Analisi di Immagini e Video (Computer Vision)

Giuseppe Manco

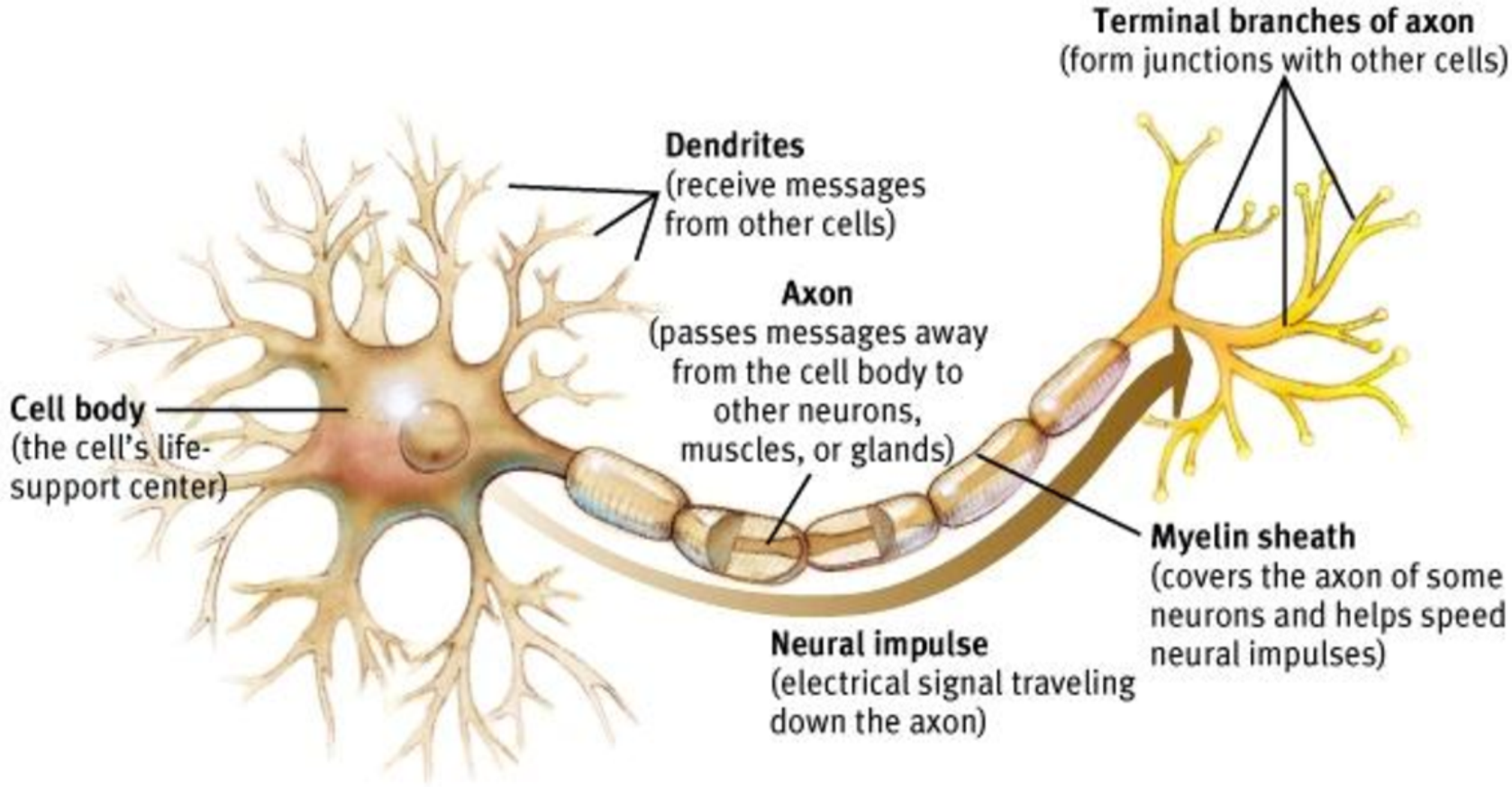
Outline

- Reti Neurali
- CNN

Crediti

- Slides adattate da vari corsi e libri
  - Deep Learning (Ettore Ritacco)
  - Deep Learning (Bengio, Courville, Goodfellow, 2017)
  - Andrey Karpathy
  - Computer Vision (I. Gkioulekas) - CS CMU Edu
  - Cmputational Visual Recognition (V. Ordonez), CS Virgina Edu

