} c á[É{ Perceptron simples pode implementar funções lógicas
elementares KÖPÖÜÁÁPUVÉ

OE•á É^Á{ aÁ } 8é[Á[á^Á^|Áç] |^••a&{ [Á{ acombinação dessas
funções lógicas elementares É} ö[Á••aÁ } 8é[Á[á^Á^|Áç] |^{ ^} ca&Á
^•a á[Á a Á^|f} á•ÉÉÉÉ

Ú[|Áç^{] |[ÉÜÜÁ[á^Á^|Áç] |^••aÁ[|Kç_F [|Á_G Dá a Áç [Áç_F a á Áç_G DÉ

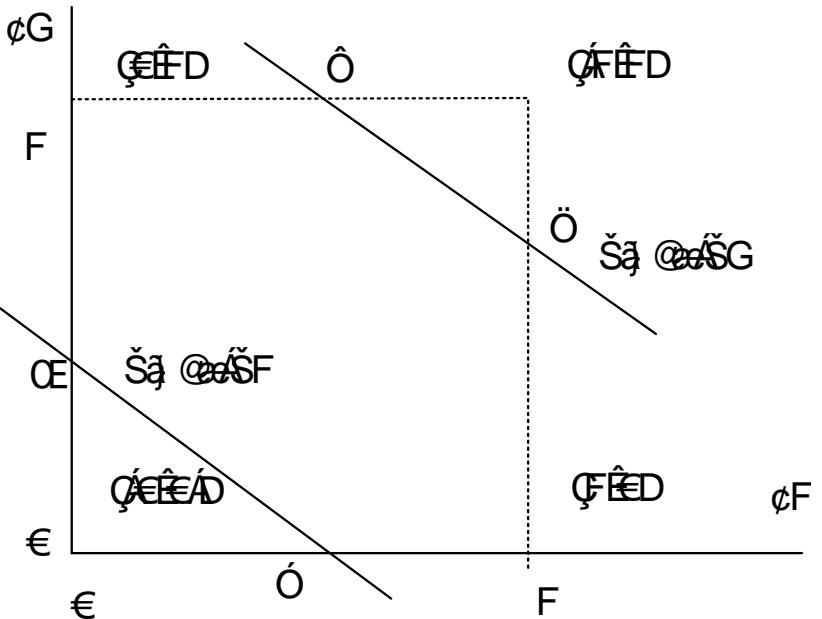
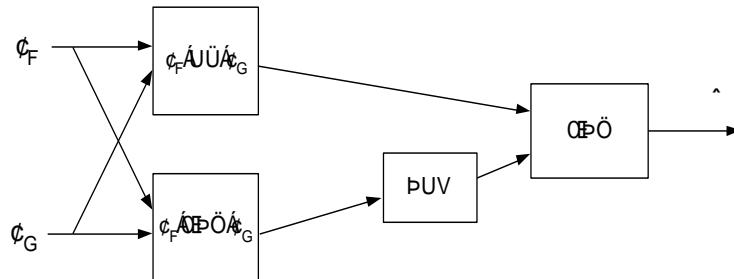


Separação linear para XOR de duas camadas

Ò{ Á^{|{ [•Á^Á^] ææë[Áá ^æÁç^{| æ&æë[□Í•[Á~^ ~ææ^Á~ Áææ[Á^Á
duas linhasÈ

OÁ æc^ Ácima åæA@eL1 & [||^•] [} å^ ÁeÁ } 8ë[ORÉA ÁæA æc^ Á
abaixo åæA@eL2 & [||^•] [} å^ ÁeÁ } 8ë[NOT ANDÉA

OÁ |^æentre as duas linhas & [||^•] [} å^ÁeÁ } 8ë[ÁXOREÁ Á^ ^Á& [||^•] [} å^ÁeÁ
| |^æOÖÖEÁ ^ ^Á& [} c^{ Á•Á [} c•Á^GÉDÁ Á^FÉDE



Ø Æ œ { • Á œ å Á ^ ¡ & ^] d [} • Á

Ü^&æ|Á@ Á| |^Á ~Á^æč |^• Á

- Y ^Á&æ Á!^æ^Áçdæ Á^æc |^• Á@ææ[, Á [|^Á&{ } |^çÁ^&æ } Á
à[` } åææ• Á
 - Ø[Áçæ] |^Ê[|` } [{ æ Á^æc |^• Á
ÁPçDÍMÆFACAC AC A Á

“ Y @eÁ c@! Á ã å• Á ~ Á^æč | ^• Á&[^|åÁ ^ Á&@ [• ^ ÑÁ

- ÚC^] Á } &ç } • ÑÁ

The figure displays three step functions labeled F1, F2, and F3. Each function is represented by a thick black line. F1 starts at a low value, remains constant for a period, and then jumps to a higher value. F2 follows a similar pattern but with a longer constant segment before its jump. F3 has the longest constant segment and a very large jump.

Linear function of features

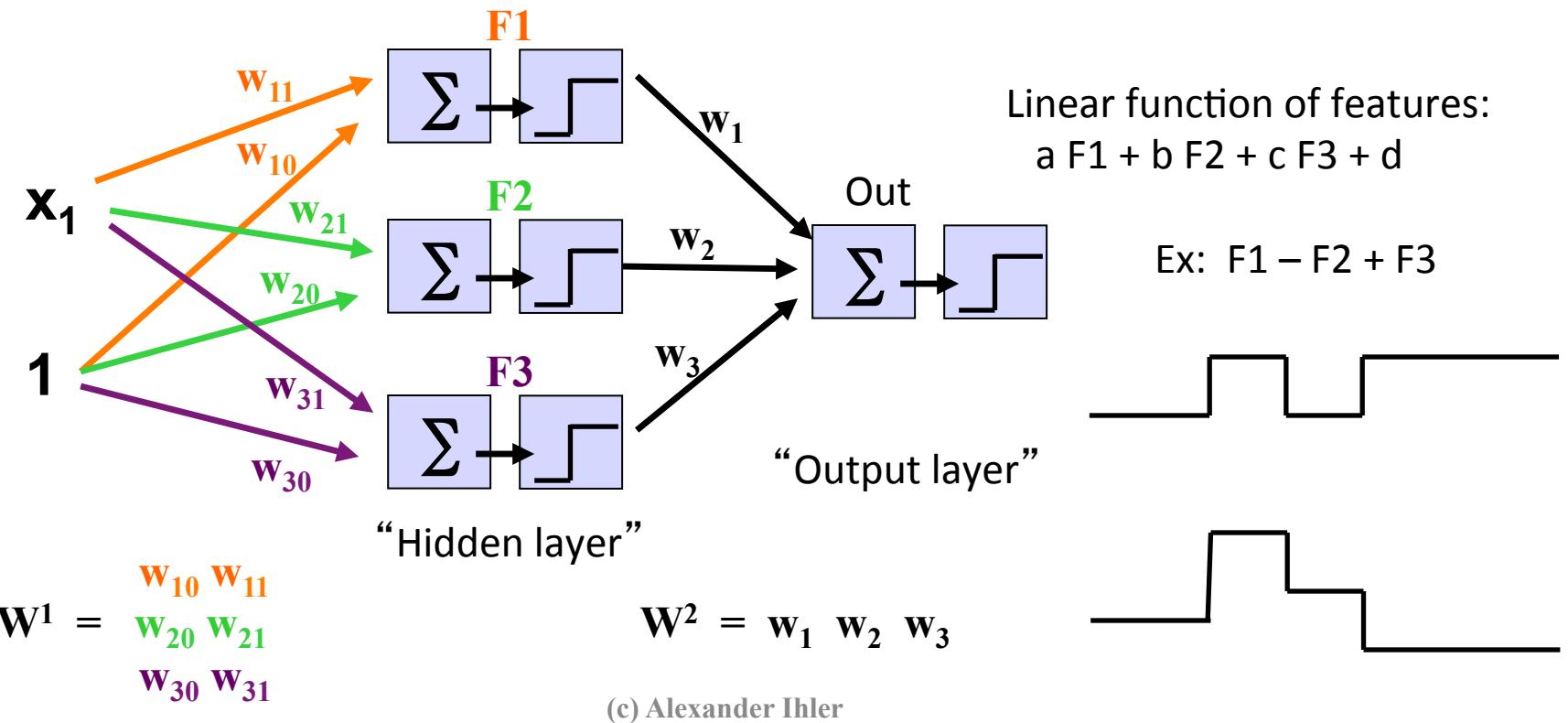
$$a F_1 + b F_2 + c F_3 + d$$

Ex: F1 – F2 + F3



T ^ |c| æ ^ |A| ^ |&^] d[} A [å ^ |A

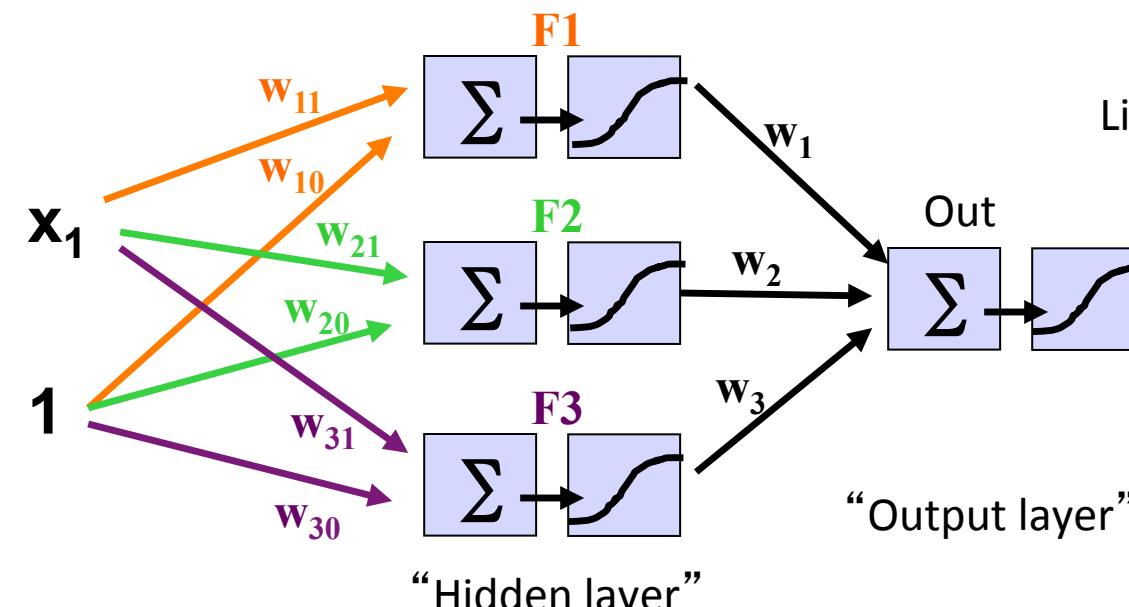
- Uc^] A^} &c^} • A^ ^ A^ • A^ ^ |&^] d[} • A^
- "Øæ |^•" A^ ^ A^ ^ q^ o A^ A^ ^ |&^] d[} A
- O[{ a^æ æ^} A^ A^æ |^• A^ ^ q^ o A^ A^ [c@|A



T ^ |cæ|^A ^ [&^] d[} A [å^|A

- Uc^] A^ } &c[} • A^&^A^ • A^&^A^ d[} • A
- “Øæ|^•” A^&^A^ q^ o A^&^A^ d[} A
- O{ aææ } A^&^æ|^• A^&^q^ o A^&^o @ A

Regression version:
Remove activation
function from output

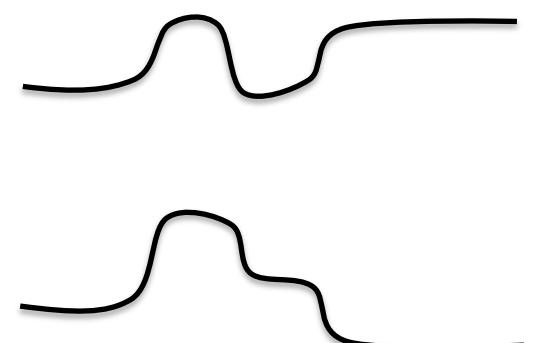


Linear function of features:
 $a F1 + b F2 + c F3 + d$

Ex: $F1 - F2 + F3$

$$W^1 = \begin{matrix} w_{10} & w_{11} \\ w_{20} & w_{21} \\ w_{30} & w_{31} \end{matrix} \quad W^2 = w_1 \quad w_2 \quad w_3$$

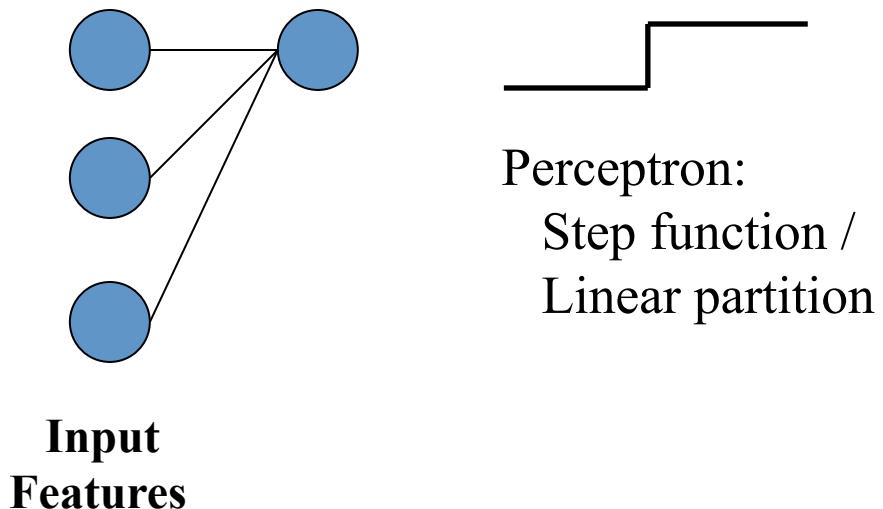
(c) Alexander Ihler



| Ø^ǣ | ^• Á | ~Á ŠÚ• Á

„ Ùǟ] | ^Á~ áåǟ * Á|[& • Á
. Òǣ@Á| ^{ ^} d̄ Á• Á Á ^{ & } d[} Á' } Á

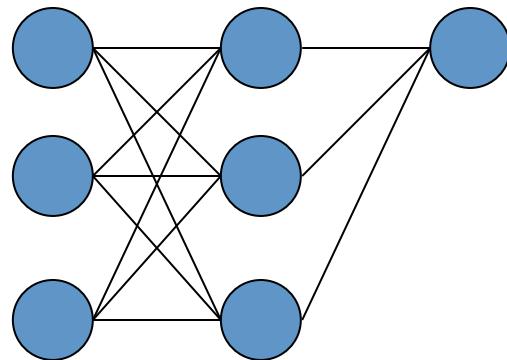
„ Ôǣ Á~ áåÁ] , æå• Á



| Ø^ǣ | ^• Á ~ Á ŠÚ• Á

- „ Ùǟ] | ^ Á~ áåǟ * Á|[& • Á
. Òǣ@ Á| ^ { ^ } d̄ Á• Á Á ^{ & ^ } d[} Á' } Á

- „ Óǣ Á~ áåÁ] , æå• Á



Input Features Layer 1



2-layer:

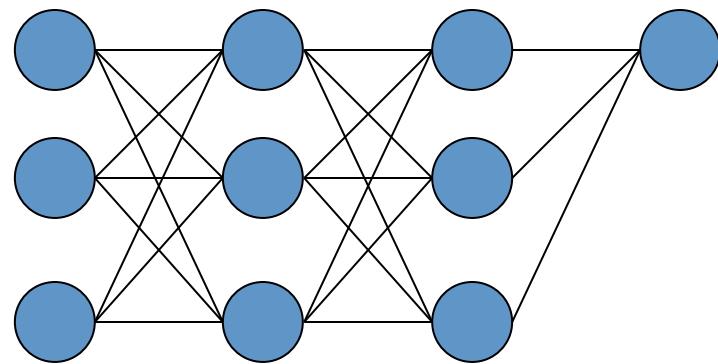
“Features” are now partitions

All linear combinations of those partitions



| Ø^ǣ | ^• Á ~ Á ŠÚ• Á

- „Ùǣ] | ^• Á ~ Á ŠÚ• Á
. Øǣ @ Á | ^{ ^ } d Á • Á Á ^{ & ^ } d [} Á' } Á
- „Øǣ Á ~ Á ŠÚ• Á , ǣ• Á



Input
Features Layer 1 Layer 2

A graph with three horizontal lines representing the activation of three different features over time. The first feature (top line) is active from time 0 to 1. The second feature (middle line) is active from time 1 to 2. The third feature (bottom line) is active from time 2 to 3. This illustrates how complex functions (represented by the network) can be built from simple linear combinations of these features.

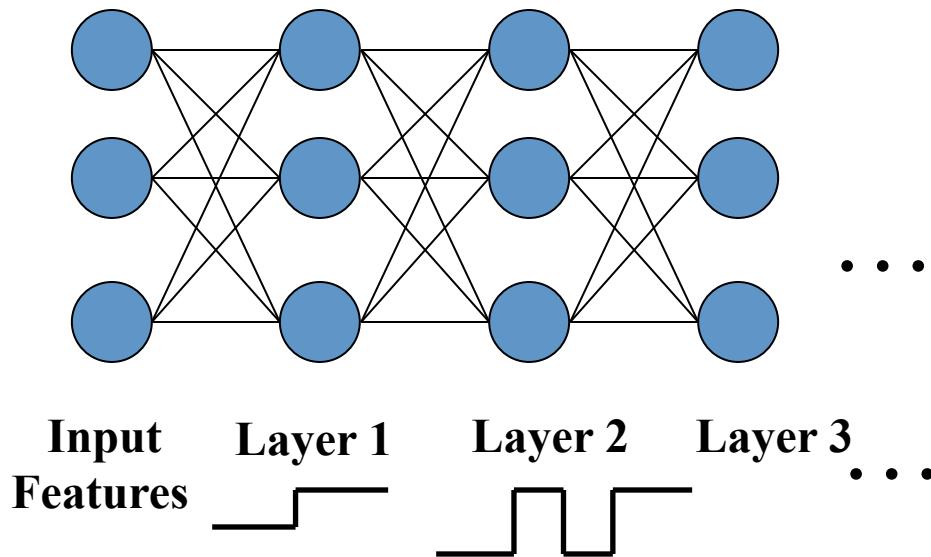
3-layer:

“Features” are now complex functions
Output any linear combination of those

| Ø^ǣ | ^• Á | ~ Á ŠÚ• Á

- „Ùǣ] | ^ Á~ áåǣ * Á ||[& • Á
. Òǣ@ Á| ^ { ^ } d Á• Á Á ^ | & ^] d [} Á' } Á

- „Óǣ Á~ áåÁ] , æå• Á



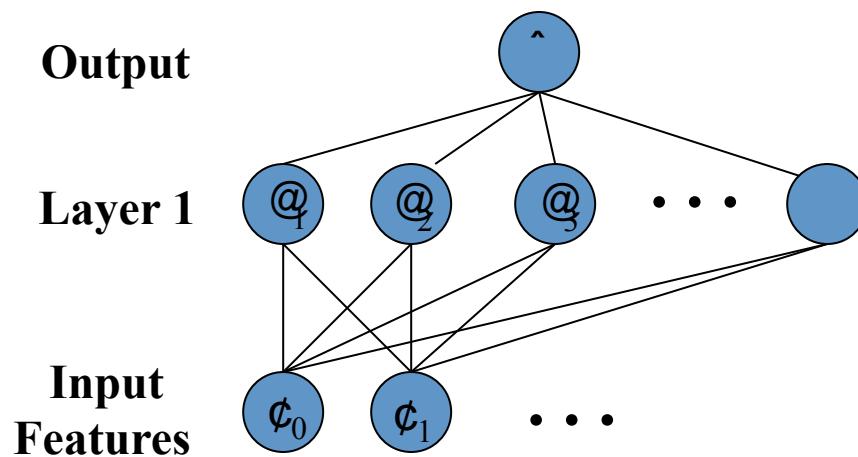
Current research:
“Deep” architectures
(many layers)

$\emptyset \wedge \text{æ} \vee \wedge \bullet \text{A} \sim \text{A} \rightarrow \text{S} \cup \text{A}$

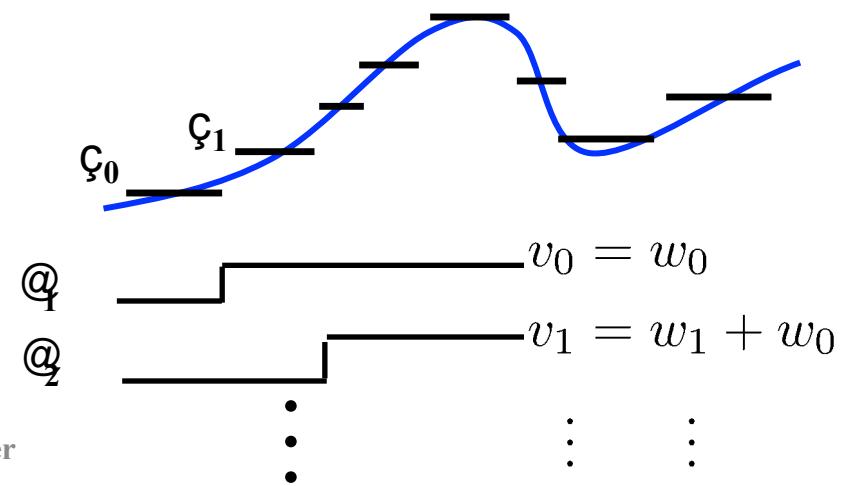
„Už] |^ Á~ áåé * Á|[& • Á
. Óæ&@| ^ { ^ } d Á• Á Á ^ { & } d [} Á } & } Á

„Óæ Á~ áåÁ] , æå• Á

„Ø^cã|^ Á } & } Á } |[cã æ } Á
. Ø } |[cã æ^ Áæàæ^ Á } & } • Á æ@ } [^ * @ Áæå^ } Á [å^ • Á



(c) Alexander Ihler



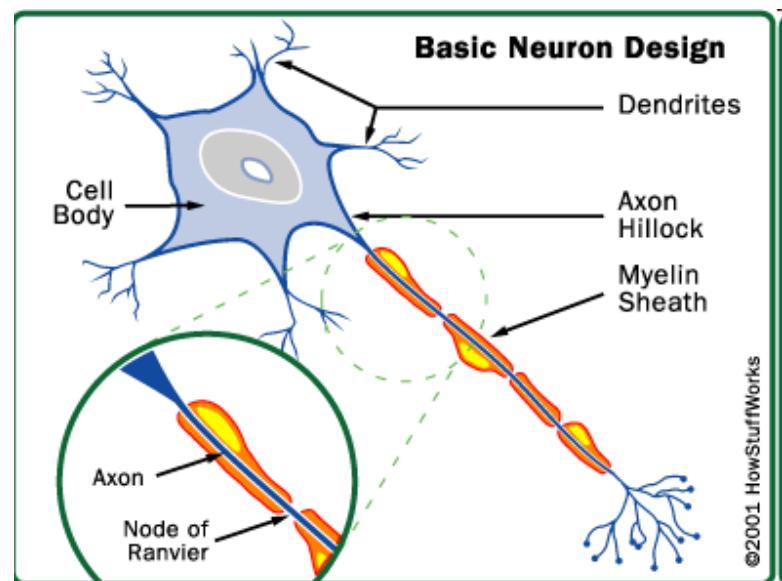
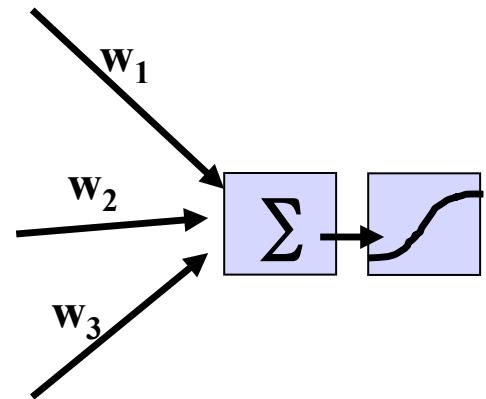
„P[^] | a Á ^ c [\| • Á