Oltre i modelli lineari

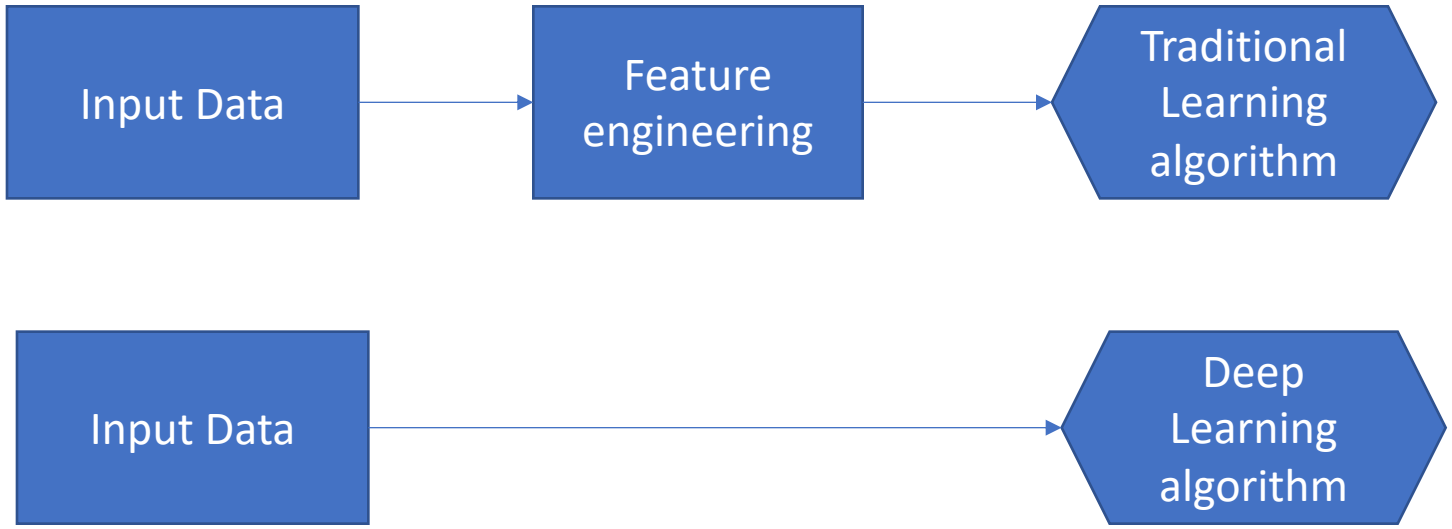

Terminal branches of axon
(form junctions with other cells)
Dendrites
(receive messages
from other cells)
Axon
(passes messages away
from the cell body to
other neurons,
muscles, or glands)
Cell body
(the cell's life-
support center)
Myelin sheath
(covers the axon of some
neurons and helps speed
neural impulses)
Neural impulse
(electrical signal traveling
down the axon)

- A biological neuron is a cell connected to other neurons and acts as a hub for electrical impulses. A neuron has a roughly spherical cell body called soma, which processes the incoming signals and converts them into output signals. Input signals are collected from extensions on the cell body called dendrites. These output signals are transmitted to other neurons through another extension called axon, which prolongs from the cell body and terminates into several branches. The branches end up into junctions transmitting signals from one neuron to another, called synapses.
- The behavior of a neuron is essentially electro-chemical. An electrical potential difference is maintained between the inside and the outside of the soma, due to different concentrations of sodium (Na) and potassium (K) ions. When a neuron receives inputs from a large number of neurons via its synaptic connections, there is a change in the soma potential. If this change is above a given threshold, it results in an electric current flowing through the axon to other cells. Then the potential drops down below the resting potential and neuron cannot fire again until the resting potential is restored.

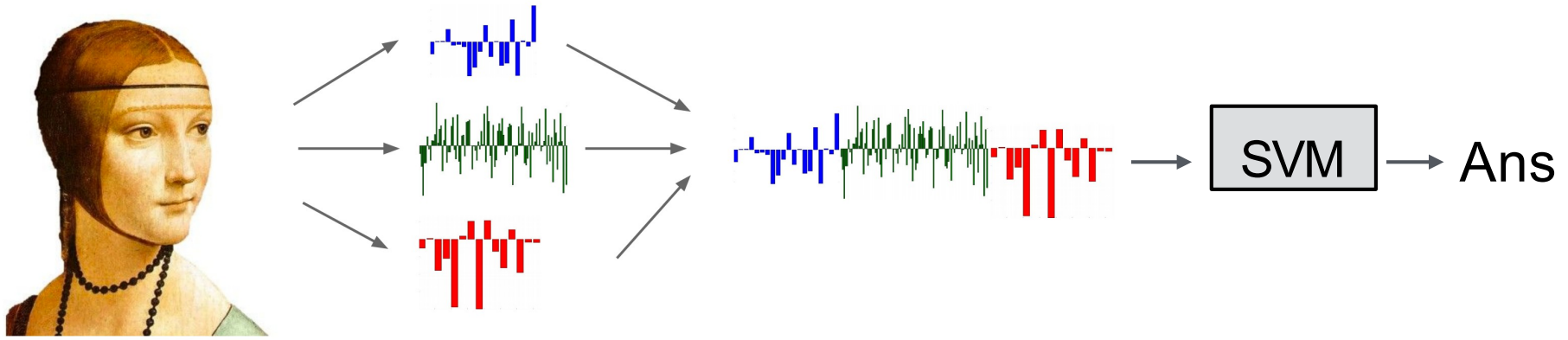
Deep Learning

- Parte del machine learning
- Apprende rappresentazioni dei dati
- Utilizza una gerarchia di layers che imitano il comportamento dei neuroni nel cervello