„O ß [o @ | Á ^ | { Á | | Á Š U • Á

„Ó á | [* á & a Á [c ã a e á } Á

„P[^] | [} • Á

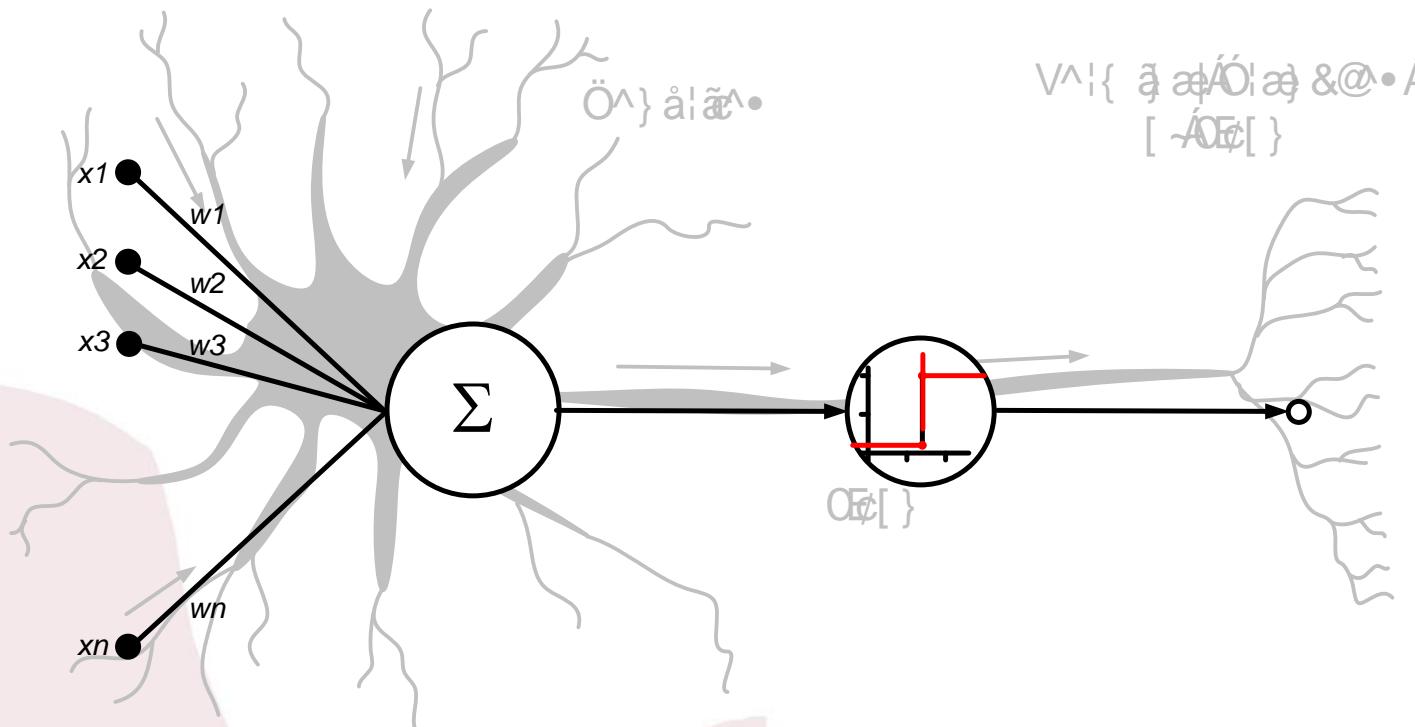
- “Ù á] | ^ ” Á & ^ || • Á
- Ö ^ } á | á ^ • Á • ^ } • ^ Á & @ e ^ * ^ Á
- Ô ^ | Á ^ á @ Á } ^ c Á
- “Ø á ^ • ” Á & [} Á



(c) Alexander Ihler

“How stuff works: the brain”

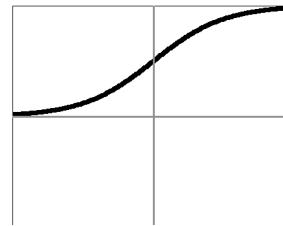
- Ø } 8/ ^• Á^{\wedge} &|æ• ã&as8ë[Á\ä | !æ
- Ø } 8ë[Á^{\wedge} Ác\äças8ë[



Activation Functions

Logistic

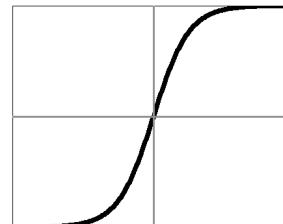
$$\sigma(z) = \frac{1}{1 + \exp(-z)}$$



$$\frac{\partial \sigma}{\partial z}(z) = \sigma(z)(1 - \sigma(z))$$

Hyperbolic Tangent

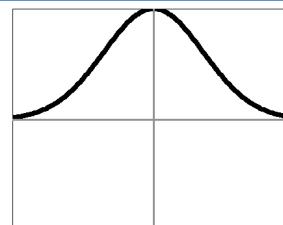
$$\sigma(z) = \frac{1 - \exp(-2z)}{1 + \exp(-2z)}$$



$$\frac{\partial \sigma}{\partial z}(z) = 1 - (\sigma(z))^2$$

Gaussian

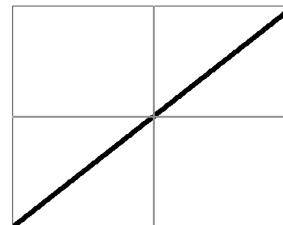
$$\sigma(z) = \exp(-z^2/2)$$



$$\frac{\partial \sigma}{\partial z}(z) = -z\sigma(z)$$

Linear

$$\sigma(z) = z$$



$$\frac{\partial \sigma}{\partial z}(z) = 1$$

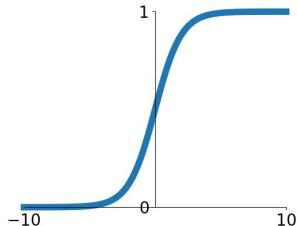
And many others...

(c) Alexander Ihler

Operations } AND } & OR •

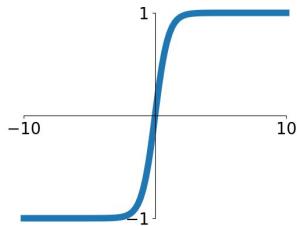
Sigmoid

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



tanh

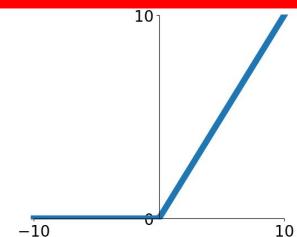
$$\tanh(x)$$



ReLU

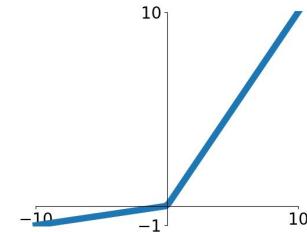
$$\max(0, x)$$

Good default choice



Leaky ReLU

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

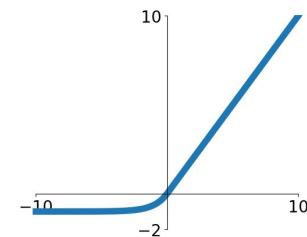


Maxout

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

ELU

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



Ø^åE[], æåÅ^ç []\bullet

- Q{ { æ } Á[, • Á^æ E ß @
- Q] ^ à•^|ç^åÁ^æ |^•Á
- Ô[{] ^ c Á^æ å^} Á[å^•Á^æ æ|D Á
- Ô[{] ^ c Á^æ å^|o Á

```

x1 = _add1(X);      # add constant feature
T  = X1.dot(W[0].T); # linear response
H  = Sig( T );       # activation f'n

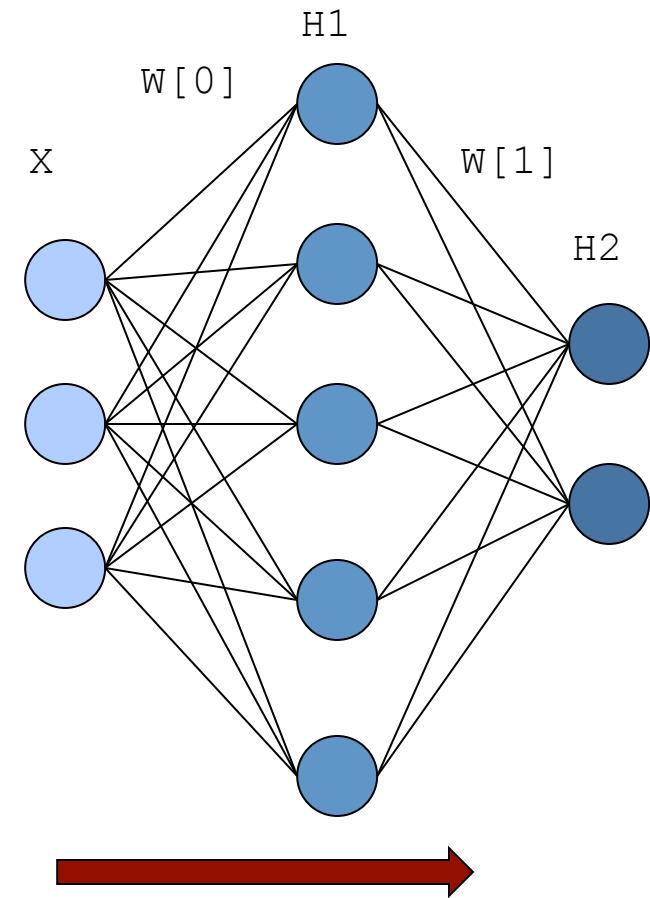
H1 = _add1(H);      # add constant feature
S  = H1.dot(W[1].T); # linear response
H2 = Sig( S );       # activation f'n

%
```

...

- OE^!} æç^K^& ||^} ØP•o Á

(c) Alexander Ihler



Ø^åÈ[; æåÅ^ç [\•Á

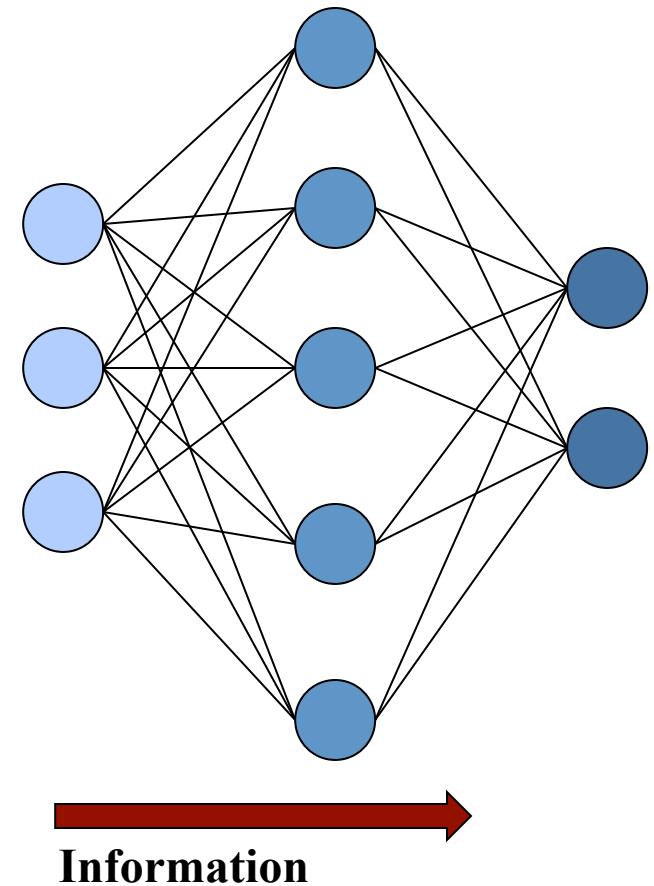
OEÁ [c^Á } Á̄ ^ |ç̄ | ^ Á̄ ^ q̄ ^ c̄ k̄

„Ü^{Λ*} |Λ••ã } KÁ

- Ú!^å&öÁ ^|cäää Á ^}•ä } äÁ Á
%`j@e^å+Á^] |^•^} cæä } Á
ÁMÁ^, ^|Á ääé ^c^|•Á

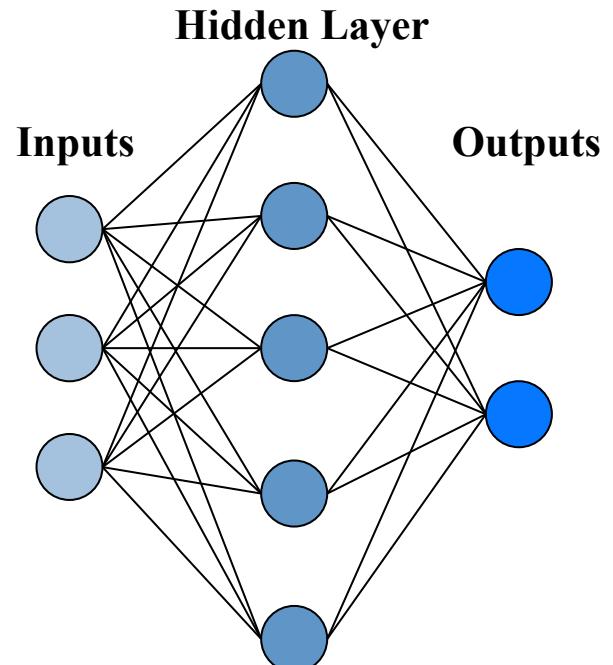
“Ô|æ•ã&ææ } Á

- Ú!^å&ø Ášk æ^ Áç^&q iÁ
 - T ^ |ç&æ • Á&æ • á&æææ } Á
 - Á ÁMÁGÁMÁSÉÁFÁEÁS ÁÁ
 - T ^ |çv | ^ Éñk ãk ãk æ^ Á i^å&q } • Á
 - Áçk æ^ Áæ* ãk * ÉA &ÉÁ
 - U c^ } Áæ ãk ^ å Áæ Á^* i^•• ã } Áç UÙ
 - Á ã@ Áæ |ææ* Á&cæææ } Á



VÍAS ESTATÍSTICAS

- „ Užití výpočtu pravděpodobnosti pro řešení klasifikačních úloh
- „ Účinnost je závislá na výběru hodnoty parametrů
- „ Obrázek: Různé funkce ztráty (Can use different loss functions if desired...)
- „ $P[\hat{y} | x] = \frac{1}{1 + e^{-(w^T x + b)}}$
- „ Úkoly s využitím klasifikace:
 - . Šířka a výška obrazu
 - . Účinnost
- „ Ukládání modelu do souboru
- „ Účinnost je závislá na výběru hodnoty parametrů



(c) Alexander Ihler

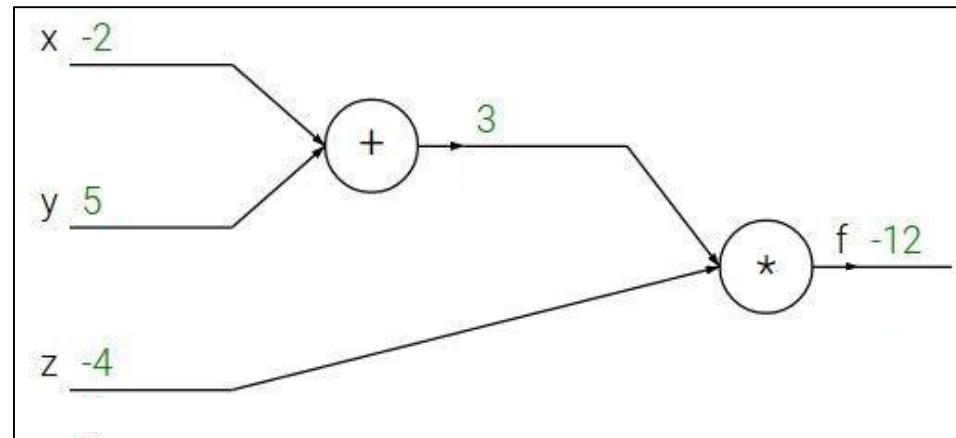
Treinamento de redes MLP

- Ó[áéíôõ] á^ Áçæßåå^ Á^ ÁDE* [iã[•
 - . Ó^|ã[^} c^ Á^] ^|çã[} æ[•
 - . Ò• a cã[•
 - " Não alteram ^• d^ c |ã^ Á^ á^
 - " Óæ\] i[] æ^ æ\ } ÊÓ^ } 8ë[Á^ Áæ^ ÁÜæ^ æ\
 - . Ô[} • d^ cã[•
 - " ØÈ^|ã[Á• d^ c |ã^ Á^ á^
 - " W] • cæ ÈÓæ &æ^ ÁÔ[||^|æ\ }



$$f(x, y, z) = (x + y)z$$

^ Èc MEGE^ MÍ È: MË



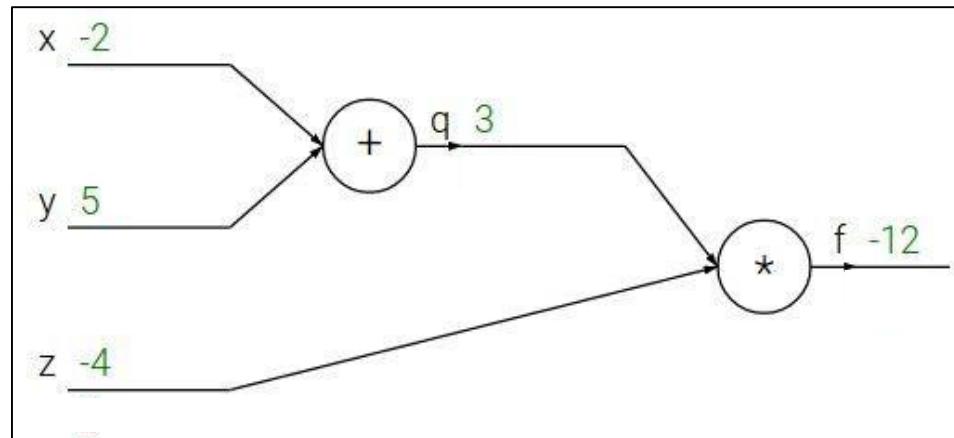
$$f(x, y, z) = (x + y)z$$

\wedge Èc MÈGÈ MÍ È MË

$$q = x + y \quad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \quad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

$$\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$$



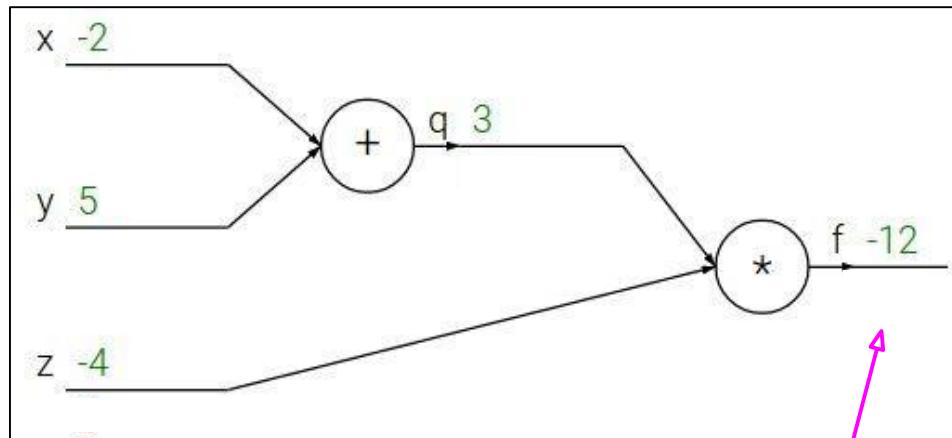
$$f(x, y, z) = (x + y)z$$

\wedge Èc MÈGÈ MÍ È MË

$$q = x + y \quad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \quad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

$$\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$$



$$\frac{\partial f}{\partial f}$$

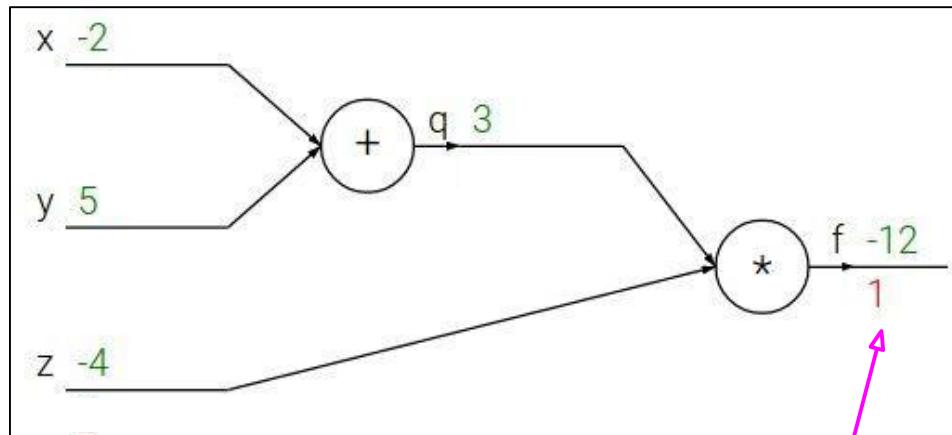
$$f(x, y, z) = (x + y)z$$

\wedge Èc MÈGÈ MÍ È MË

$$q = x + y \quad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \quad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

$$\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$$



$$\frac{\partial f}{\partial f}$$

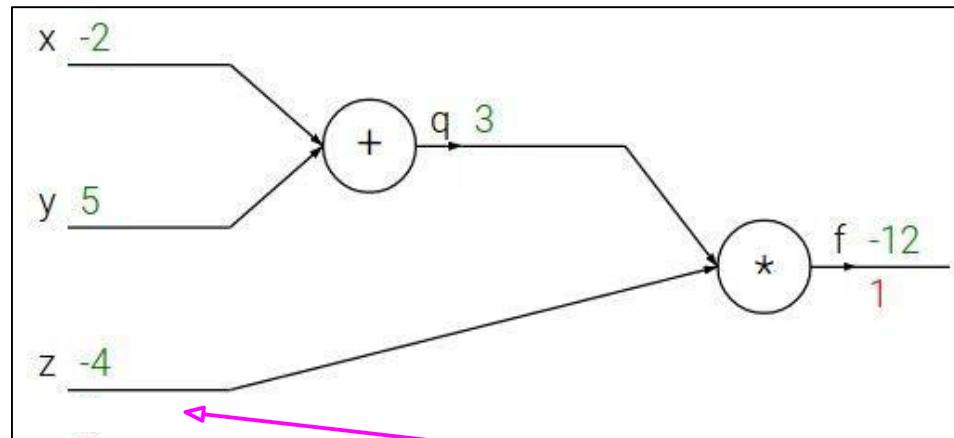
$$f(x, y, z) = (x + y)z$$

\wedge Èc MÈGÈ MÍ È MË

$$q = x + y \quad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \quad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

$$\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$$



$$\frac{\partial f}{\partial z}$$

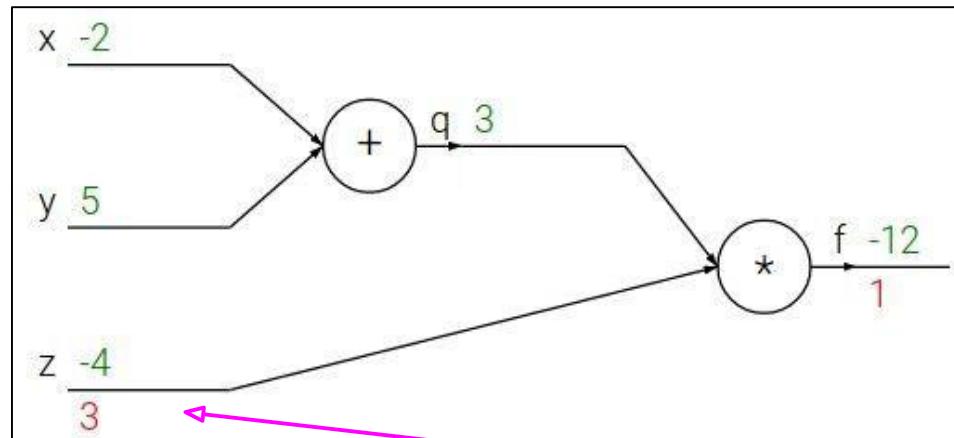
$$f(x, y, z) = (x + y)z$$

\wedge Èc MÈGÈ MÍ È MË

$$q = x + y \quad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \quad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

$$\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$$



$$\frac{\partial f}{\partial z}$$

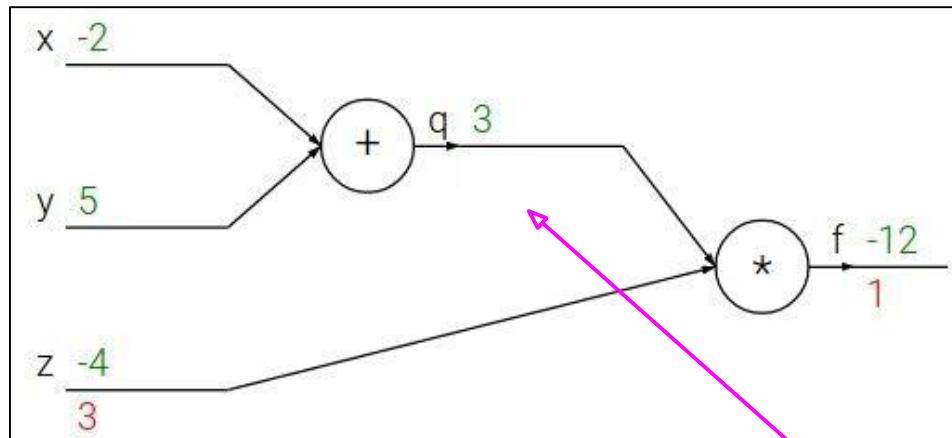
$$f(x, y, z) = (x + y)z$$

\wedge Èc MÈGÈ MÍ È MË

$$q = x + y \quad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \quad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

$$\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$$



$$\frac{\partial f}{\partial q}$$

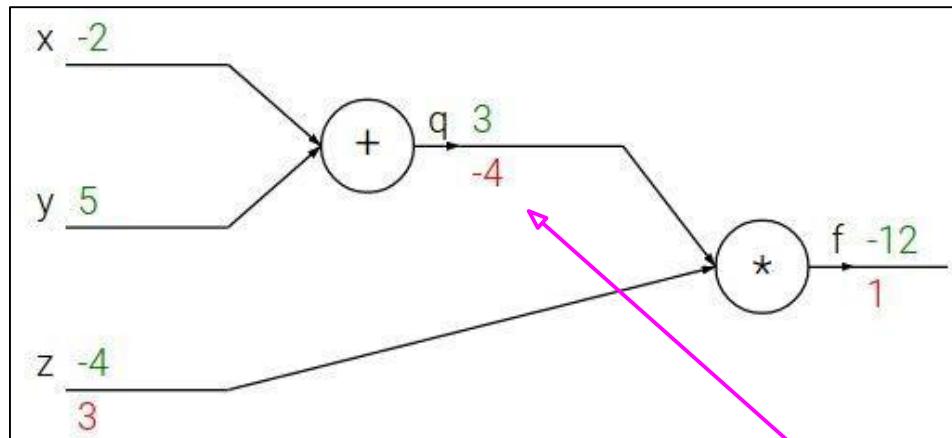
$$f(x, y, z) = (x + y)z$$

\wedge Èc MÈGÈ MÍ È MË

$$q = x + y \quad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \quad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

$$\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$$



$$\frac{\partial f}{\partial q}$$

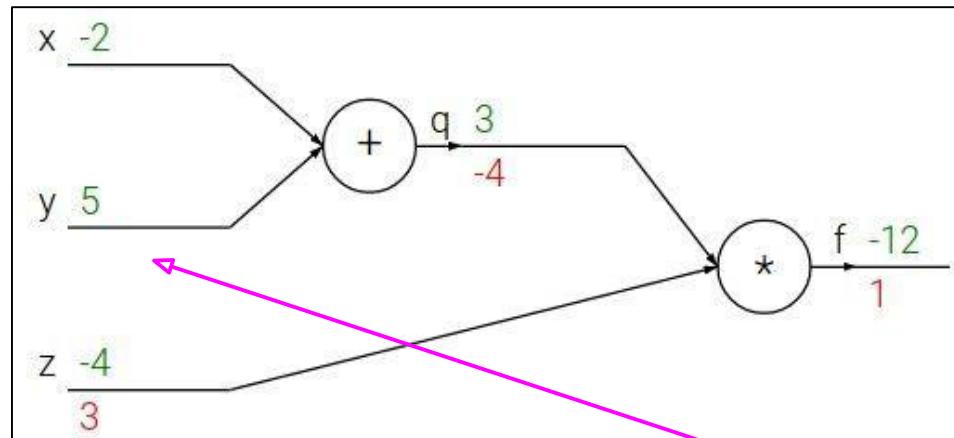
$$f(x, y, z) = (x + y)z$$

\wedge Èc MÈGÈ MÍ È MË

$$q = x + y \quad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \quad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

$$\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$$



$$\frac{\partial f}{\partial y}$$

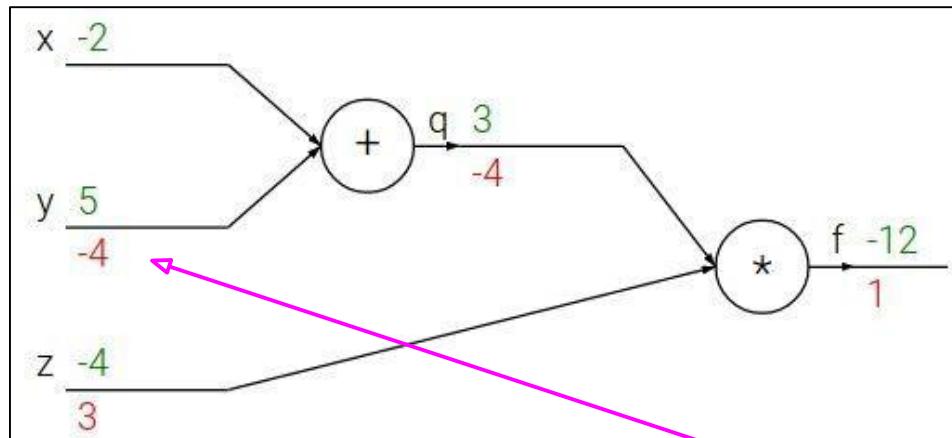
$$f(x, y, z) = (x + y)z$$

^ Èc MEGE MÍ Ê: MË

$$q = x + y \quad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \quad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

$$\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$$



O@e |^ |^ K

$$\frac{\partial f}{\partial y} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial y}$$

$$\frac{\partial f}{\partial y}$$

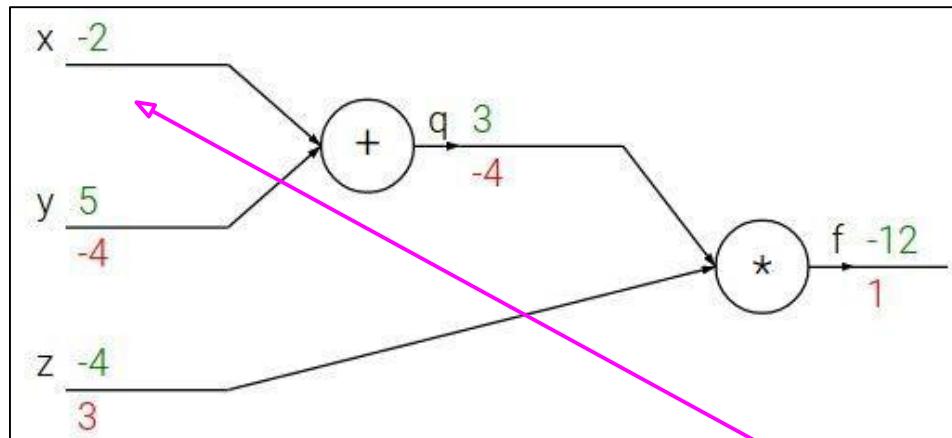
$$f(x, y, z) = (x + y)z$$

\wedge Èc MÈGÈ MÍ È MË

$$q = x + y \quad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \quad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

$$\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$$



$$\frac{\partial f}{\partial x}$$

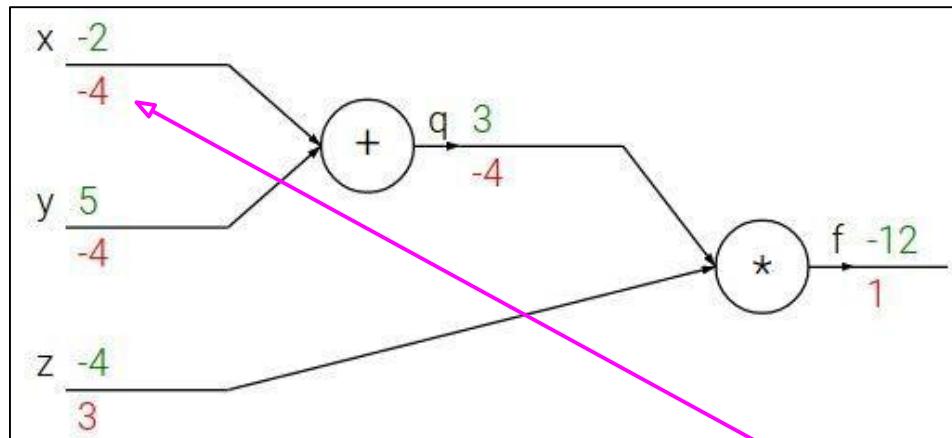
$$f(x, y, z) = (x + y)z$$

^ Èc MEGE MÍ Ê MË

$$q = x + y \quad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \quad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

$$\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$$

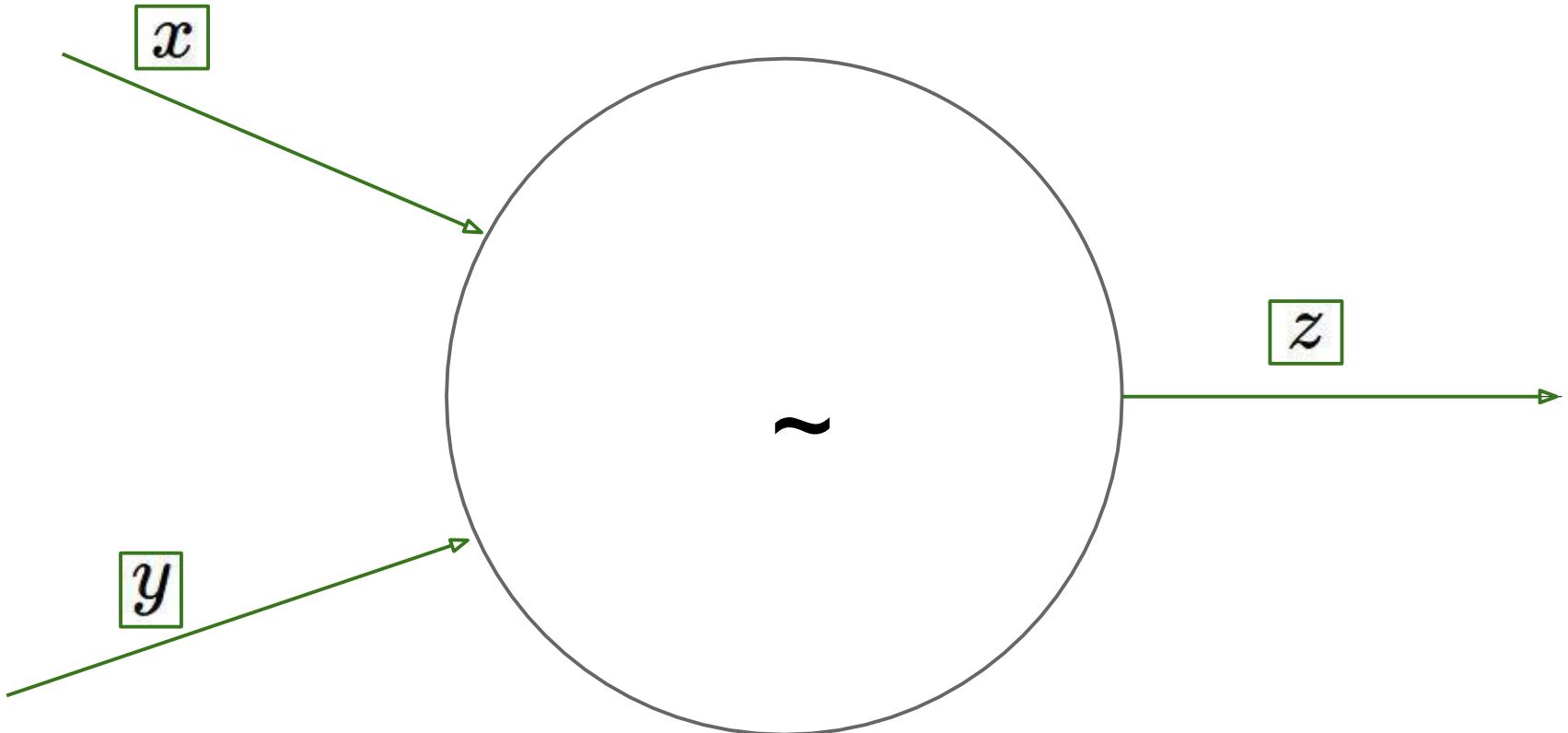


Ô@ë |^ |^ K

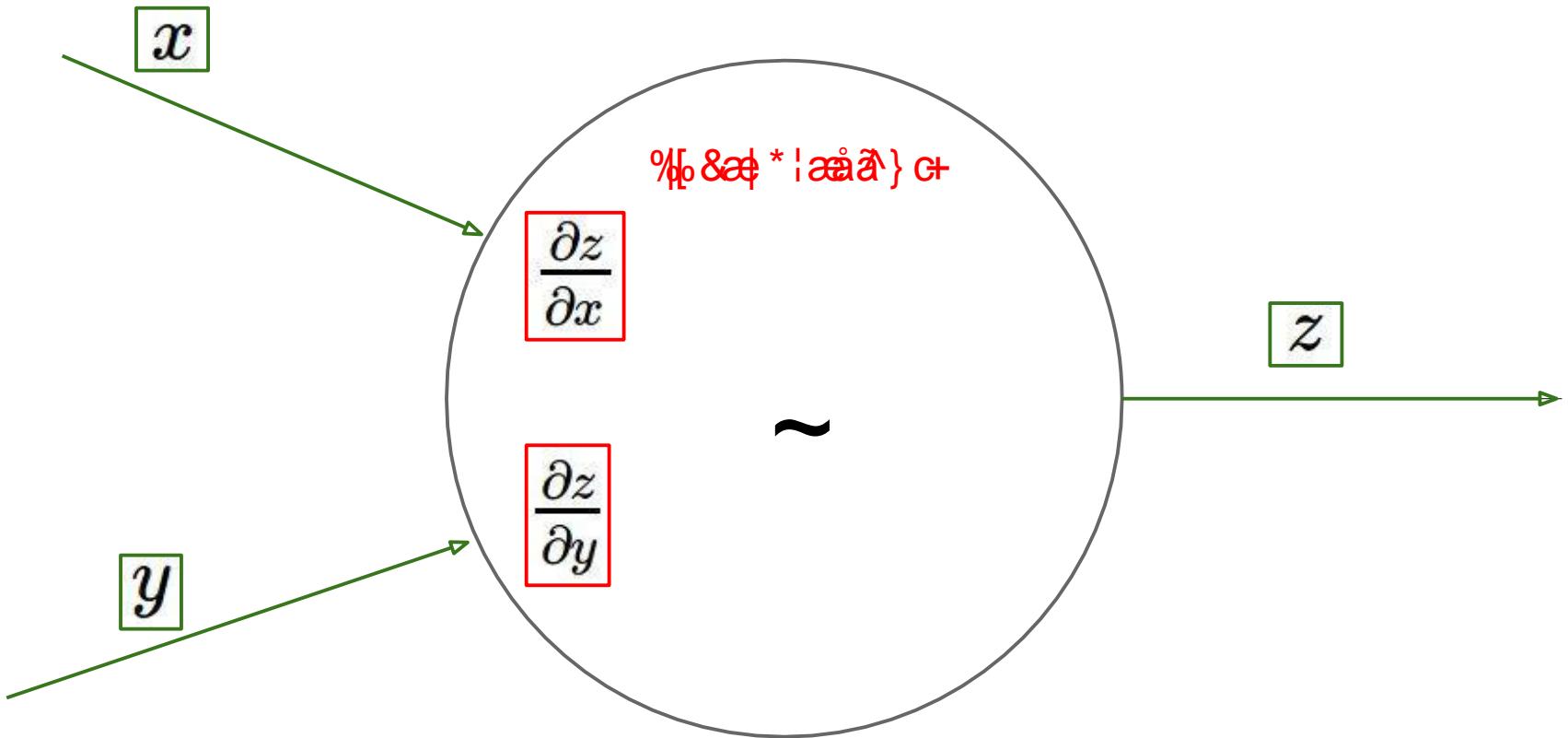
$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial x}$$

$$\frac{\partial f}{\partial x}$$

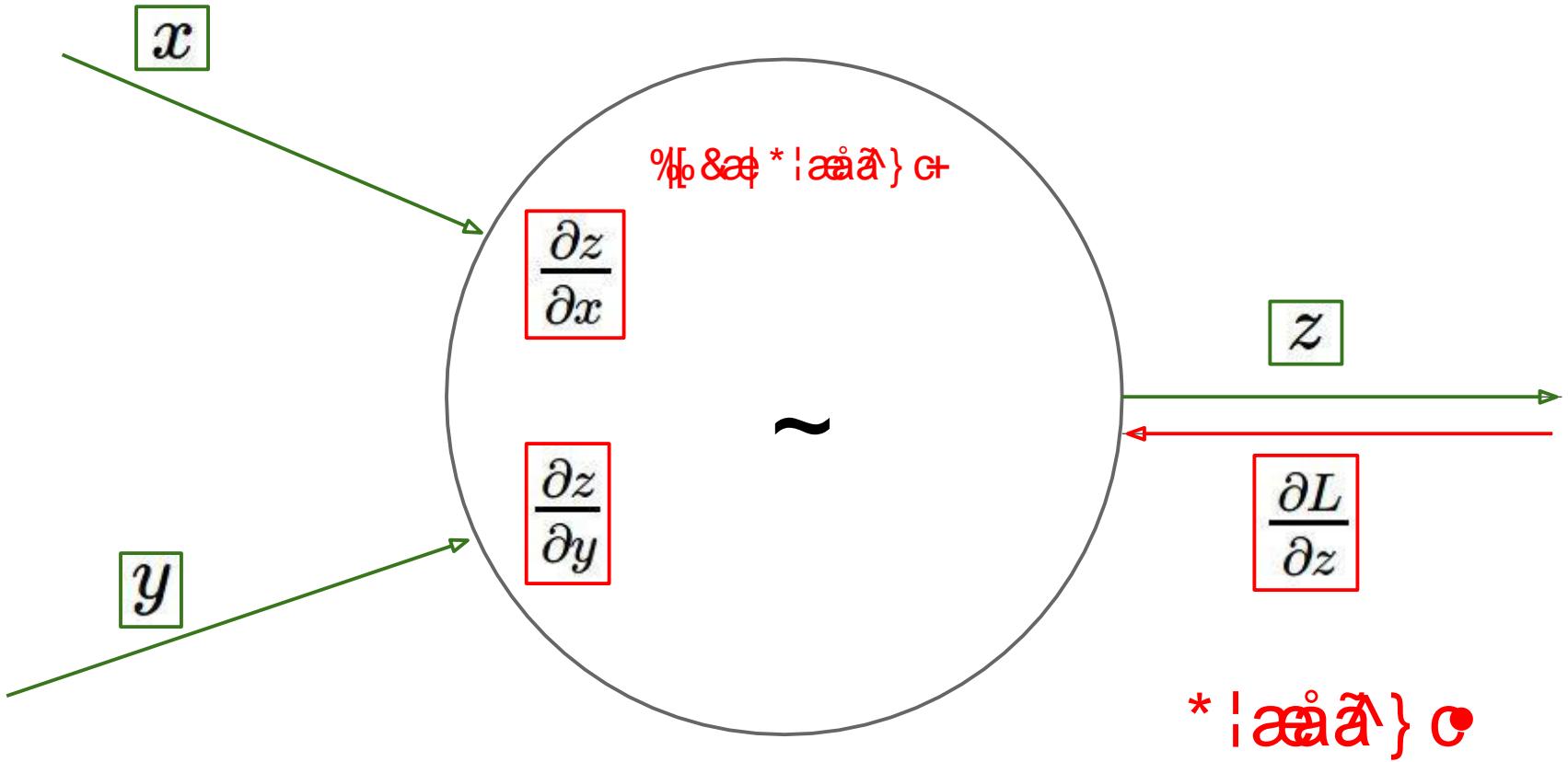
अंतर्गत } •



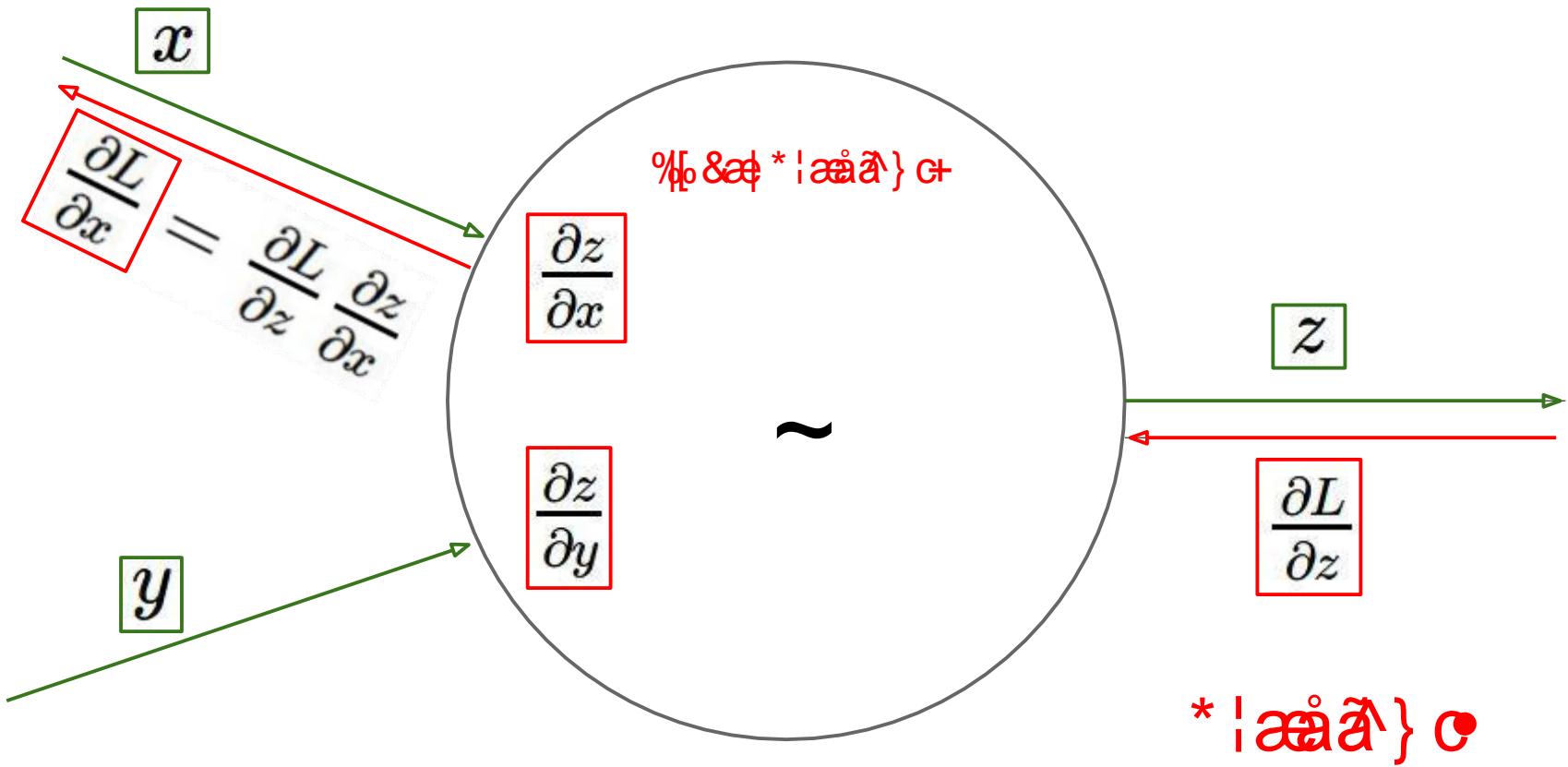
અન્દું અન્દી } •



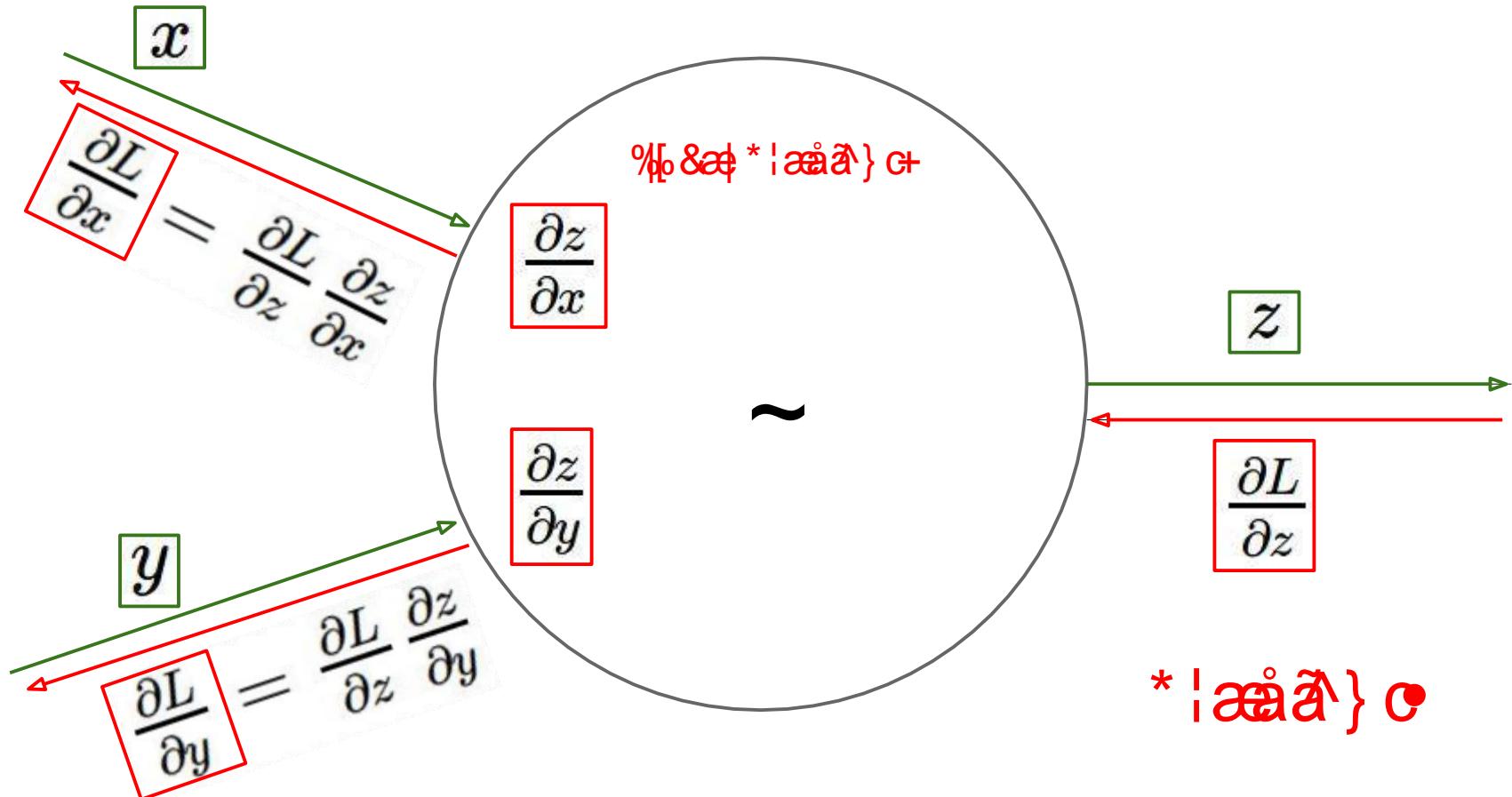
અન્દું અન્દી } •



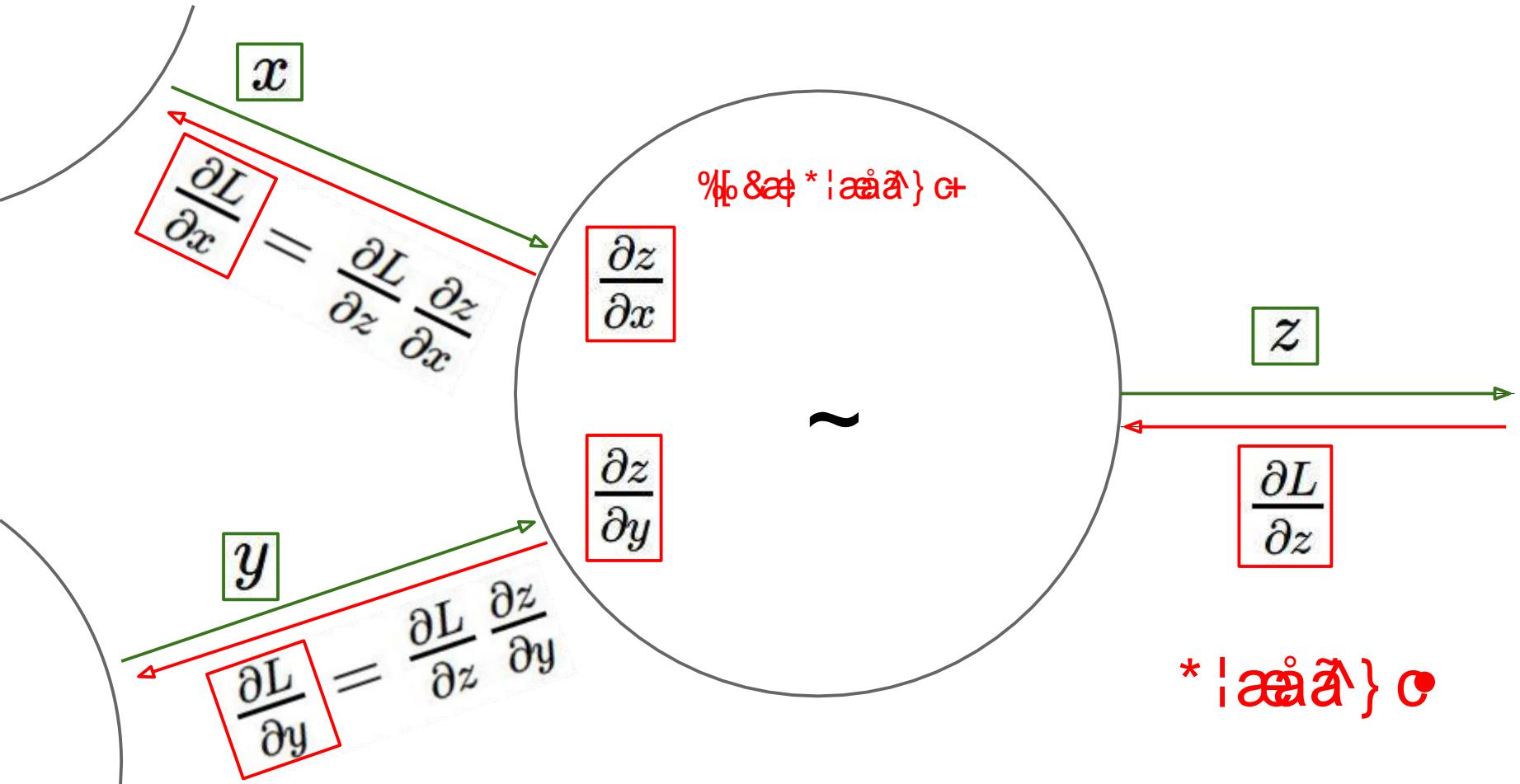
અન્દું આવે } •



અંગ્રેજી } •



અંગેંજેન્યુ } •



| Óæ&\] :[] æ* ææ\ } Á

„ R • o Á ;ææ\ } o Á ^• &^} o Á

„ O@] | ^ Á @ Á @ e Á ^ | ^ Á @ Á @ Á T ŠÚÁ

$$\frac{\partial J}{\partial w_{kj}^2} = -2 \sum_{k'} (y_{k'} - \hat{y}_{k'}) (\partial \hat{y}_{k'})$$

= $-2(y_k - \hat{y}_k) \sigma'(s_k) h_j$ (Identical to logistic mse regression with inputs “@”)

Forward pass

Loss function

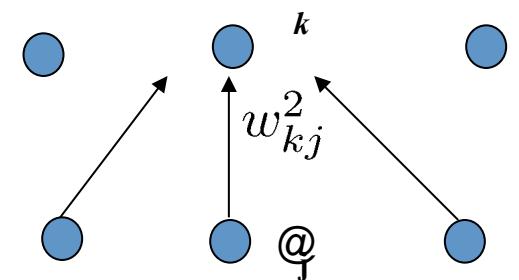
$$J_i(W) = \sum_k (y_k^{(i)} - \hat{y}_k^{(i)})^2$$

Output layer

$$\hat{y}_k = \sigma(s_k) = \sigma(\sum_j w_{kj}^2 h_j)$$

Hidden layer

$$h_j = \sigma(t_j) = \sigma(\sum_i w_{ji}^1 x_i)$$



Óæ&\] :[] æ* ææ\ } Á

- „ R • o Á !ææ\ } o Á ^• &^} ø
- œ] | ^ Á @ Á @ œ Á ^ Á @ Á @ Á ŠÚ

$$\frac{\partial J}{\partial w_{kj}^2} = -2 \sum_{k'} (y_{k'} - \hat{y}_{k'}) (\partial \hat{y}_{k'})$$

$$= -2(y_k - \hat{y}_k) \sigma'(s_k) h_j$$

$$\beta_k^2$$

$$\frac{\partial J}{\partial w_{ji}^1} = \sum_k -2(y_k - \hat{y}_k) (\partial \hat{y}_k)$$

$$= \sum_k -2(y_k - \hat{y}_k) \sigma'(s_k) w_{kj}^2 \partial h_j$$

$$= \sum_k -2(y_k - \hat{y}_k) \sigma'(s_k) w_{kj}^2 \sigma'(t_j) x_i$$

β_k^2

(c) Alexander Ihler

Forward pass

Loss function

$$J_i(W) = \sum_k (y_k^{(i)} - \hat{y}_k^{(i)})^2$$

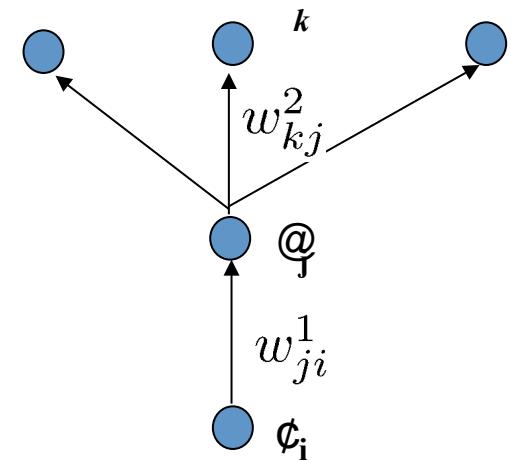
Output layer

$$\hat{y}_k = \sigma(s_k) = \sigma(\sum_j w_{kj}^2 h_j)$$

Hidden layer

$$h_j = \sigma(t_j) = \sigma(\sum_i w_{ji}^1 x_i)$$

(Identical to logistic mse regression with inputs “@”)



Forward pass

Loss function

$$J_i(W) = \sum_k (y_k^{(i)} - \hat{y}_k^{(i)})^2$$

Output layer

$$\hat{y}_k = \sigma(s_k) = \sigma(\sum_j w_{kj}^2 h_j)$$

Hidden layer

$$h_j = \sigma(t_j) = \sigma(\sum_i w_{ji}^1 x_i)$$

```
% X : (1xN1)
H = Sig(X1.dot(W[0]))
% W1 : (N2 x N1+1)
% H : (1xN2)
Yh = Sig(H1.dot(W[1]))
% W2 : (N3 x N2+1)
% Yh : (1xN3)
```

$$B2 = (Y - Yhat) * dSig(S) \quad \# (1xN3)$$

$$G2 = B2.T.dot(H) \quad \# (N3x1) * (1xN2) = (N3xN2)$$

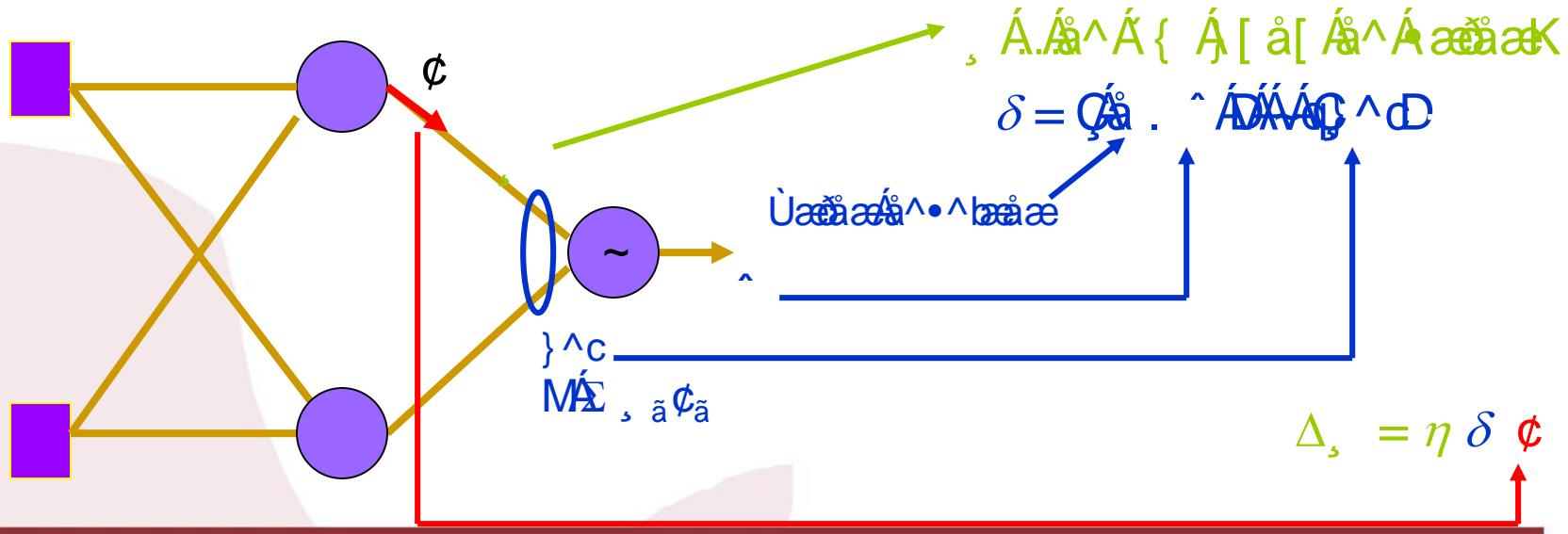
$$B1 = B2.dot(W[1]) * dSig(T) \quad \# (1xN3) . (N3*N2) * (1xN2)$$

$$G1 = B1.T.dot(X) \quad \# (N2 x N1+1)$$

Óa& ^ Áo&, æ åKDE • caí • Á ^• [• ÁaÁ^å^ÁA æcÁ^æ^æ æ^æ^A æ^æ^E

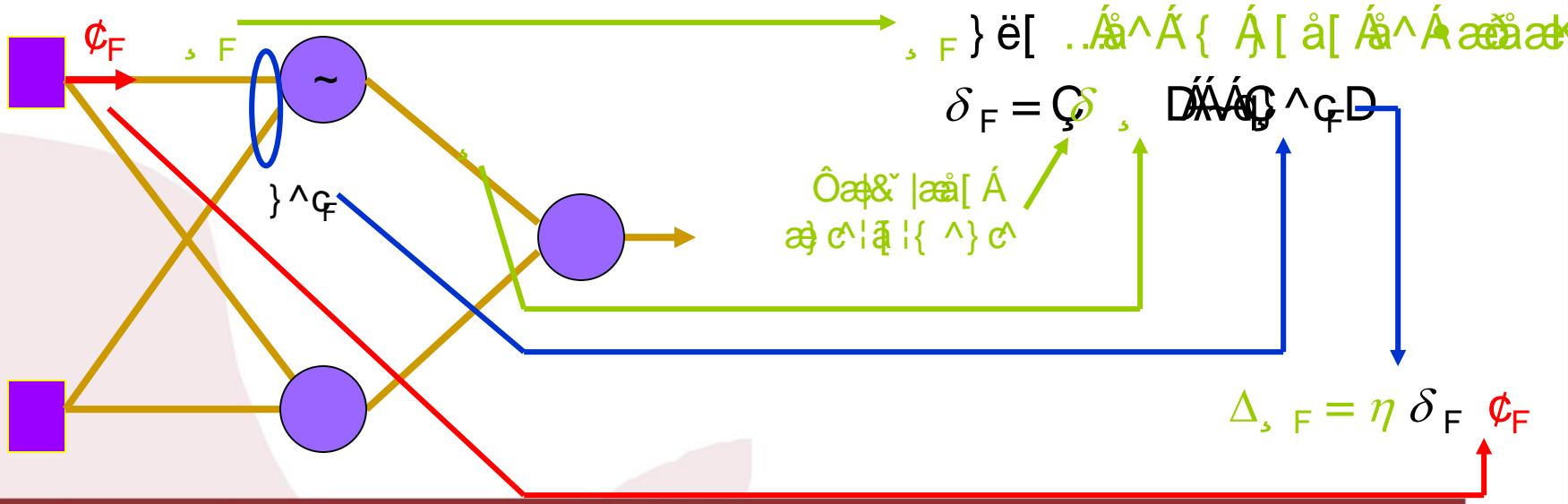
$$\Delta w_{ji} = \eta \delta_j(t) x_i(t)$$

onde $\delta_j(t) = \begin{cases} (d_j - y_j) f'(net_j), & \text{se for nodo de saída} \\ (\sum_l \delta_l w_{lj}) f'(net_j), & \text{caso contrário} \end{cases}$



$$\Delta w_{ji} = \eta \delta_j(t) x_i(t)$$

onde $\delta_j(t) = \begin{cases} (d_j - y_j) f'(net_j), & \text{se for nodo de saída} \\ (\sum_l \delta_l w_{lj}) f'(net_j), & \text{caso contrário} \end{cases}$



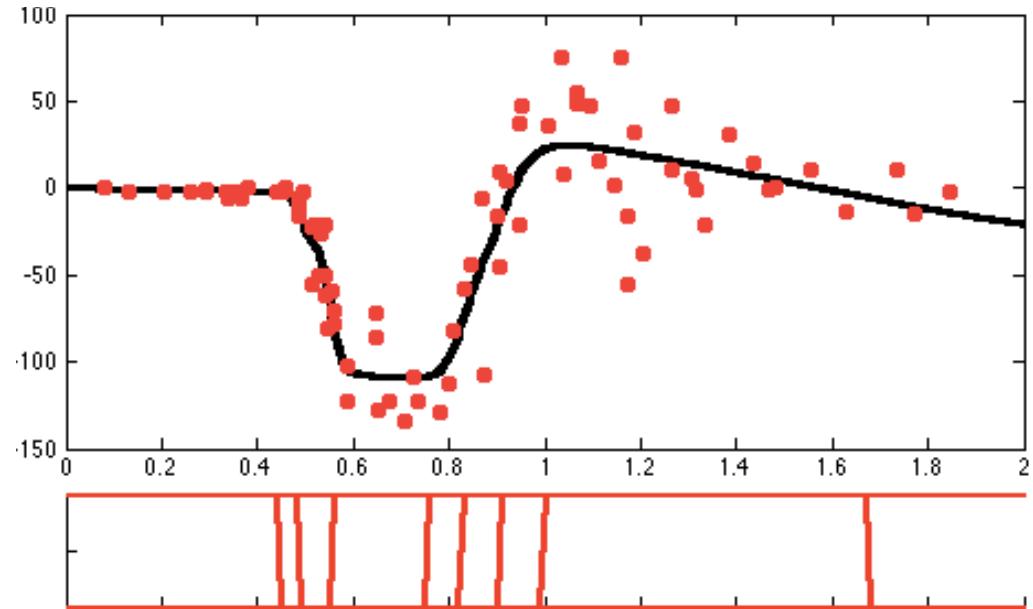
- **O que é a função específica:**
 - **O camada de saída** constrói o padrão
 - **As camadas intermediárias** extratoras de características
 - **Uma classe** tem sua própria representação

Example: Regression, MCycle data

Train NN model, 2 layer

- . 1 input features => 1 input units
- . 10 hidden units
- . 1 target => 1 output units
- . Logistic sigmoid activation for hidden layer, linear for output layer

Data:
+
learned prediction f'n:



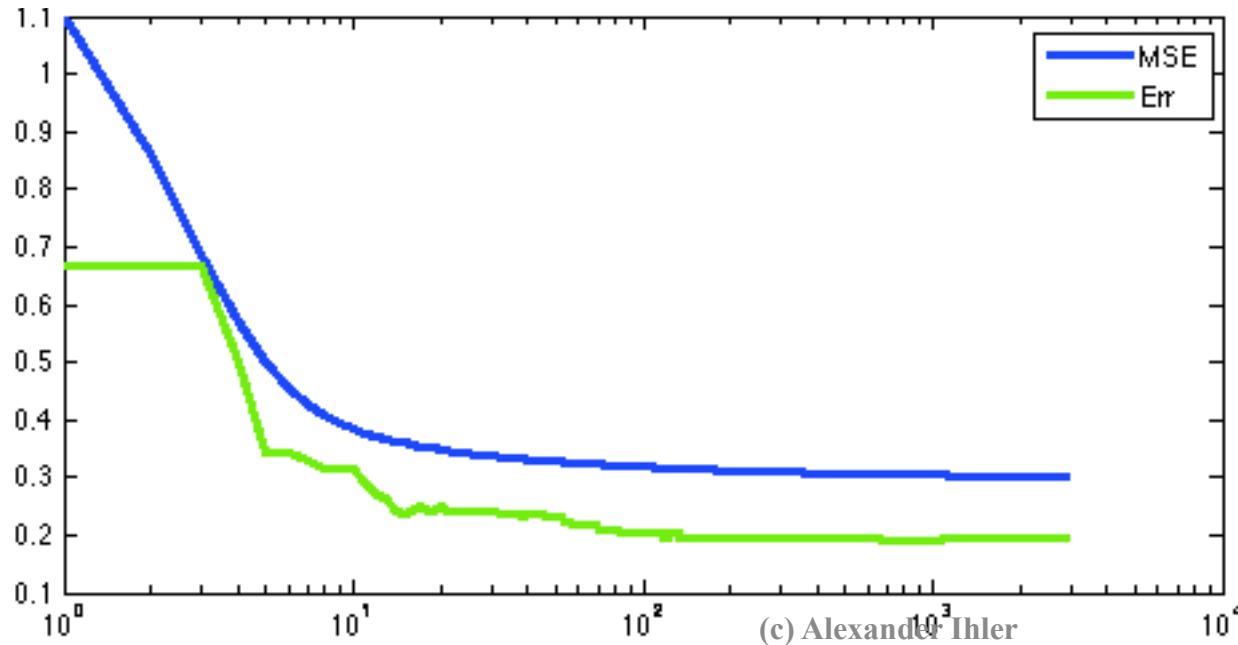
Responses of hidden nodes
(= features of linear regression):
select out useful regions of "x"

(c) Alexander Ihler

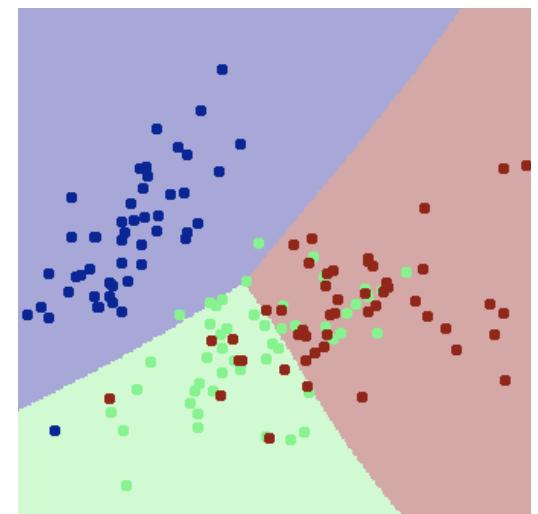
Example: Classification, Iris data

Train NN model, 2 layer

- . 2 input features => 2 input units
- . 10 hidden units
- . 3 classes => 3 output units ($y = [0 0 1]$, etc.)
- . Logistic sigmoid activation functions
- . Optimize MSE of predictions using stochastic gradient



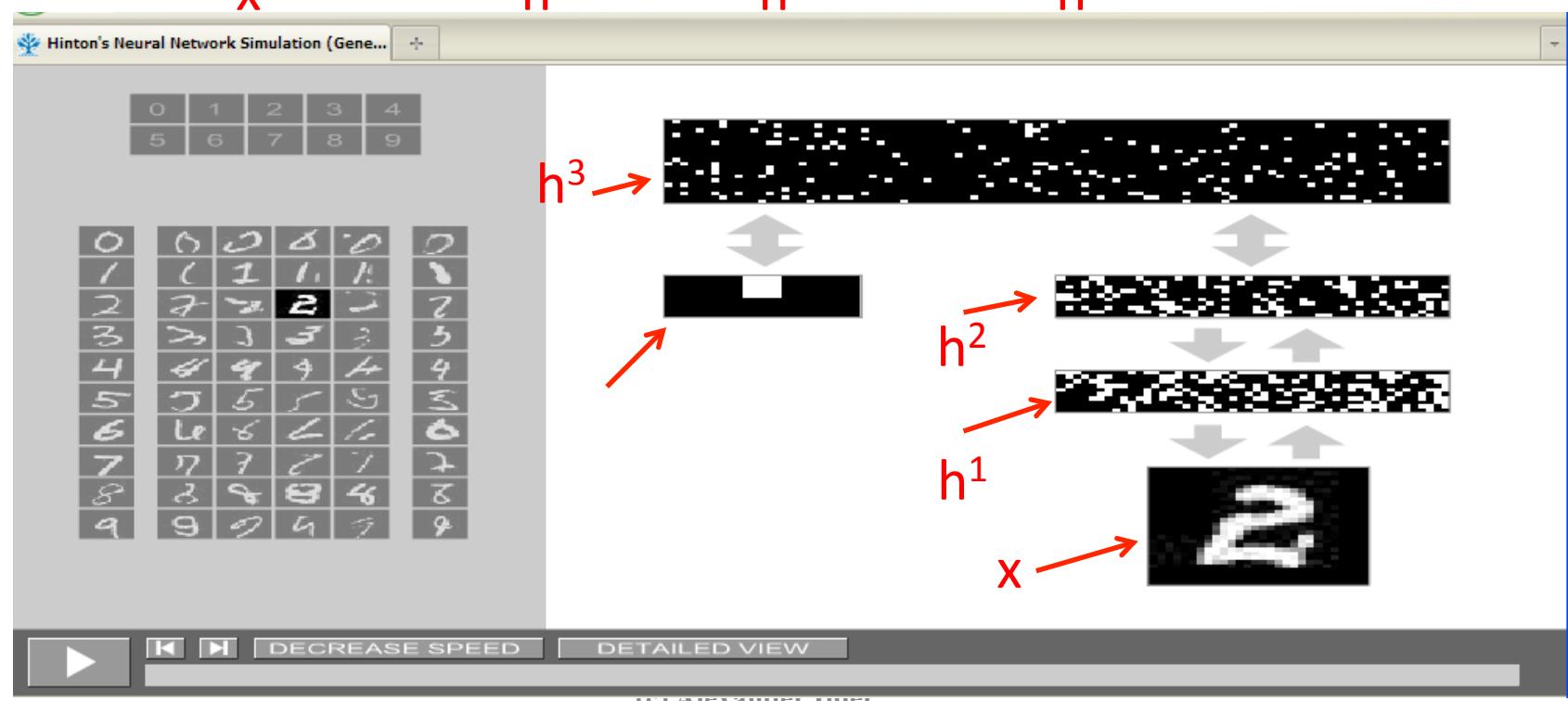
(c) Alexander Ihler



T ŠÚ•Áš Áí a&c&a^ Á

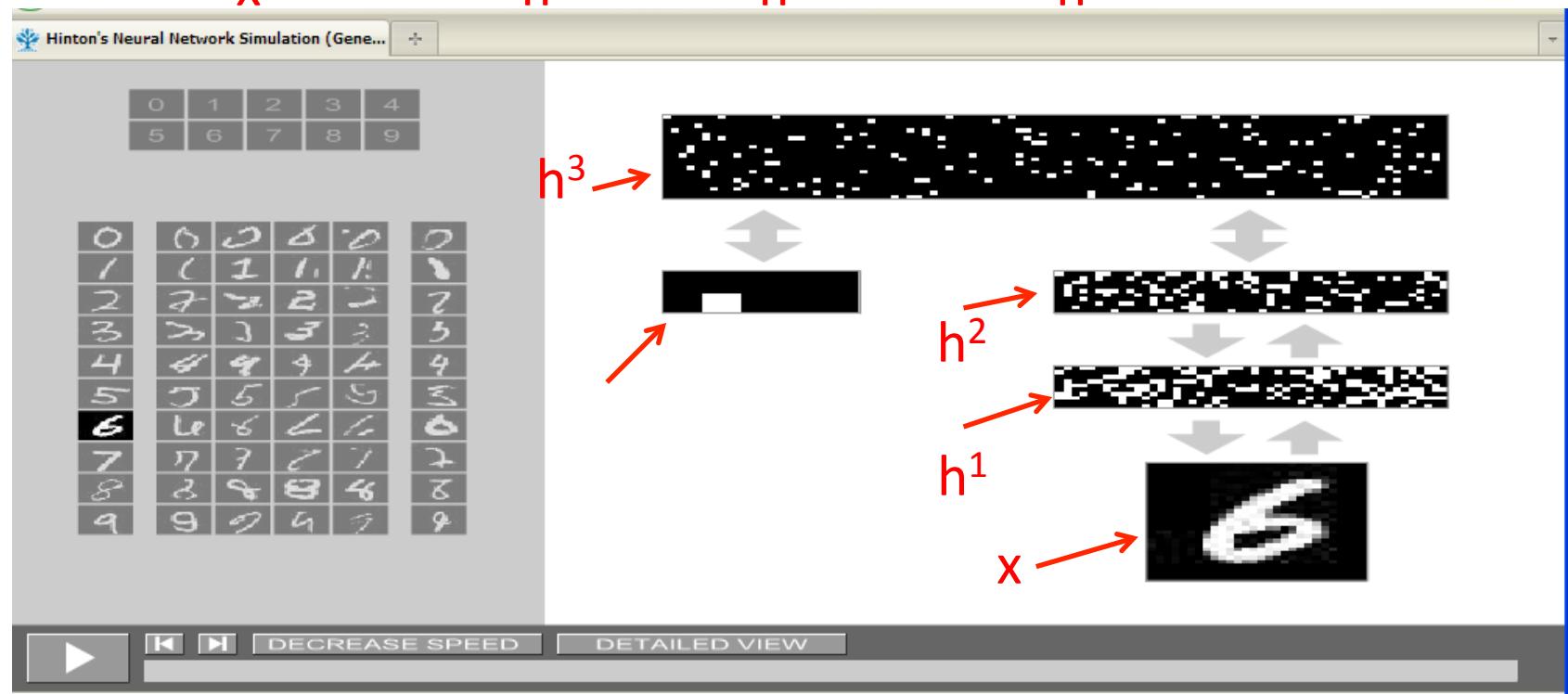
Òcæg] | ^ KÖ^ ^] Á^ | ã Á ^ c Áç p ã d { } Á ã d E G E C I D

- Pæs å, |æs * Á^&[* } æs }
 - U} |æ^Á^&{ [
 - í i | Á æ^|• Á⇒Á €€Á æ Á⇒Á €€Á@ @Á⇒Á€€€Á] Á⇒Á€€Áæ^|•
x h¹ h² h³



T ŠÚ• Áš Áš | a&c&^Á

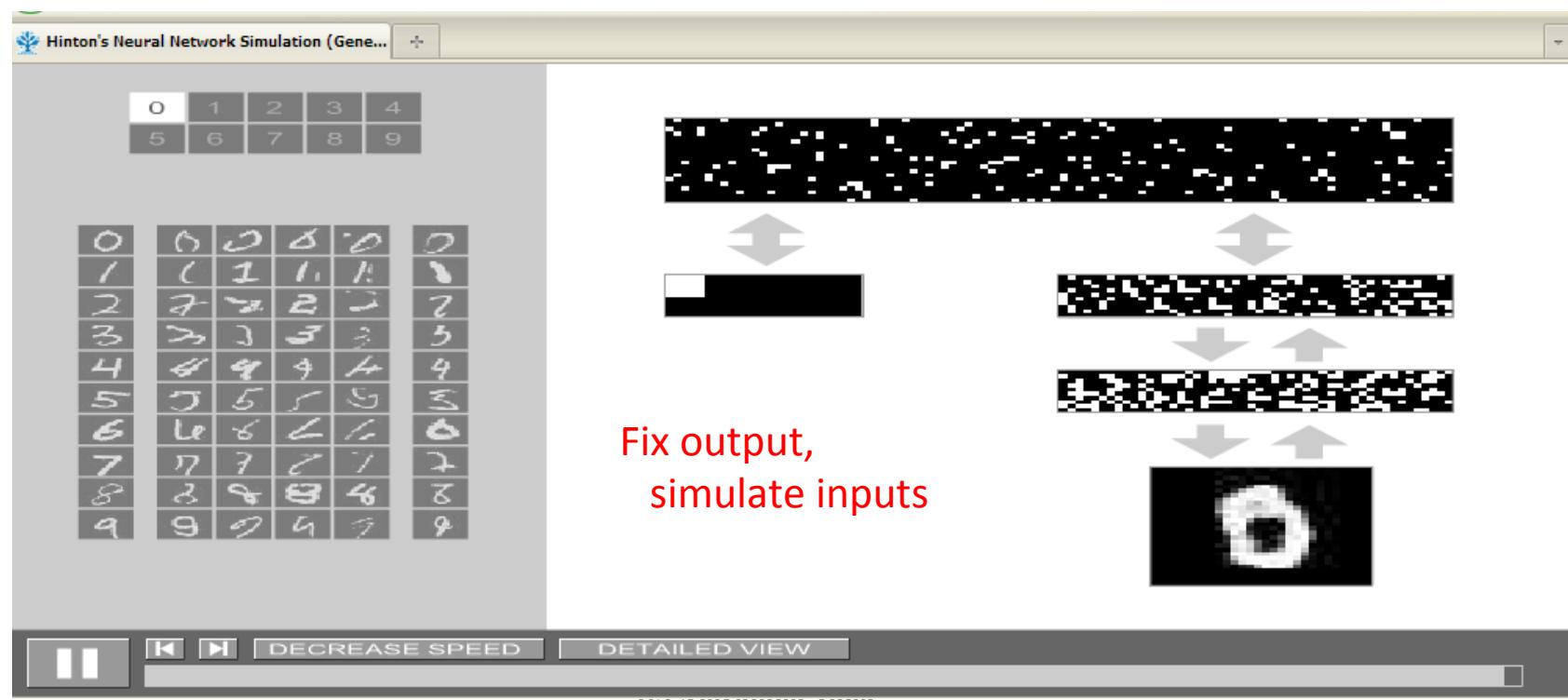
- Ócæ[] |^KÖ^[[Á^|â Á^c Á^P Þ Þ{ } Á Á Þ G E Þ D Á
 - . P æ å, |ã* Á^& * } ã } Á
 - . U} |ã ^ Á^ { [Á
 - . Í Í | Á ã | • Á Þ Á Þ Á Þ Á Þ Á Þ Á Þ Á Þ Á Þ Á Þ Á
- x h^1 h^2 h^3



T ŠÚ• Áš Áš | a&c&^Á

Ócæ[]|^KÖ^] Á^|á~ Á^c Á^P á{ } Á^d Á^G e D

- . Pæ å, |ã* Á^& * } ã }
- . U} |ã^ Á^ { [
- . i i | Á^ |• Á^ |•



■ K-NN

• K-Nearest Neighbors Classificação

■ Árvore de decisão

• Decision Tree Classificação

■ Random Forest demo

• Random Forest Classificação

■ Support Vector Machine in Javascript

• SVM Classificação

■ Artificial Neural Networks

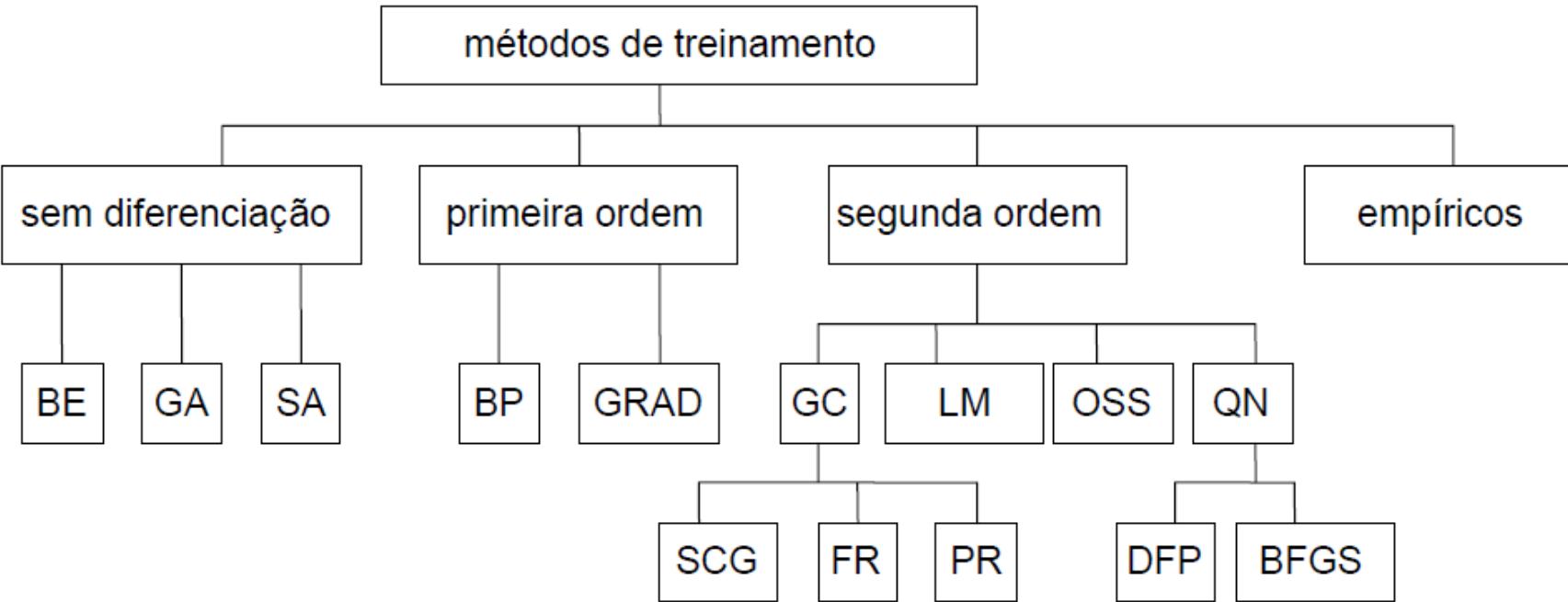
• ANNs Classificação

- @d K^D• È cæ { låÈå` D^[] |^Dæ] æ@Dç{ b D^{} [D^{} [}
{ } È@{ }
- @d K^D• È cæ { låÈå` D^[] |^Dæ] æ@D{ } c} ^d D^{} [D^{} æ• ã GåÈ@{ }
- @d K^D |æ * l{ } åÈ^{} • [l{ } , È l{* D
- @d • K^D , , È^{} • [l{ } , È l{* D^{} • ã } • D æ c^{} D^D , ' d • D læ
l @ çã D^D å^çÈ@{ |Àc^{} } • [l{ } à[ãåÈ^{} ; ã @çã ^ ãã æã }

Ù{ Áðð^|^\} &æðë[Æ* [|ð[•Á~^Á ð[Á^~^|^\{ Á^|ðæðë[Æ^|^\} æ Á
ððð^|^\} ðë[Á{ Áðð^|^\} c^•Á[} ð•Á[Á•] ð[Æðë[Á
&@ð ð[•Á ..ð å[•Á^|^\{ Áðð^|^\} &æðë[È

Ú! á ^á a Á | á ^{ | K Á æ Á • [Á a á ^{ | á a á Á | á ^{ | á a Á a Á | Á á á a Á
Ù e [Á a á ^{ | á a Á ..ç å [• Á ^{ | Á | á ^{ | á a Á | å ^{ | È

Ù^* } åǽ|å^{ K cíã æ Á{ |{ æy^• Á[à|^ Áá^|ææå Á^* } åæÈ



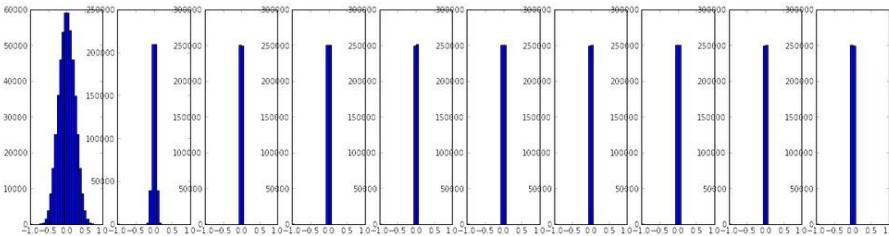
BE-Busca Exaustiva
 GA-Genetic Algorithms
 SA-Simulated Annealing

GC-Gradiente Conjugado
 BP –BackPropagation
 GRAD- Gradiente
 SCG-Scaled CG
 FR-Fletcher-Reeves
 PR-Polak-Ribière

LM- Levenberg-Marquardt
 OSS- One Step Secant
 QN-Quasi Newton
 DFP-Davidon-Fletcher-Powell
 BFGS-Broyden-Fletcher-Goldfarb-Shanno

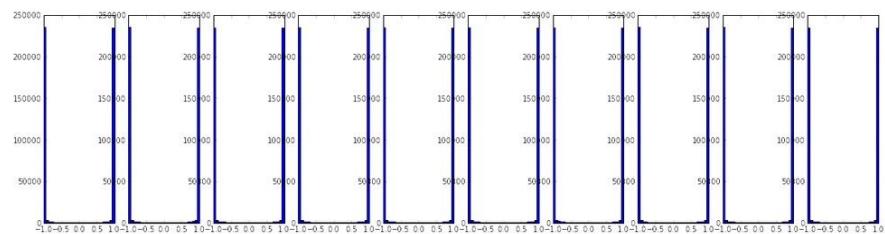


Y ^ ē @ Á Q ā ā ē ā ē }



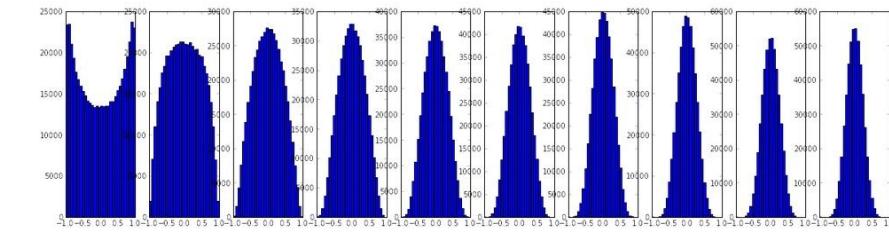
Initialization too small K

O&cáæ } • Á[Á[Á^|[É* |jää} c Á• [Á^|[É
P[Á^æ} ã *



Initialization too big K

O&cáæ } • Áæ } æ^ ÁG[Áæ @É
Ö|jää} c Á^|[É[Á^æ} ã *



Initialization just right K

þä^ Áä dä^ cä } Á Á&cáæ } • Á Á| Áæ ^|• É
Š^æ} ã * Á|[&^& å• Á &^|

Ø ä ö h á s á B Á R • c Á R @ • [} Á B Á U ^ | ^ } a Æ Y ^ } *

Š^&c | ^ Á Æ i

o t h á G É G E F I

Ú[] ^Á Á~ ã~ ã~ æ~ } Á Á~ Á~ &ç~ ^Á~ ^æ~ Á~ •~ æ~ &@

Understanding the difficulty of training deep feedforward neural networks

à^ ÁÖ|[!{ [ÓÀ, àÓ^ } * á ÉÈ€

Exact solutions to the nonlinear dynamics of learning in deep linear neural networks

Random walk initialization for training very deep feedforward networks à^ ÁJ••â[Áæ åÁ
Oââ[cÉGFI

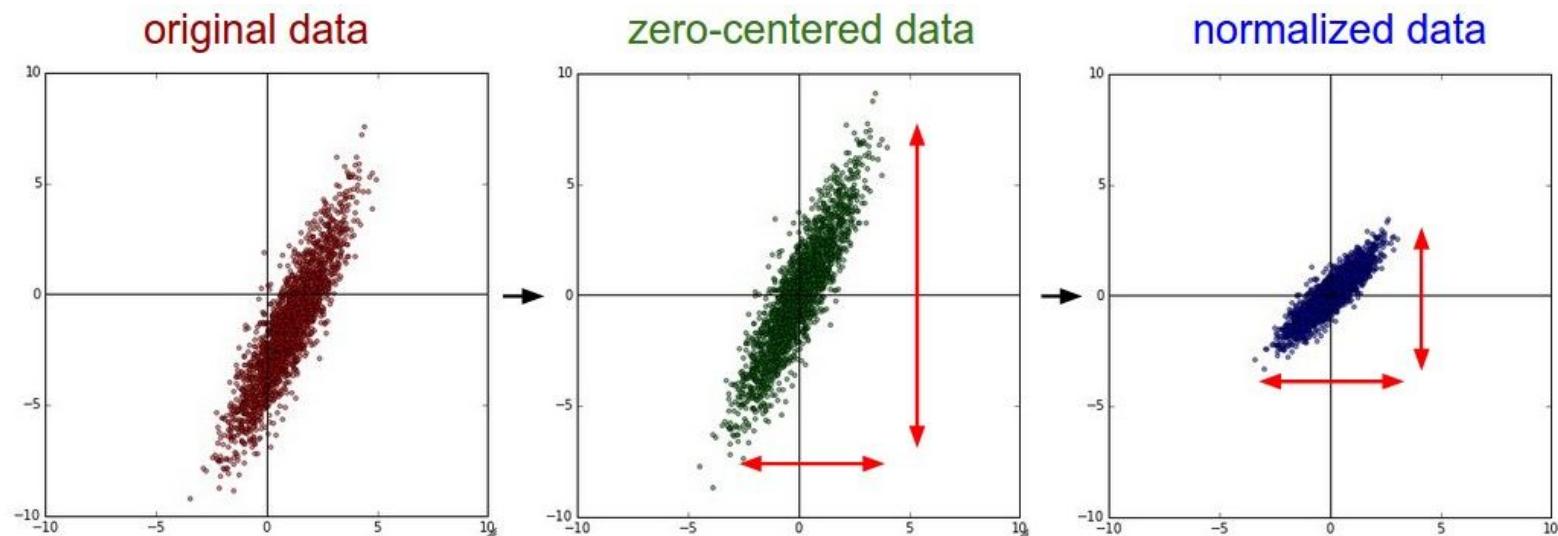
Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification

Data-dependent Initializations of Convolutional Neural Networks à ÁSlê@} à@Áo@ÁÉGEFÍ

All you need is a good init

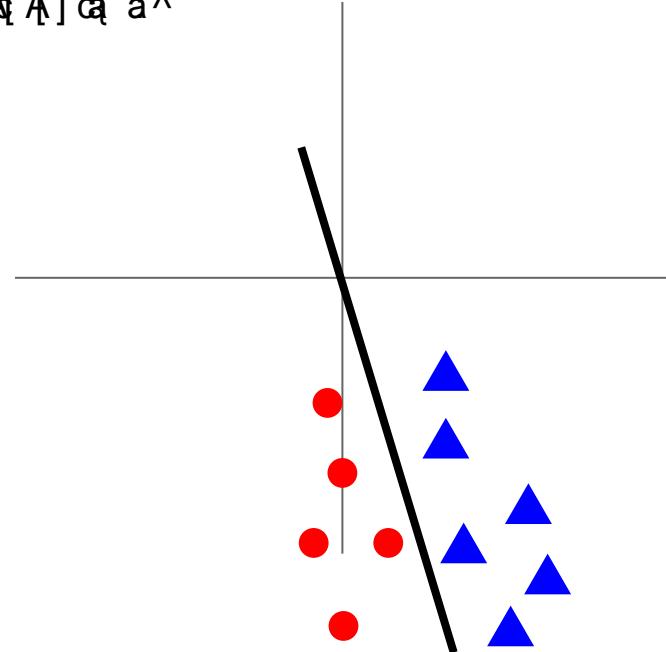
०

ÖææÁÚ!^] ![&^••ã *

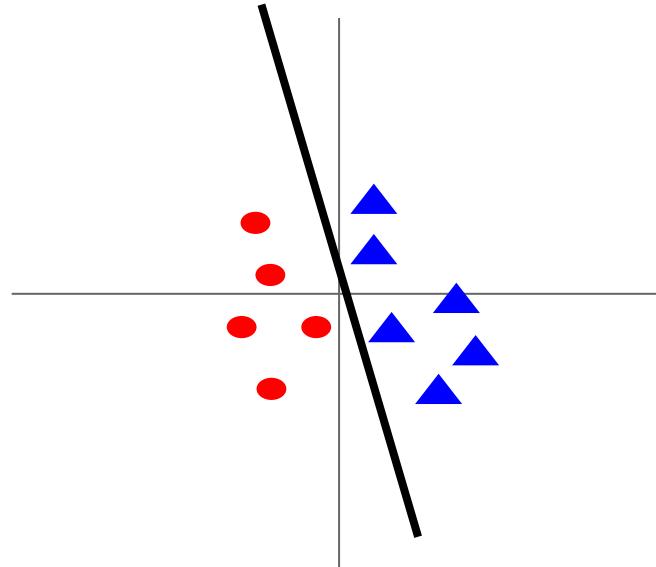


ÖææÚ!^] |[&^••ã *

Before normalization
ç^!^ Á^} • ã^ææ } Á^ • Á
&@^ * ^• Á^ Á^ @^ * ^• Á^ Á^ @^
@^éå Á^ Á^] ã^ ã^



After normalization
&@^ * ^• Á^ Á^ } • ã^ææ } Á^ • Á
@^éå Á^ Á^] ã^ ã^



Óæ&@í [:{ ǣ ã ǣ }]

Ø Æ Ø Å Æ Á Ø Æ Å! ^ b S Æ] æ @ Á R • c Á R @ • []

Š ^ & č | ^ Á F F Á F F

F i Á ^ à Á G E F i

Óæ&@P[|{ æã æã }

žq~^á åÙ: ^*^å^ ÙGEFÍ á

%d ~ Á æ Ó } ãÁ æ •• ãé Áæçæ } • ÑÁš • Ó æ ^ Á@{ Á[È

& } • ã^| Á Áæ&@ Áæçæ } • Áæ Á[{ ^ Áæ ^ | È
V[Á æ ^ Áæçæ ^ } • Á } Áæ •• ãé Èæ] | ^ K

$$\hat{x}^{(k)} = \frac{x^{(k)} - \text{E}[x^{(k)}]}{\sqrt{\text{Var}[x^{(k)}]}}$$

øæ Áæçæ ñæ Á
åã~^| ^ } ñæ | ^ Á } & } È

Øæ^æsáBØ å|^bSæ] æ@ ÁR•ç Á @•[}

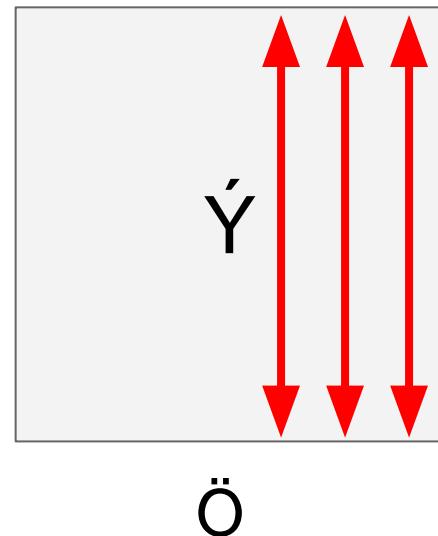
Š^&c | ^ Á Æí í

GEÆRæ ÁGEFÍ

Óæ&@P[|{ ǣ ã ǣ }

žq~^á åÙ: ^*^å^ GEFÍ á

%d ^ Á æ Á } ã Á æ •• ã Á æ } • ÑÁ
b • Á æ ^ Á @ { Á [È



FÈ&[] ^ c^ Á @ Á {] ã ã ã ã ^ æ Á ã å
ç æ æ & ^ Á å ^] ^ } å ^ } d ^ Á [! Á a & @ Á
å ã ^ } • ã } È

GEP[|{ ǣ ã ^

$$\hat{x}^{(k)} = \frac{x^{(k)} - E[x^{(k)}]}{\sqrt{\text{Var}[x^{(k)}]}}$$

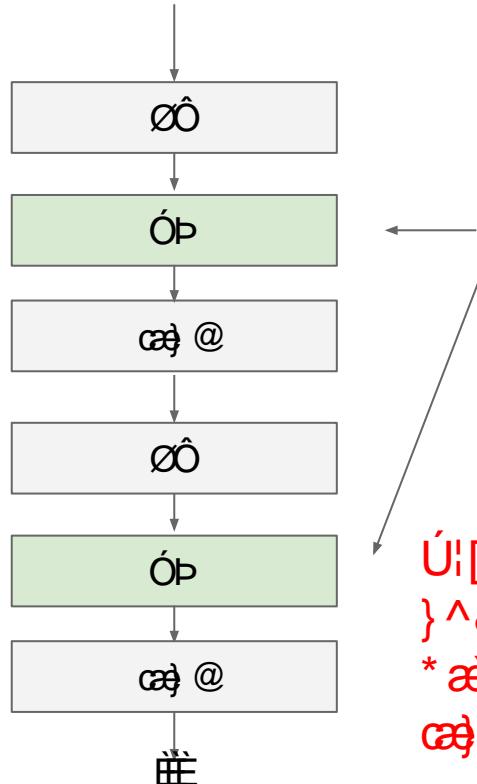
Ø æ ã ã ã ã B Ø å! ^ b S æ] æ @ Á B R • æ Á R @ • [}

Š^&c | ^ Á Æ i i

GE Ræ Á GEFÍ

Óæ&@P[|{ æã æã }]

žq~^áš åÙ: ^*^å^ ïGÉFÍ á



W•^ æ|^ Áš •^|c^åÁee^|Å||^ Á
 Ô[] } ^&c^åÁC|Å[] c[|^ c } æEæ Á
 , ^d|Á^&A[[] D|æ^&|• ïæ åÁ^ { |^ Á
 } [] |f^ ææ È

Ú!| à|^ { K| Á, ^Á
 } ^&&•• ïf^ Á ï d| Á } ãÁ
 * æ •• ïæ Á] ^ ãf| Á Á Á
 æ @æ ^|N

$$\hat{x}^{(k)} = \frac{x^{(k)} - \text{E}[x^{(k)}]}{\sqrt{\text{Var}[x^{(k)}]}}$$

Øæ^æsáBØ å|^bSæ] æ@ÁR•æ ÁR @•[]

Š^&c |^ Á Æí i

GEÆæ ÁGÉFÍ

Batch Normalization

[Ioffe and Szegedy, 2015]

Normalize:

$$\hat{x}^{(k)} = \frac{x^{(k)} - \text{E}[x^{(k)}]}{\sqrt{\text{Var}[x^{(k)}]}}$$

And then allow the network to squash the range if it wants to:

$$y^{(k)} = \gamma^{(k)} \hat{x}^{(k)} + \beta^{(k)}$$

Note, the network can learn:

$$\gamma^{(k)} = \sqrt{\text{Var}[x^{(k)}]}$$

$$\beta^{(k)} = \text{E}[x^{(k)}]$$

to recover the identity mapping.

Batch Normalization

[Ioffe and Szegedy, 2015]

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_1 \dots m\}$;

Parameters to be learned: γ, β

Output: $\{y_i = \text{BN}_{\gamma, \beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m} \sum_{i=1}^m x_i \quad // \text{mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m} \sum_{i=1}^m (x_i - \mu_{\mathcal{B}})^2 \quad // \text{mini-batch variance}$$

$$\hat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \quad // \text{normalize}$$

$$y_i \leftarrow \gamma \hat{x}_i + \beta \equiv \text{BN}_{\gamma, \beta}(x_i) \quad // \text{scale and shift}$$

- Improves gradient flow through the network
- Allows higher learning rates
- Reduces the strong dependence on initialization
- Acts as a form of regularization in a funny way, and slightly reduces the need for dropout, maybe

Batch Normalization

[Ioffe and Szegedy, 2015]

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_1 \dots m\}$;

Parameters to be learned: γ, β

Output: $\{y_i = \text{BN}_{\gamma, \beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m} \sum_{i=1}^m x_i \quad // \text{mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m} \sum_{i=1}^m (x_i - \mu_{\mathcal{B}})^2 \quad // \text{mini-batch variance}$$

$$\hat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \quad // \text{normalize}$$

$$y_i \leftarrow \gamma \hat{x}_i + \beta \equiv \text{BN}_{\gamma, \beta}(x_i) \quad // \text{scale and shift}$$

Note: at test time BatchNorm layer functions differently:

The mean/std are not computed based on the batch. Instead, a single fixed empirical mean of activations during training is used.

(e.g. can be estimated during training with running averages)

VanillaNet vs BatchNorm activations

<https://www.youtube.com/watch?v=txTv4Cjswg8>

Regularization: Add term to loss

$$L = \frac{1}{N} \sum_{i=1}^N \sum_{j \neq y_i} \max(0, f(x_i; W)_j - f(x_i; W)_{y_i} + 1) + \boxed{\lambda R(W)}$$

In common use:

L2 regularization

$$R(W) = \sum_k \sum_l W_{k,l}^2 \quad (\text{Weight decay})$$

L1 regularization

$$R(W) = \sum_k \sum_l |W_{k,l}|$$

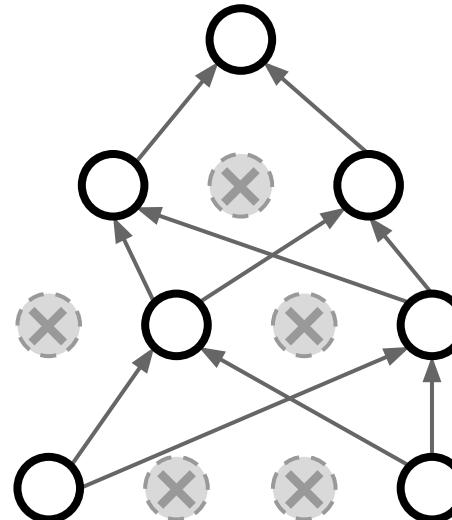
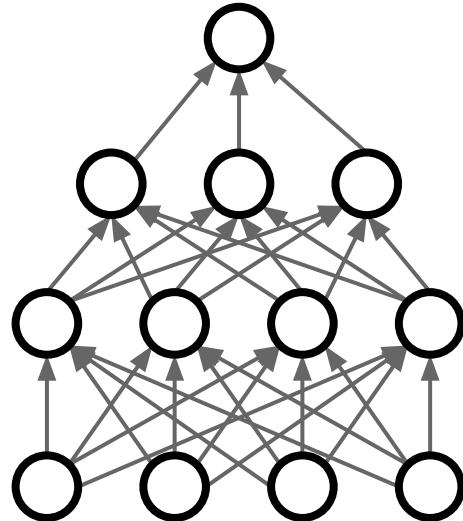
Elastic net (L1 + L2)

$$R(W) = \sum_k \sum_l \beta W_{k,l}^2 + |W_{k,l}|$$

Regularization: Dropout

In each forward pass, randomly set some neurons to zero

Probability of dropping is a hyperparameter; 0.5 is common



Srivastava et al, "Dropout: A simple way to prevent neural networks from overfitting", JMLR 2014

Regularization: Dropout

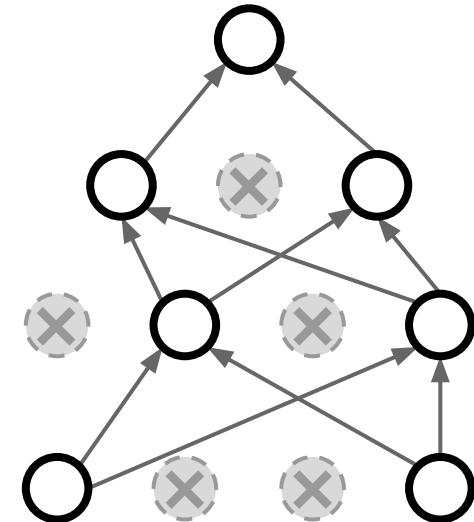
```
p = 0.5 # probability of keeping a unit active. higher = less dropout

def train_step(X):
    """ X contains the data """

    # forward pass for example 3-layer neural network
    H1 = np.maximum(0, np.dot(W1, X) + b1)
    U1 = np.random.rand(*H1.shape) < p # first dropout mask
    H1 *= U1 # drop!
    H2 = np.maximum(0, np.dot(W2, H1) + b2)
    U2 = np.random.rand(*H2.shape) < p # second dropout mask
    H2 *= U2 # drop!
    out = np.dot(W3, H2) + b3

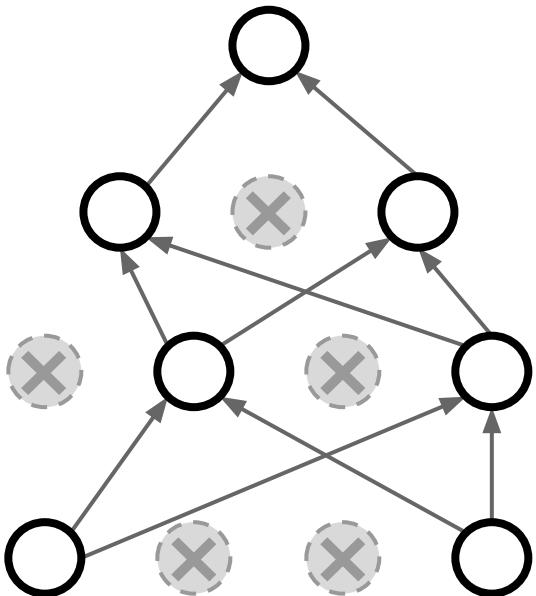
    # backward pass: compute gradients... (not shown)
    # perform parameter update... (not shown)
```

Example forward pass with a 3-layer network using dropout



Regularization: Dropout

How can this possibly be a good idea?

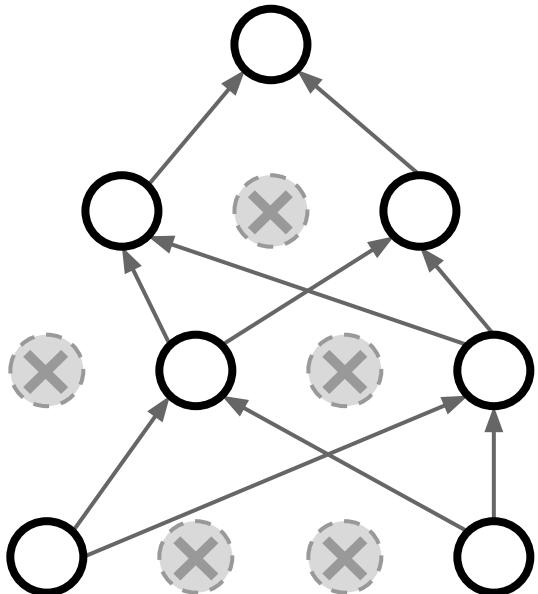


Forces the network to have a redundant representation;
Prevents co-adaptation of features



Regularization: Dropout

How can this possibly be a good idea?



Another interpretation:

Dropout is training a large **ensemble** of models (that share parameters).

Each binary mask is one model

An FC layer with 4096 units has $2^{4096} \sim 10^{1233}$ possible masks!

Only $\sim 10^{82}$ atoms in the universe...

Dropout: Test time

Dropout makes our output random!

$$\boxed{y} = f_W(\boxed{x}, \boxed{z})$$

Output (label) Input (image) Random mask

Want to “average out” the randomness at test-time

$$y = f(x) = E_z[f(x, z)] = \int p(z)f(x, z)dz$$

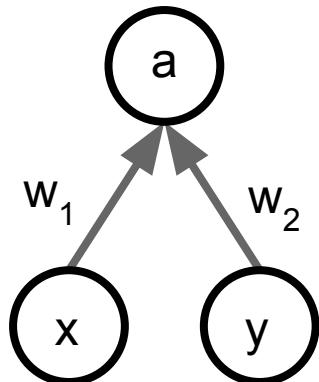
But this integral seems hard ...

Dropout: Test time

Want to approximate
the integral

$$y = f(x) = E_z[f(x, z)] = \int p(z)f(x, z)dz$$

Consider a single neuron.

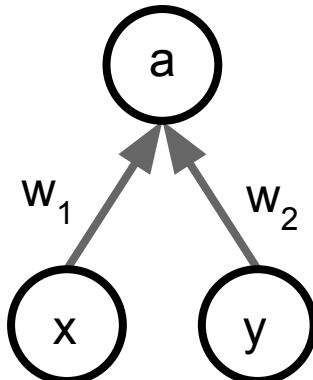


Dropout: Test time

Want to approximate
the integral

$$y = f(x) = E_z[f(x, z)] = \int p(z)f(x, z)dz$$

Consider a single neuron.



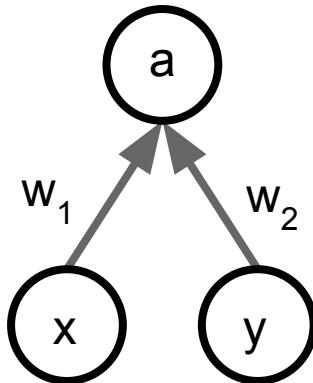
At test time we have: $E[a] = w_1x + w_2y$

Dropout: Test time

Want to approximate
the integral

$$y = f(x) = E_z[f(x, z)] = \int p(z)f(x, z)dz$$

Consider a single neuron.



At test time we have: $E[a] = w_1x + w_2y$

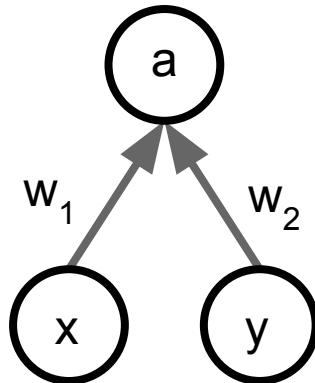
During training we have:

$$\begin{aligned} E[a] &= \frac{1}{4}(w_1x + w_2y) + \frac{1}{4}(w_1x + 0y) \\ &\quad + \frac{1}{4}(0x + 0y) + \frac{1}{4}(0x + w_2y) \\ &= \frac{1}{2}(w_1x + w_2y) \end{aligned}$$

Dropout: Test time

Want to approximate
the integral

$$y = f(x) = E_z[f(x, z)] = \int p(z)f(x, z)dz$$



Consider a single neuron.

At test time we have: $E[a] = w_1x + w_2y$

During training we have:

$$\begin{aligned} E[a] &= \frac{1}{4}(w_1x + w_2y) + \frac{1}{4}(w_1x + 0y) \\ &\quad + \frac{1}{4}(0x + 0y) + \frac{1}{4}(0x + w_2y) \\ &= \frac{1}{2}(w_1x + w_2y) \end{aligned}$$

At test time, multiply
by dropout probability

Dropout: Test time

```
def predict(X):
    # ensembled forward pass
    H1 = np.maximum(0, np.dot(W1, X) + b1) * p # NOTE: scale the activations
    H2 = np.maximum(0, np.dot(W2, H1) + b2) * p # NOTE: scale the activations
    out = np.dot(W3, H2) + b3
```

At test time all neurons are active always

=> We must scale the activations so that for each neuron:

output at test time = expected output at training time

Dropout Summary

```
""" Vanilla Dropout: Not recommended implementation (see notes below) """

p = 0.5 # probability of keeping a unit active. higher = less dropout

def train_step(X):
    """ X contains the data """

    # forward pass for example 3-layer neural network
    H1 = np.maximum(0, np.dot(W1, X) + b1)
    U1 = np.random.rand(*H1.shape) < p # first dropout mask
    H1 *= U1 # drop!
    H2 = np.maximum(0, np.dot(W2, H1) + b2)
    U2 = np.random.rand(*H2.shape) < p # second dropout mask
    H2 *= U2 # drop!
    out = np.dot(W3, H2) + b3

    # backward pass: compute gradients... (not shown)
    # perform parameter update... (not shown)

def predict(X):
    # ensembled forward pass
    H1 = np.maximum(0, np.dot(W1, X) + b1) * p # NOTE: scale the activations
    H2 = np.maximum(0, np.dot(W2, H1) + b2) * p # NOTE: scale the activations
    out = np.dot(W3, H2) + b3
```

drop in forward pass

scale at test time

More common: “Inverted dropout”

```
p = 0.5 # probability of keeping a unit active. higher = less dropout

def train_step(X):
    # forward pass for example 3-layer neural network
    H1 = np.maximum(0, np.dot(W1, X) + b1)
    U1 = (np.random.rand(*H1.shape) < p) / p # first dropout mask. Notice /p!
    H1 *= U1 # drop!
    H2 = np.maximum(0, np.dot(W2, H1) + b2)
    U2 = (np.random.rand(*H2.shape) < p) / p # second dropout mask. Notice /p!
    H2 *= U2 # drop!
    out = np.dot(W3, H2) + b3

    # backward pass: compute gradients... (not shown)
    # perform parameter update... (not shown)

def predict(X):
    # ensembled forward pass
    H1 = np.maximum(0, np.dot(W1, X) + b1) # no scaling necessary
    H2 = np.maximum(0, np.dot(W2, H1) + b2)
    out = np.dot(W3, H2) + b3
```

test time is unchanged!



Aplicação prática

Problema Abordado

- Os problemas poderão ser de dois tipos:
 - classificação ou
 - aproximação.
- 1) Problemas de Classificação: Dado um padrão (exemplo), a rede deve dar como resposta a classe à qual ele pertence.

Idade	Renda	...	Profissão	Classe	
24	1070	...	Engenheiro	Bom pagador	← Padrão 1
...
41	4700	...	Professor	Mau pagador	← Padrão N

Atributos numéricos (ou quantitativos)

Atributo categórico (ou qualitativo)

Problema Abordado

- 2) Problemas de **Aproximação**: Dado um padrão, a rede deve gerar saídas que se aproximem das saídas verdadeiras.

Temperatura	Umidade	...	Dir. dos Ventos	Quant. chuva
27	0.28	...	Norte	0.12
...
21	0.67	...	Sudeste	1.32

- Em ambos os casos, deseja-se **generalização**, ou seja, que a rede seja capaz de gerar as saídas mais corretas possíveis não apenas para os **padrões apresentados no treinamento**, mas também para **padrões novos**.

Pré-processamento

- 1) Para os atributos **numéricos**: Fazer **normalização** (escalonamento):
 - Valores devem ser normalizados para o **intervalo [0,1]** (**depende da função de ativação a ser utilizada – sigmóide logística**).
 - Expressão de normalização:

$$x_{norm} = \frac{(x - x_{min})}{(x_{max} - x_{min})}$$

- onde x_{norm} é o **valor normalizado** correspondente ao valor **original** x , e x_{min} e x_{max} são os valores **mínimo** e **máximo** entre todos os valores da base de dados.
- Pode ser feito separadamente por atributo.
- Este processo não é necessariamente o melhor (**o objetivo é apenas didático**).

Pré-processamento

- 2) Para os atributos **categóricos** e saídas: Fazer **codificação ortogonal**.
 - Atribui-se uma **seqüência de bits** ao atributo, sendo que apenas um dos bits vale 1, indicando a categoria.
 - Ex.: Se um atributo tiver como valores possíveis **A**, **B** e **C**, então:
 - a categoria **A** fica codificada como **1 0 0**,
 - a categoria **B** fica codificada como **0 1 0**,
 - a categoria **C** fica codificada como **0 0 1**.

Pré-processamento



- 3) No caso de **missing data** (valores faltando):
 - a) **Eliminação de padrões**
 - Quando há padrões suficientes, eliminamos aqueles com valores faltando.
 - Obs.: É difícil definir quantos padrões são “suficientes” (neste projeto, isto não é cobrado rigorosamente).
 - b) **Substituição de valores**
 - i) Atributos **numéricos**: o valor faltando pode ser **substituído pela média** dos valores nos demais padrões.
 - ii) Atributos **categóricos**:
 - » pode ser criada uma **nova categoria** (“ausente”),
 - » ou o valor faltando pode ser **substituído por alguma categoria existente** (observando a semântica).

Particionamento dos Dados

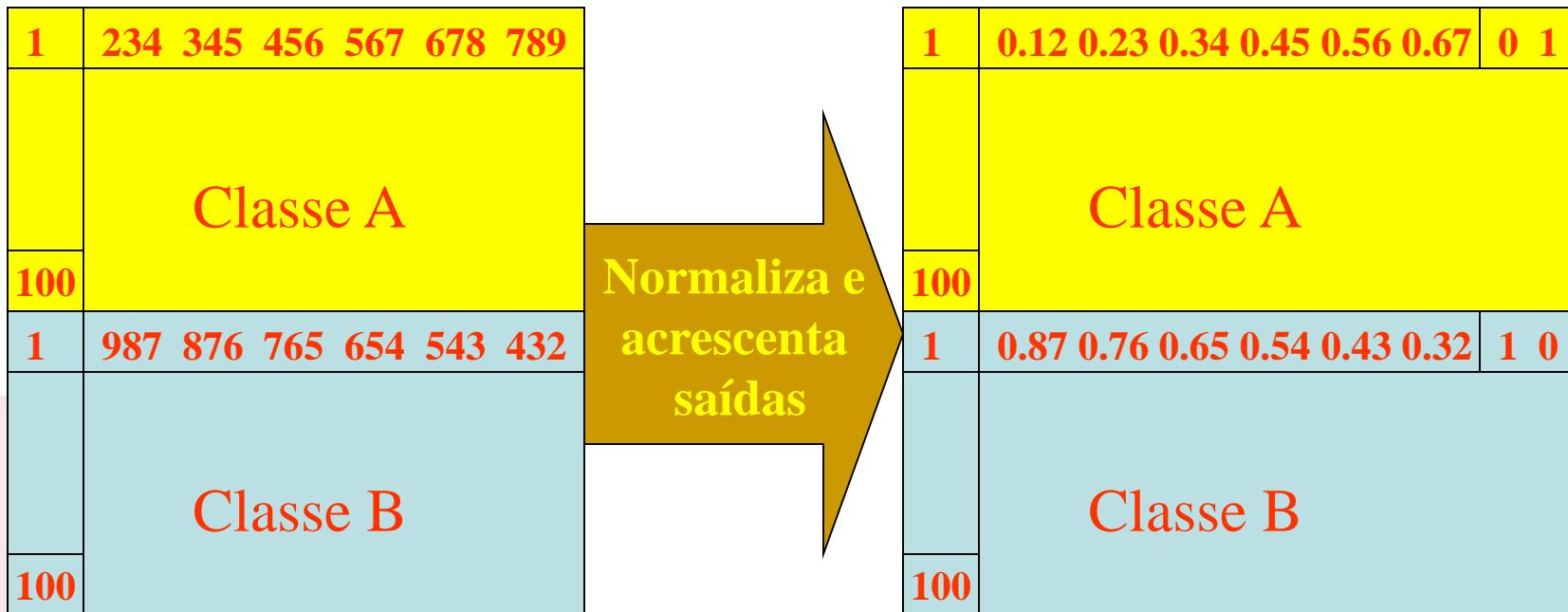
- O particionamento adotado é o sugerido pelo conhecido relatório *Proben1*:
 - **50%** dos padrões de cada classe escolhidos aleatoriamente para **treinamento**,
 - **25%** para **validação**,
 - **25%** para **teste**.
- É importante que as **proporções entre as classes** no conjunto completo sejam mantidas nos conjuntos de treinamento, validação e teste.

Particionamento dos Dados

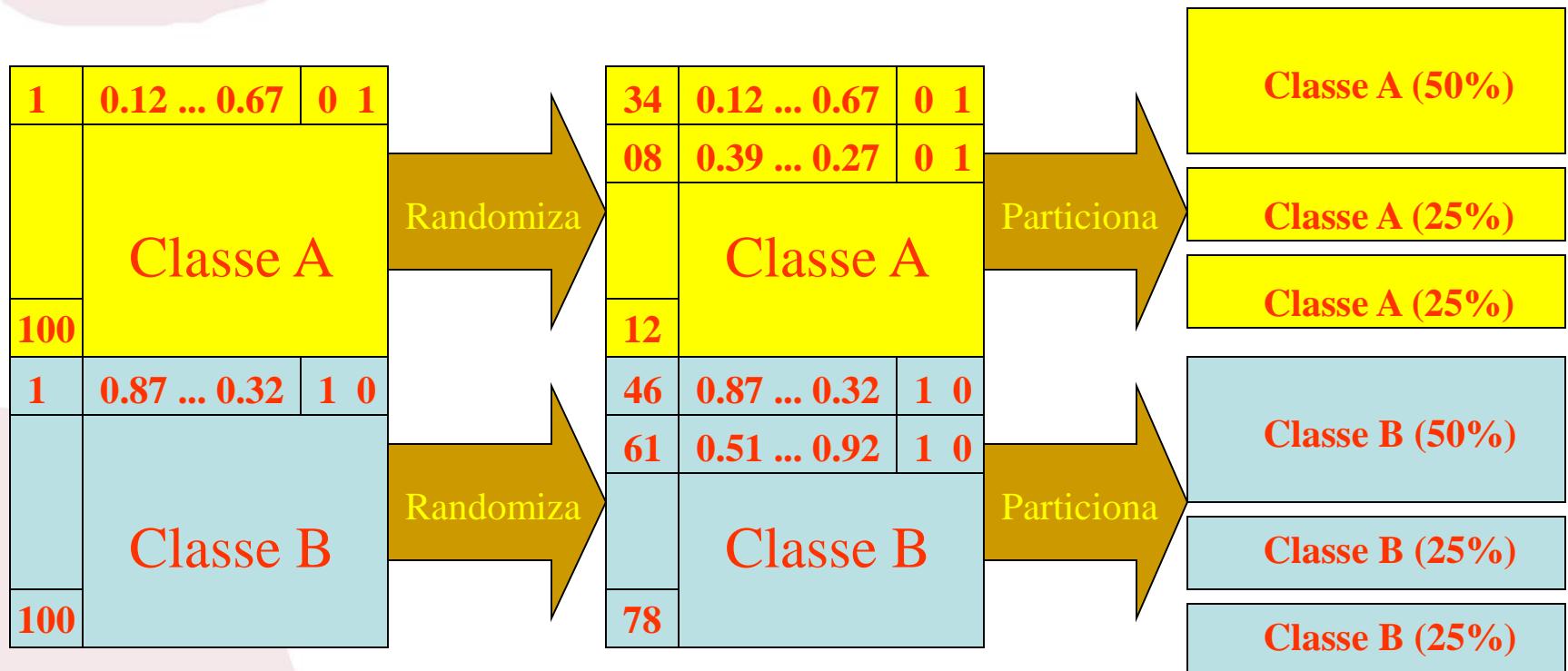
- Se houver **quantidades diferentes de padrões** nas classes, a equipe pode escolher entre:
 - a) **Usar a mesma quantidade de padrões para todas as classes.**
 - Ex.: Classe A com 200 padrões e classe B com 100 padrões.
 - » Usa todos da classe B e escolhe aleatoriamente 100 padrões da classe A.
 - Obs.: Pode ser que a base fique com poucos padrões (é difícil definir a quantidade certa; não será cobrado com rigor).
 - b) **Usar todos os dados, mantendo a proporção existente.**
 - Pode ser que as classes com mais padrões sejam “muito mais aprendidas” do que as outras.

Particionamento dos Dados

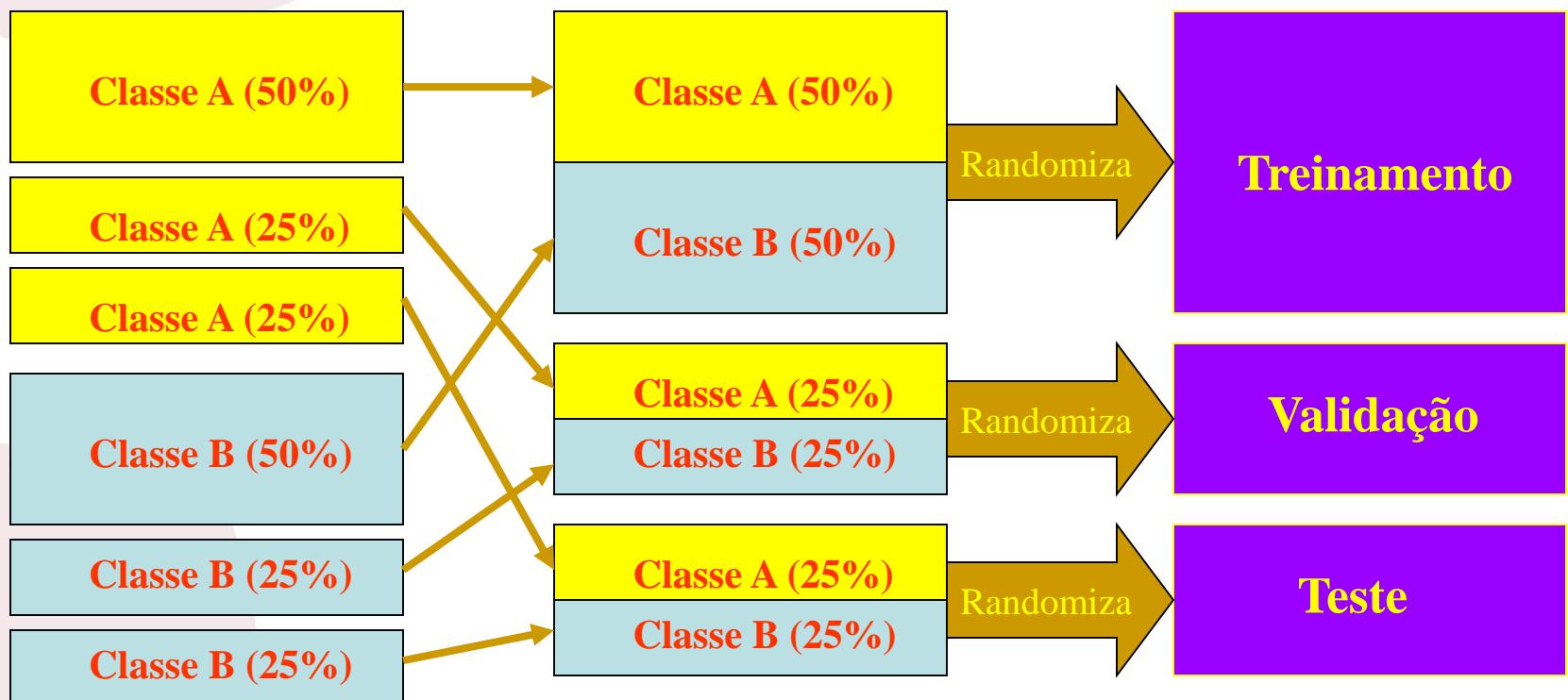
Exemplo:



Particionamento dos Dados



Particionamento dos Dados

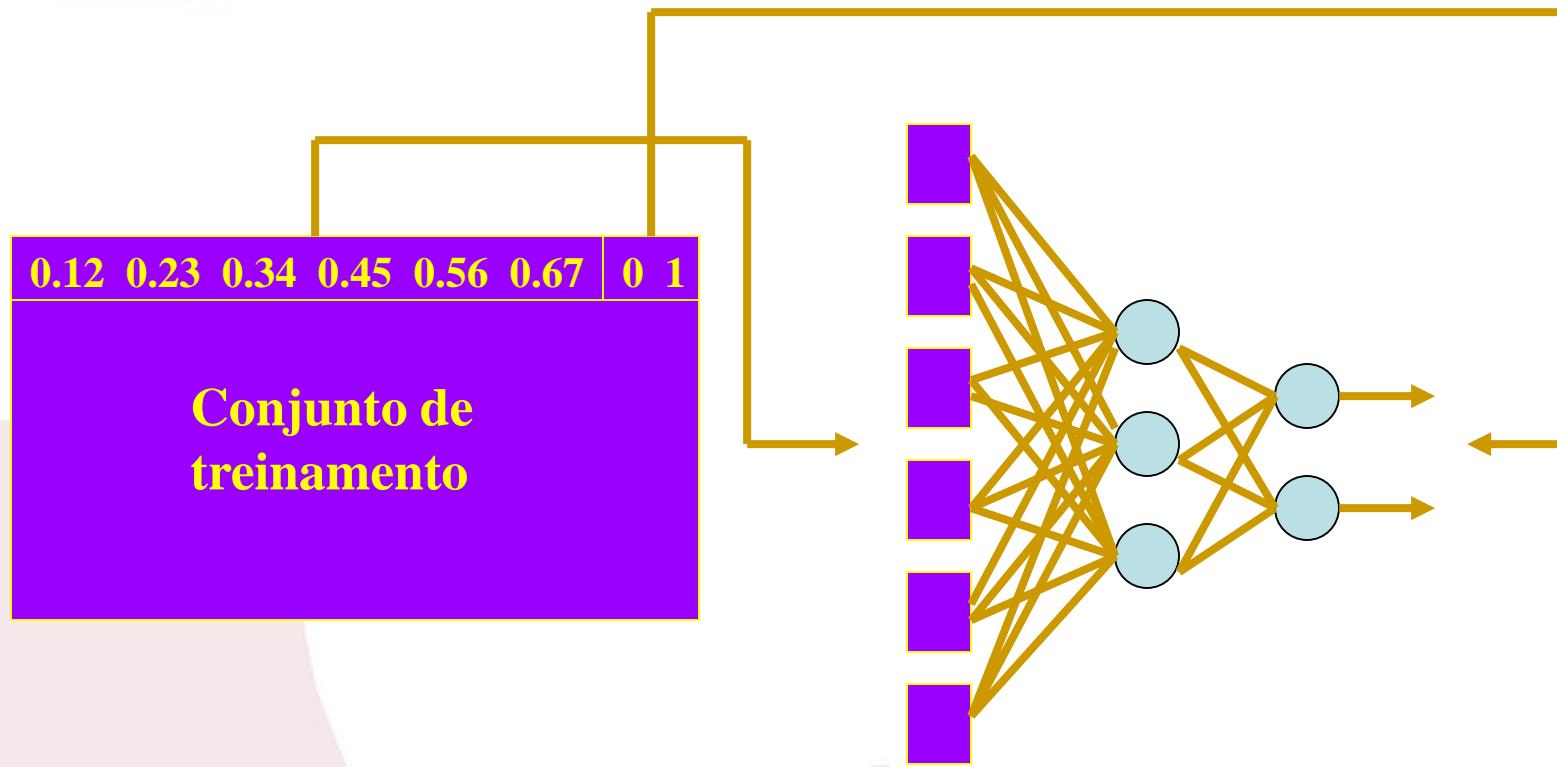


Definição da Topologia MLP

- Aspectos de treinamento:
 - N^º de nodos de **entrada**: Quantidade de atributos de entrada.
 - N^º de nodos de **saída**:
 - Em problemas de **classificação**, é a quantidade de classes.
 - » Regra de classificação **winner-takes-all**: o nodo de saída que gerar a maior saída define a classe do padrão.
 - Em problemas de **aproximação**, é a quantidade de variáveis de saída.
 - **Uma única camada escondida**.
 - **Função de ativação** dos neurônios: sigmóide logística.
 - Todas as possíveis conexões entre camadas adjacentes, sem conexões entre camadas não-adjacentes.

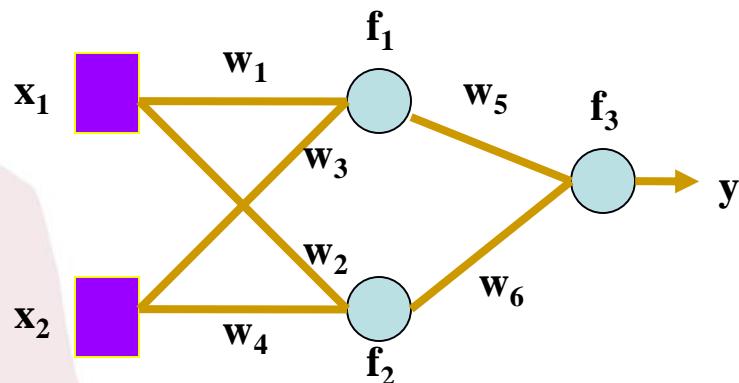
Definição da Topologia MLP

Exemplo: 6 entradas e 2 saídas.



Definição da Topologia MLP

- Aspectos normalmente **variáveis**:
 - **Nº de neurônios escondidos** (projetista).
- Variando o nº de neurônios escondidos, estamos variando a quantidade de pesos da rede.
- Explicação: Uma rede neural **implementa uma função**.



As funções f_i são do tipo sigmóide logística.

$$y = f_3(w_5 f_1 (w_1 x_1 + w_3 x_2) + w_6 f_2 (w_2 x_1 + w_4 x_2)).$$

Definição da Topologia MLP

- Os pesos da rede são os **parâmetros da função**.
- Dessa forma, aumentar a quantidade de pesos da rede significa **aumentar a complexidade da função** implementada.
 - Se a quantidade de pesos for **pequena demais**, pode haver **underfitting**.
 - A função implementada **não tem complexidade suficiente** para resolver o problema abordado.
 - Se a quantidade de pesos for **grande demais**, pode haver **overfitting**.
 - A função implementada tem **complexidade demais para o problema**, sendo capaz de **modelar detalhes** demais dos dados de treinamento.
 - Dessa forma, a rede **não generaliza** bem.

Medidas de Erro

- Para ambos os tipos de problema, será usado o **erro SSE** (*sum squared error* - soma dos erros quadráticos).
- Ex.:

	Saídas da rede			Saídas desejadas		
Padrão	1	...	N	1	...	N
Nodo 1	0.98	...	0.12	1.00	...	0.00
Nodo 2	0.02	...	0.96	0.00	...	1.00

- Soma dos erros quadráticos (SSE):

$$\text{SSE} = (0.98 - 1.00)^2 + \dots + (0.12 - 0.00)^2 + \\ (0.02 - 0.00)^2 + \dots + (0.96 - 1.00)^2.$$

$$SSE = \sum_{i=1}^p \|t^{(i)} - y^{(i)}\|^2$$

Medidas de Erro

- Para problemas de **classificação**, também será calculado o **erro de classificação** (neste projeto, só para o conjunto de teste).
- Regra de classificação **winner-takes-all**:
 - O neurônio de saída que apresentar o maior valor de saída determina a classe do padrão.
- Ex.:

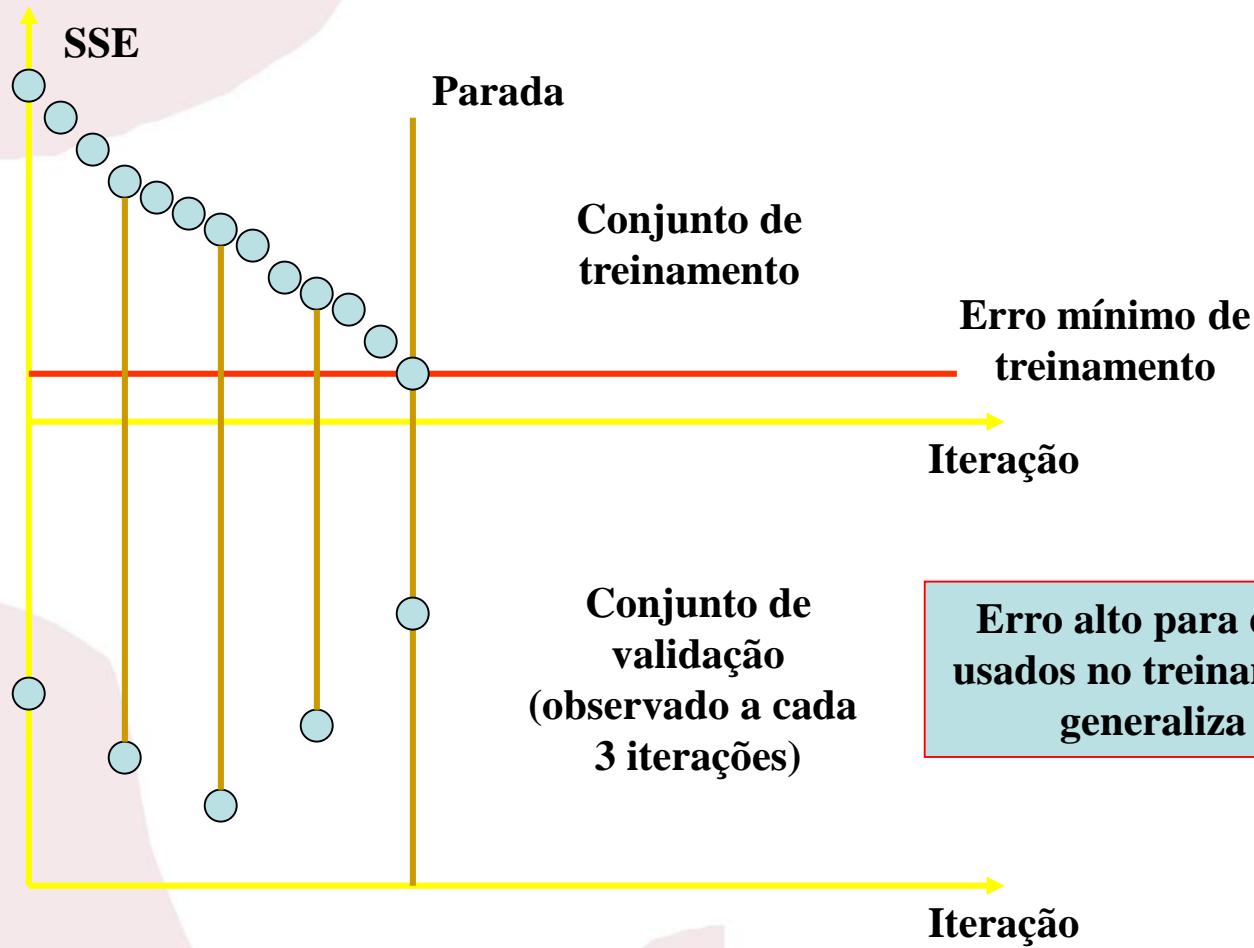
	Saídas da rede			Saídas desejadas		
Padrão	1	...	N	1	...	N
Nodo 1	0.98	...	0.12	1.00	...	0.00
Nodo 2	0.02	...	0.96	0.00	...	1.00
Classe	1	...	2	1	...	2

$$\text{Erro Classif.} = 100 \times \frac{\text{Quant. de padrões classificados erradamente}}{\text{Quant. total de padrões}}$$

Treinamento com Backpropagation

- Será usado o algoritmo **backpropagation** padrão.
 - É um algoritmo de **gradiente descendente**, ou seja, utiliza informações de **derivada**.
 - Por isso, as funções de ativação devem ser **contínuas e diferenciáveis** (é o caso da sigmóide logística).
- **Objetivo:** Fazer “ajuste de pesos”, ou seja, **escolher os pesos que geram as saídas mais corretas possíveis** (menor erro) de forma iterativa.
- **Idéia geral:** A cada iteração, obter um **erro cada vez menor** para os dados de treinamento.
- **Cuidado:** Não permitir que a rede aprenda **detalhes demais** do conjunto de treinamento (**overfitting**).

Parada por Erro Mínimo de Treinamento



Parada por Erro Mínimo de Validação



- É recomendável que o treinamento seja interrompido quando o **erro no conjunto de validação atingir um mínimo.**
 - A partir deste ponto, supõe-se que a rede só aprenderia **detalhes irrelevantes** do conjunto de treinamento.
 - O erro para dados de treinamento seria cada vez menor, mas o **erro para dados novos** (validação) seria cada vez mais alto.
- Neste projeto, será usado o seguinte **critério de parada**:
 - Interromper o treinamento quando o **erro de validação subir por 5 iterações consecutivas**.
 - É o critério implementado no Matlab (parâmetro “max_fail = 5”).

Parâmetros de Treinamento

- Os parâmetros **variáveis** do treinamento serão:
 - **Taxa de aprendizado**,
 - **Máximo de iterações permitidas** (é outro critério de parada).
- A equipe deve escolher **3 valores** para cada um deles.
- Usando **taxa de aprendizado muito baixa**, cada iteração faz um **ajuste muito pequeno** nos pesos (passo muito pequeno).
 - Pode precisar de **muitas iterações** para convergir para o ponto de mínimo desejado na superfície de busca.
- Usando **taxa de aprendizado muito alta**, cada iteração faz um **ajuste muito grande** nos pesos (passo muito grande).
 - Pode causar **oscilações** em torno de um ponto de mínimo.

Análise de Resultados

- Serão usadas:
 - 3 quantidades de neurônios escondidos,
 - 3 taxas de aprendizado,
 - 3 quantidades máximas de iterações permitidas.
- Temos um total de 27 configurações a serem testadas.
- Para cada configuração, será realizado um treinamento.
- A melhor configuração a ser escolhida é a de

Config.	SSE de Treinamento	SSE de Validação
1	2.13	3.45
2	1.44	0.71
...	...	
27	4.43	5.18

Melhor configuração



Análise de Resultados

- Para a **melhor configuração** escolhida, devem ser feitos **30 treinamentos com diferentes inicializações de pesos**.
- O objetivo é verificar como a melhor rede se comporta quando variamos os pesos iniciais.

Inicialização	SSE de Treinamento	SSE de Validação	SSE de Teste	E.Class. de Teste
1	1.12	0.66	0.79	12.08
2	1.44	0.71	0.88	13.32
...
30	1.23	0.66	0.90	09.87
Média	1.15	0.70	0.85	11.24
Desv-pad	0.07	0.11	0.10	02.35

Análise de Resultados

- Deve ser escrito um **relatório** com a análise dos resultados.
- Os **gráficos** a serem incluídos ficam a critério da equipe.
- **Dicas:**

- Matriz de confusão

		Classe verdadeira		
		1	2	3
Classe prevista	1	40	5	0
	2	0	45	0
3	10	1	39	

- Matriz de custo

		Classe verdadeira		
		1	2	3
Classe prevista	1	0	2	5
	2	1	0	2
3	3	1	0	

- Gráficos de desempenho

Medidas de erro

$$MAE \text{ (mean absolute error)} = \frac{1}{N} \sum_{i=1}^N |T_i^{predicted} - T_i^{true}|$$

$$MSE \text{ (mean squared error)} = \frac{1}{N} \sum_{i=1}^N (T_i^{predicted} - T_i^{true})^2$$

$$RMSE \text{ (root mean squared error)} = \sqrt{\frac{1}{N} \sum_{i=1}^N (T_i^{predicted} - T_i^{true})^2}$$

$$MARE \text{ (mean absolute relative error)} = \frac{1}{N} \sum_{i=1}^N \left| \frac{T_i^{predicted} - T_i^{true}}{T_i^{true}} \right|$$

$$MSRE \text{ (mean squared relative error)} = \frac{1}{N} \sum_{i=1}^N \left(\frac{T_i^{predicted} - T_i^{true}}{T_i^{true}} \right)^2$$

$$RMSRE \text{ (root mean squared relative error)} = \sqrt{\frac{1}{N} \sum_{i=1}^N \left(\frac{T_i^{predicted} - T_i^{true}}{T_i^{true}} \right)^2}$$

$$MAPE \text{ (mean absolute percentage error)} = \frac{1}{N} \sum_{i=1}^N \left| \frac{T_i^{predicted} - T_i^{true}}{T_i^{true}} \right| \times 100$$

$$MSPE \text{ (mean squared percentage error)} = \frac{1}{N} \sum_{i=1}^N \left(\frac{T_i^{predicted} - T_i^{true}}{T_i^{true}} \right)^2 \times 100$$

$$RMSPE \text{ (root mean squared percentage error)} = \sqrt{\frac{1}{N} \sum_{i=1}^N \left(\frac{T_i^{predicted} - T_i^{true}}{T_i^{true}} \right)^2} \times 100$$

Aspectos do treinamento de redes MLP

O aprendizado é resultado de **apresentação repetitiva** de todas as amostras do **conjunto de treinamento**.

Cada apresentação de todo o conjunto de treinamento é denominada **época ou iteração**.

O processo de aprendizagem é **repetido** época após época, até que um **critério de parada** seja satisfeito.

É recomendável que a **ordem de apresentação** das amostras seja **aleatória** de uma época para outra. Isso tende a fazer com que o **ajuste de pesos** tenha um caráter **estocástico** ao longo do treinamento.

Atualização local ou por lote.



Atualização pode ser de duas maneiras básicas: **local e por lote**.

Local: a atualização é feita imediatamente após a apresentação de cada amostra de treinamento.

- é também chamado de método de atualização **on-line** ou padrão a padrão.
- requer um menor armazenamento para cada conexão, e apresenta **menos possibilidade de convergência para um mínimo local**.

Lote: a atualização dos pesos só é feita após a apresentação de todas as amostras de treinamento que constituem uma época.

- é também conhecido como método de atualização **off-line** ou **batch**.
- o **ajuste** relativo a cada apresentação de uma amostra é **acumulado**.
- fornece uma melhor estimativa do vetor gradiente.

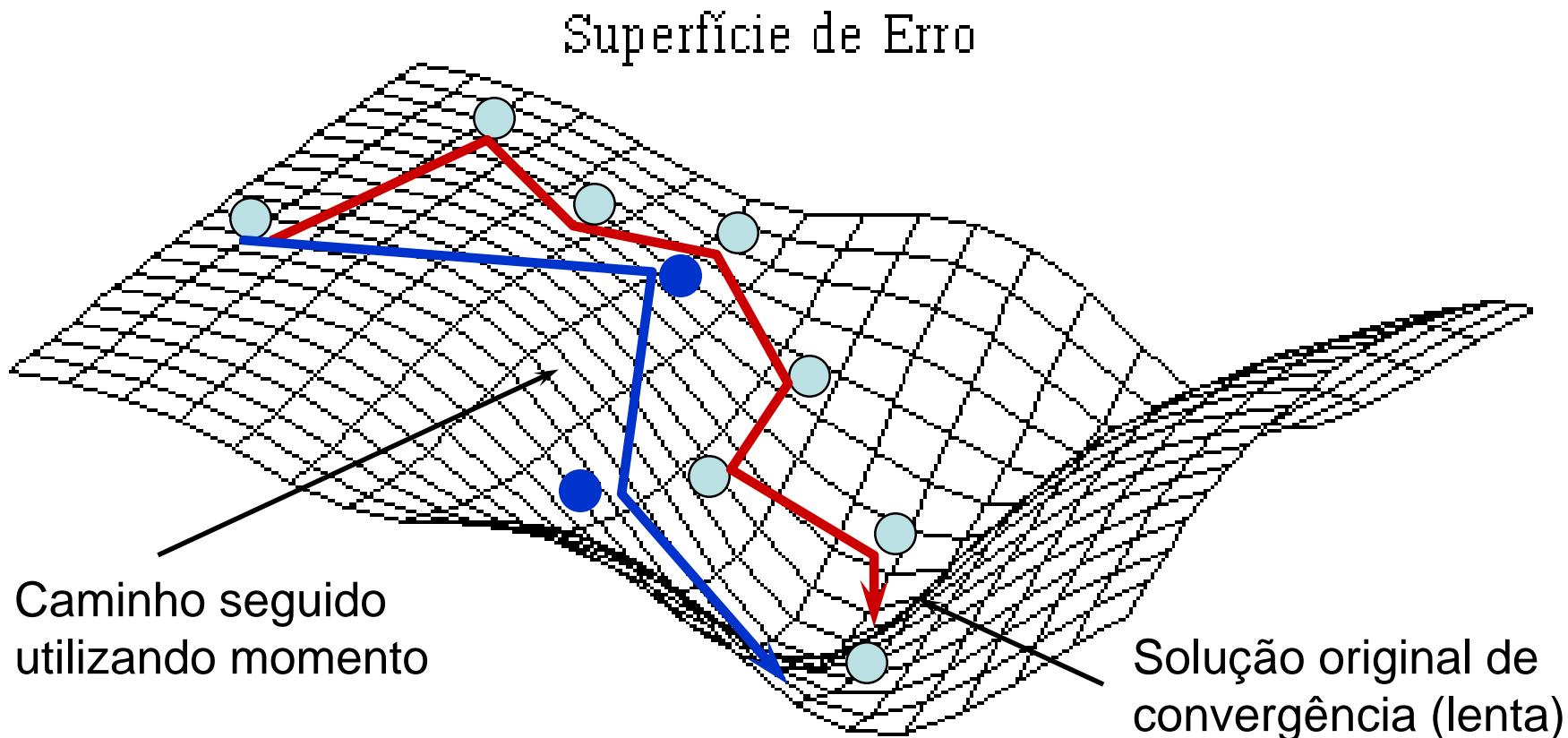
Atualização dos pesos

■ Momentum

$$\Delta w_{ij}(t + 1) = \eta x_i y_j (1 - y_j) \delta_j + \boxed{\alpha (w_{ij}(t) - w_{ij}(t - 1))}$$

- Aumenta velocidade de aprendizado evitando perigo de instabilidade
- Pode acelerar treinamento em regiões muito planas da superfície de erro
- Suprime oscilação de pesos em vales e ravinas
- Normalmente, α é ajustada entre 0,5 e 0,9

Momento



Critérios de parada

O processo de minimização do MSE (função custo) não apresenta convergência garantida e não possui um critério de parada bem definido.

Um critério de parada não muito recomendável, que não leva em conta o estado do processo iterativo é o da **pré-definição do número total de iterações**.

Apresenta-se a seguir um critério de parada que leva em conta o processo iterativo.

Critérios de parada (cont.)

Consideremos um critério que leva em conta informações a respeito do estado iterativo. Considera-se nesse caso a **possibilidade de existência de mínimos locais**.

Seja θ^* o **vetor de pesos** que denota um ponto **mínimo**, local ou global.

Uma **condição** para que θ^* seja um **mínimo** é que o gradiente $\nabla \mathfrak{J}(\theta)$, da função custo, seja zero em $\theta = \theta^*$.

Como critério tem-se as **seguintes alternativas** de parada:

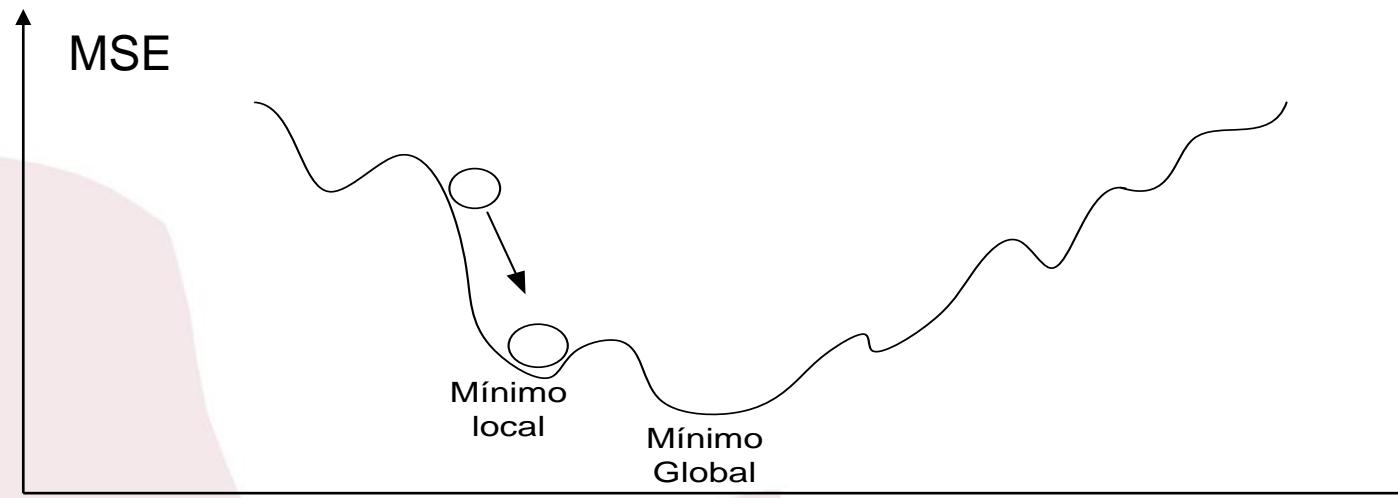
1 - quando a norma euclidiana da estimativa do **vetor gradiente** $\|\nabla \mathfrak{J}(\theta)\|$ atinge um valor suficientemente **pequeno**.

2 - quando a **variação** do **erro quadrático médio** (MSE) de uma época para outra atingir um valor suficientemente **pequeno**.

3 - quando o erro **quadrático médio** atingir um valor suficientemente **pequeno** ou seja, $\mathfrak{J}_{med}(\theta) \leq \varepsilon$ onde ε é um valor suficientemente pequeno.

Critérios de parada (cont.)

- Nota-se que se o critério é de valor **mínimo de MSE** então não se garante que o algoritmo irá atingir esse valor.
- Por outro lado, se o critério é o mínimo valor do **vetor gradiente** deve-se considerar que o algoritmo **termina no mínimo local** mais próximo.



Critérios de parada (cont.)



Outro critério de parada que pode ser usado em conjunto com um dos critérios anteriores é a **avaliação da capacidade de generalização** da rede após cada época de treinamento.

O processo de treinamento é interrompido antes que a capacidade de generalização da rede fique restrita.

Arquitetura da rede

A quantidade de neurônios na **camada de entrada** é dada pelo **problema** a ser abordado.

No entanto, a quantidade de neurônios **nas camadas de processamento** são características do **projeto**.

Aumentando-se o número de neurônios na **camada escondida** aumenta-se a capacidade de mapeamento não-linear da rede.

No entanto, quando esse número for **muito grande**, o modelo pode se sobre-ajustar aos dados, na presença de ruído nas amostras de treinamento. Diz-se que a rede está sujeita ao sobre-treinamento (**overfitting**).

Arquitetura da rede



Por outro lado, uma rede com **poucos neurônios** na camada escondida pode não ser capaz de realizar o mapeamento desejado, o que é denominado de ***underfitting***.

O *underfitting* também pode ser causado quando o **treinamento é interropido** de forma prematura.

Unidades intermediárias

- Número de neurônios nas camadas intermediárias (cont.)
 - Depende de:
 - Número de exemplos de treinamento
 - Quantidade de ruído
 - Complexidade da função a ser aprendida
 - Distribuição estatística

Unidades intermediárias

- Número de neurônios nas camadas intermediárias (cont.)
 - Existem problemas com uma entrada e uma saída que precisam de milhares de unidades e vice-versa
 - Pode crescer exponencialmente com o número de entradas
 - Solução neural eficiente: aquela onde o número de unidades cresce apenas de forma polinomial em relação ao número de entradas

Unidades intermediárias

- Algumas direções podem ser encontradas na literatura, porém normalmente server somente para certas aplicações

Palmer, A., J. José Montaño, and A. Sesé, Designing an artificial neural network for forecasting tourism time series. *Tourism Management*, 2006. 27(5): p. 781-790.

Azoff, E.M., *Neural Network Time Series Forecasting of Financial Markets*. 1994: John Wiley & Sons, Inc. 196.

Kaastra, I. and M. Boyd, Designing a neural network for forecasting financial and economic time series. *Neurocomputing*, 1996. 10(3): p. 215-236.

Zhang, G., B. Eddy Patuwo, and M. Y. Hu, Forecasting with artificial neural networks:: The state of the art. *International Journal of Forecasting*, 1998. 14(1): p. 35-62.

Unidades intermediárias

- Alguns autores descrevem o comportamento das funções de transferência para ambos os neurônios e camadas, funções de cálculo de erro e funções de alteração de escala

Nelson, M.M. and W.T. Illingworth, A practical guide to neural nets. Vol. 1. 1991: Addison-Wesley Reading, MA.

Unidades intermediárias

- Vários trabalhos sugerem que uma ou duas camadas intermediárias com um número adequado de neurónios é suficiente para modelar quaisquer sistemas não-lineares e adicionar mais camadas não irá melhorar os resultados.

Lapades, A. and R. Ferber, How neural network works. Neural information processing systems, American Institute of Physics, New York, 1988: p. 442-456.

Hecht-Nielsen, R. Theory of the backpropagation neural network. in Neural Networks, 1989. IJCNN., International Joint Conference on. 1989. IEEE.

Unidades intermediárias: Sugestões

- A regra da pirâmide geométrica com neurônios de entrada n e neurônios de saída m , a camada escondida teria

$$NHN_{Pyramid\ Geometric} = \sqrt{(m \times n)}$$

- O número ideal de neurônios ocultos ainda pode variar de

$$\frac{1}{2} (NHN_{Pyramid\ Geometric}) \leq NHN_{Optimal\ Number} \leq 2 (NHN_{Pyramid\ Geometric})$$

Masters, T., 10 - Designing Feedforward Network Architectures, in Practical Neural Network Recipies in C++, T. Masters, Editor. 1993, Morgan Kaufmann: San Francisco (CA). p. 173-185.

Unidades intermediárias: Sugestões

- Dependendo da complexidade do problema o número pode aumentar.
- O número de neurônios escondidos em uma rede neuronal de três camadas deve ser igual a 75% do número de neurônios entradas

$$\frac{3}{4} n = NHN \textit{Optimal Number}$$

Bailey, D.L. and D. Thompson, Developing neural-network applications. AI Expert, 1990. 5(9): p. 34-41

Unidades intermediárias: Sugestões

- O número inicial de nós na camada escondida única pode ser calculada usando a equação desenvolvida por Carpenter e Hoffman:

$$Ts = \eta [NHN_{Optimal\ Number} (n + 1) + m (NHN_{Optimal\ Number} + 1)]$$

Onde, η é uma constante maior do que 1,0 ($\eta = 1,25$ daria um 25% aproximação), Ts o número de pares de formação disponíveis, H o número de nós escondidos para ser utilizados na rede com uma camada escondida, n e m , o número de nós de entrada e de saída, respectivamente.

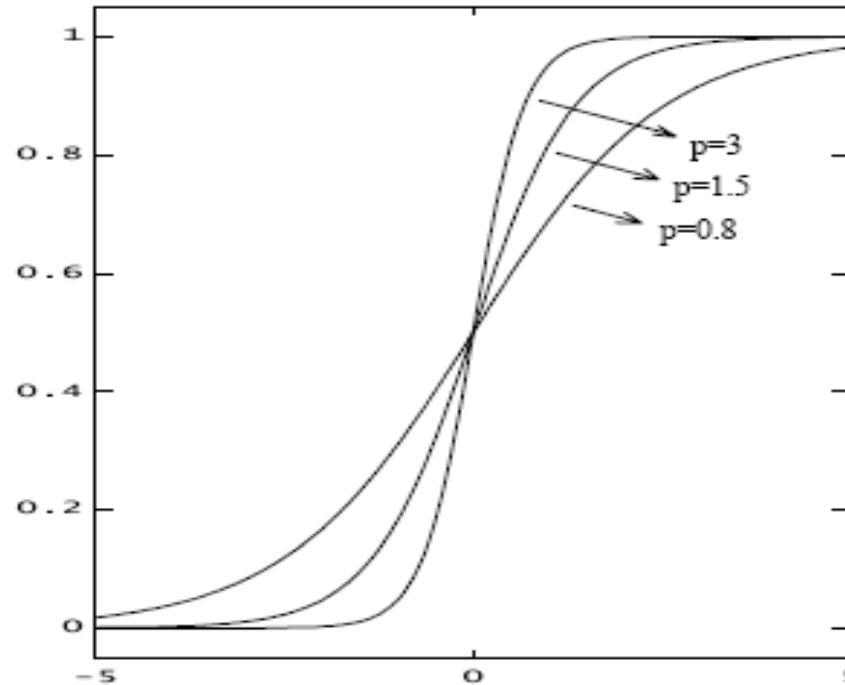
Carpenter, W.C. and M.E. Hoffman, Selecting the architecture of a class of back-propagation neural networks used as approximators. Artificial Intelligence for Engineering, Design, Analysis and Manufacturing, 1997. 11(01): p. 33-44.

Funções de ativação

- Evitar atingir valores muito grandes que podem saturar a rede neural artificial
- A função de transferência é geralmente não linear. A transformação linear não é apropriada quando as funções alvo não são linearmente separáveis.
- A abordagem de escalonamento de dados (entre 0 e 1 ou 1 e -1), deve ser compatível com o tipo de função de transferência selecionada.

Normalização dos dados de entrada

Uma característica das funções sigmoidais é a **saturação**, ou seja, para valores grandes de argumento, a função opera numa região de saturação.



É importante portanto trabalhar com valores de entrada que estejam contidos num intervalo que **não atinjam a saturação**, por exemplo, $[0,1]$ ou $[-1,1]$.

Inicialização dos vetores de pesos e bias

A eficiência do aprendizado em redes multicamadas depende da:

- especificação de arquitetura da rede,
- função de ativação,
- regra de aprendizagem, e
- valores iniciais dos vetores de pesos e bias.

Considerando-se que os três primeiros itens já foram definidos, verifica-se agora a **inicialização dos vetores de pesos e bias**.

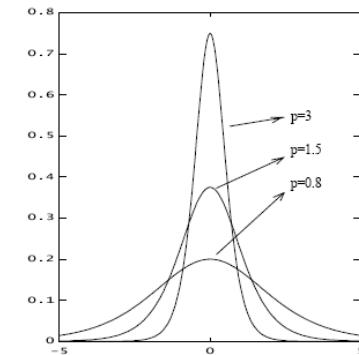
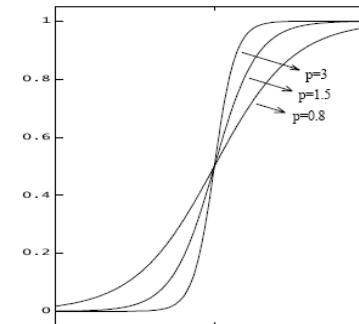
Inicialização aleatória dos pesos

A atualização de um peso entre duas unidades depende da derivada da função de ativação da unidade posterior e função de ativação da unidade anterior.

Por esta razão, é importante evitar escolhas de pesos iniciais que tornem as funções de ativação ou suas derivadas iguais a zero.

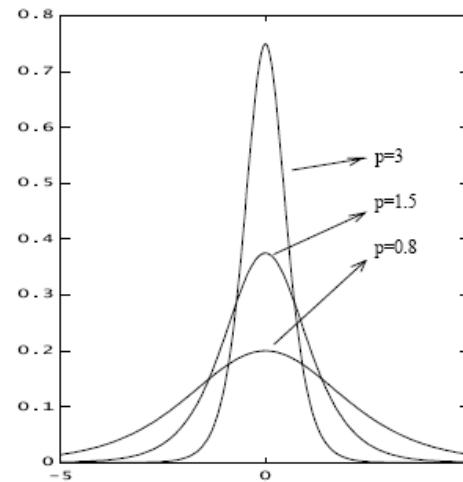
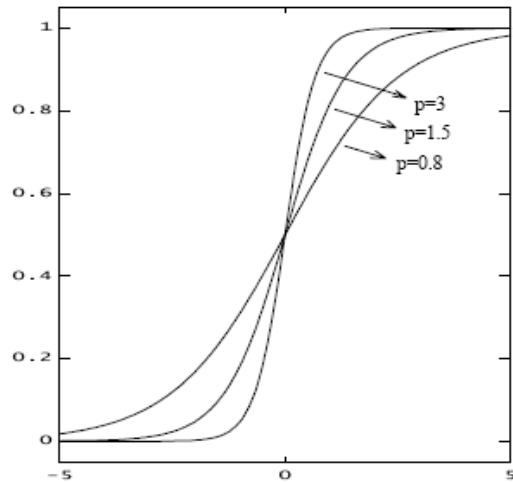
Os valores para os pesos iniciais não devem ser muito grandes, tal que as **derivadas** das **funções de ativação** tenham valores **muito pequenos** (região de saturação).

Por outro lado, se os **pesos iniciais** são **muito pequenos**, a soma pode cair perto de zero, onde o **aprendizado** é **muito lento**.



Inicialização aleatória dos pesos

Um **procedimento comum** é inicializar os pesos e *bias* a valores randômicos entre -0.5 e 0.5, ou entre -1 e 1. Os valores podem ser positivos ou negativos porque os pesos finais após o treinamento também podem ser positivos ou negativos.



Simon Haykin – Redes Neurais – Princípios e Prática, 2a. Edição, Ed. Artmed: Bookman, Porto Alegre, 1999.

Braga, A. P., Carvalho, A. C. P. L., Ludermir, T. B. Redes Neurais Artificiais: teoria e aplicações. LTC - Livros Técnicos e Científico, 2^a edição, 2007 p.260.

Some Useful Notation

We often need to talk about ordered sets of related numbers – we call them *vectors*, e.g.

$$\mathbf{x} = (x_1, x_2, x_3, \dots, x_n) , \quad \mathbf{y} = (y_1, y_2, y_3, \dots, y_m)$$

The components x_i can be added up to give a *scalar* (number), e.g.

$$s = x_1 + x_2 + x_3 + \dots + x_n = \sum_{i=1}^n x_i$$

Two vectors of the same length may be *added* to give another vector, e.g.

$$\mathbf{z} = \mathbf{x} + \mathbf{y} = (x_1 + y_1, x_2 + y_2, \dots, x_n + y_n)$$

Two vectors of the same length may be *multiplied* to give a scalar, e.g.

$$p = \mathbf{x} \cdot \mathbf{y} = x_1 y_1 + x_2 y_2 + \dots + x_n y_n = \sum_{i=1}^n x_i y_i$$

To avoid any ambiguity or confusion, we will mostly be using the component notation (i.e. explicit indices and summation signs) throughout this module.

The Power of the Notation : Matrices

We can use the same vector component notation to represent complex more things with many dimensions/indices. For two indices we have matrices, e.g. an $m \times n$ **matrix**

$$\mathbf{w} = \begin{pmatrix} w_{11} & w_{12} & \dots & w_{1n} \\ w_{21} & w_{22} & \dots & w_{1n} \\ \vdots & \vdots & & \vdots \\ w_{m1} & w_{m2} & \dots & w_{mn} \end{pmatrix}$$

Matrices of the same size can be **added** or **subtracted** component by component.

An $m \times n$ matrix **a** can be **multiplied** with an $n \times p$ matrix **b** to give an $m \times p$ matrix **c**. This becomes straightforward if we write it in terms of components:

$$c_{ik} = \sum_{j=1}^n a_{ij} b_{jk}$$

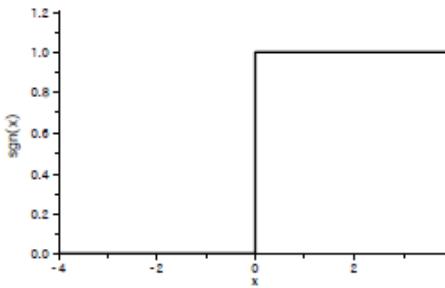
An n component vector can be regarded as a $1 \times n$ or $n \times 1$ matrix.

Some Useful Functions

A function $y = f(x)$ describes a relationship (input-output mapping) from x to y .

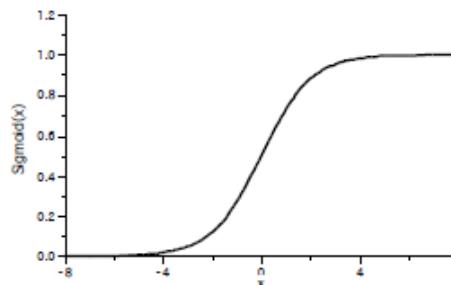
Example 1 The threshold or sign function $\text{sgn}(x)$ is defined as

$$\text{sgn}(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases}$$



Example 2 The logistic or sigmoid function $\text{Sigmoid}(x)$ is defined as

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



This is a smoothed (differentiable) form of the threshold function.

The McCulloch-Pitts Neuron Equation

Using the above notation, it is possible to write down a simple equation for the *output out* of a McCulloch-Pitts neuron as a function of its n *inputs* in_i :

$$out = \text{sgn}\left(\sum_{i=1}^n in_i - \theta\right)$$

where θ is the neuron's activation *threshold*. We can easily see that:

$$\begin{aligned} out &= 1 \quad \text{if } \sum_{k=1}^n in_k \geq \theta \\ out &= 0 \quad \text{if } \sum_{k=1}^n in_k < \theta \end{aligned}$$

Note that the McCulloch-Pitts neuron is an extremely simplified model of real biological neurons. Some of its missing features include: non-binary inputs and outputs, non-linear summation, smooth thresholding, stochasticity, and temporal information processing.

Nevertheless, McCulloch-Pitts neurons are computationally very powerful. One can show that assemblies of such neurons are capable of universal computation.

Leitura recomendada

1. Haykin: Sections 1.1, 1.2, 1.3
2. Ham & Kostanic: Sections 1.2, 1.3
3. Beale & Jackson: Sections 1.2, 3.1, 3.2
4. Gurney: Sections 2.1, 2.2.

Simon Haykin – Redes Neurais – Princípios e Prática, 2a. Edição, Ed. Artmed: Bookman, Porto Alegre, 1999.

Braga, A. P., Carvalho, A. C. P. L., Ludermir, T. B. Redes Neurais Artificiais: teoria e aplicações. LTC - Livros Técnicos e Científico, 2^a edição, 2007 p.260.