Senza feature engineering

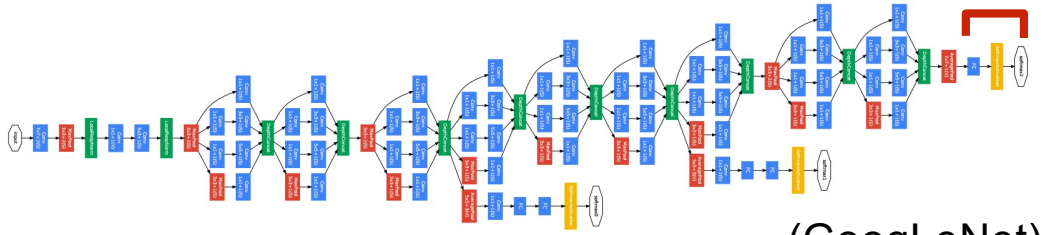


Representation learning



Input Pixels *Features* *concatenazione* *Linear Classifier*

Representation learning



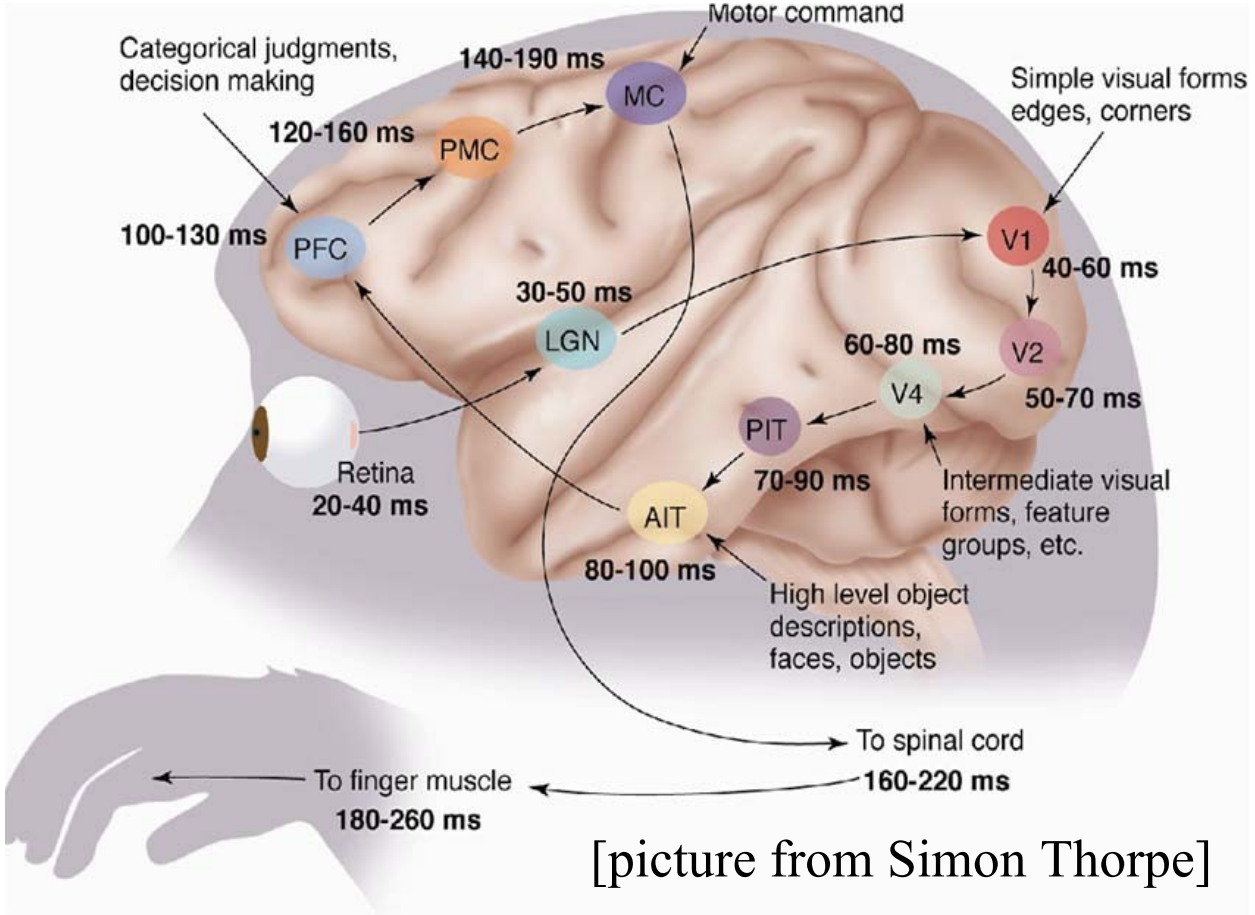
(GoogLeNet)

Ans

Input Pixels

Gli strati della rete apprendono le features automaticamente

- The ventral (recognition) pathway in the visual cortex has multiple stages
  - Retina - LGN - V1 - V2 - V4 - PIT - AIT ....
- Lots of intermediate representations



[picture from Simon Thorpe]

Perceptron Learning

- Funzione di base:

$$y = \sigma(a)$$

$$\mathrm{a} = \sum_{i=1}^{n} w_i x_i + b$$

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

  - Classificazione binaria
  - Separabilità lineare
- Due estensioini:
  - K classi
  - Relazioni nonlineari

Estensioni

- K classi

$$y_k = f(a_k)$$

$$a_k = \sum_{i=1}^{n} w_{ki} x_i + b_k$$

$$\boldsymbol{a} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$$

- Relazioni nonlineari

$$\boldsymbol{y} = f(\boldsymbol{a})$$

$$a_k = \sum_{i=1}^{n} w_{ki} \phi(x_i) + b_k$$

$$\boldsymbol{a} = \boldsymbol{W}\phi(\boldsymbol{x}) + \boldsymbol{b}$$

Format generale

- Una ennupla:

$$net = \{g, l, o, i, \text{fpp}\}$$

g Il grafo

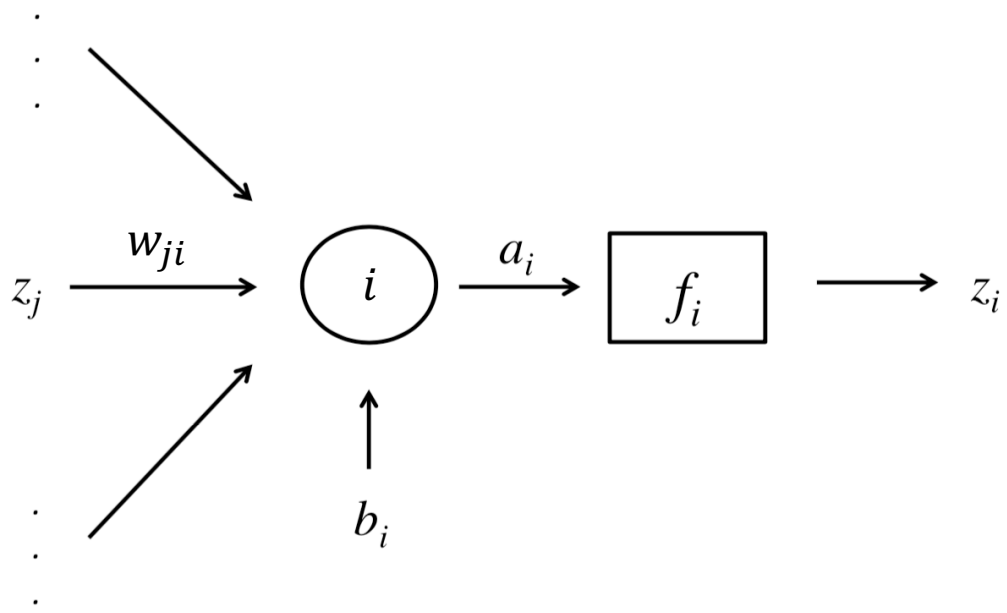
… network topology and operator

g il grafo

- $g = \{N, E\}$ è un grafo diretto pesato
- Ogni nodo $i \in N$ è un perceptron
  - È caratterizzato da due elementi
    - Un valore a_i, (l'attivazione)
    - Una funzione di attivazione f_i
      - Applicata all'attivazione, produce l'output z_i
- Un arco $e = \{j \in N \rightarrow i \in N\} \in E$ è associato a un peso w_{ji}
- Ogni nodo i è associato ad un arco speciale ad un nodo fantasma, il cui peso b_i è chiamato bias

g Il grafo

- Ogni neurone è una unità di calcolo



$$z_i = f_i(a_i)$$

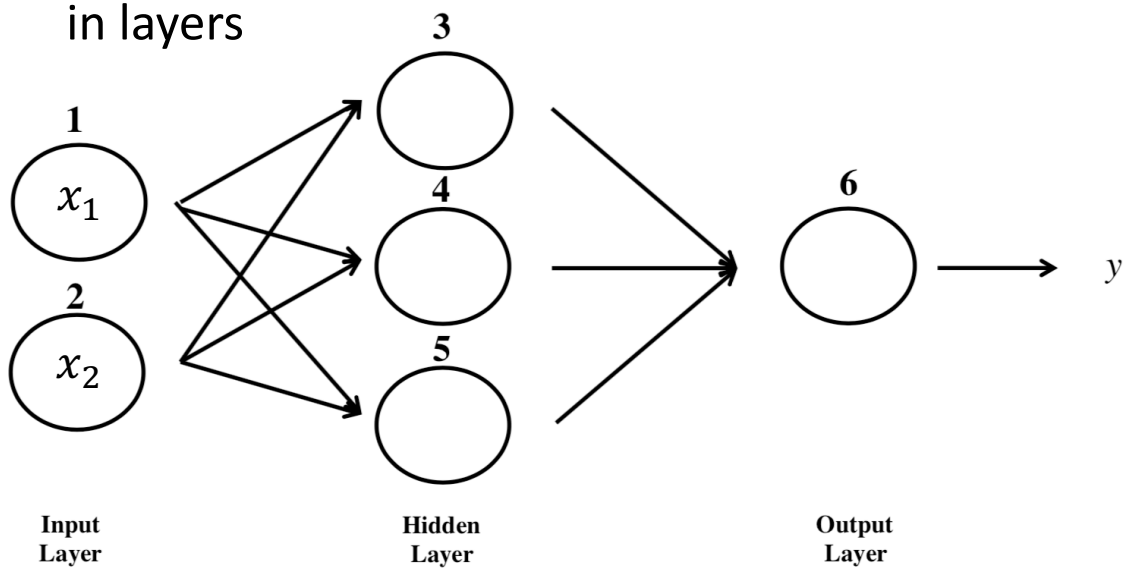
$$a_i = b_i + \sum_{j:j \to i \in E} w_{ji} z_j$$

g Il grafo

- Tre categorie di nodi
  - Input
    - I valori sono ”sovrascritti” dall’esterno
  - Hidden
    - Unità di calcolo
  - Output
    - Forniscono valori all’esterno

g Il grafo

- Combinazione di neuroni connessi → Calcolo complesso → Operazione
  - Nodi che condividono lo stesso input sono strutturati in layers



$$z_1 = x_1$$

$$z_2 = x_2$$

$$z_3 = f_3\left(b_3 + \sum_{j:j\to 3\in E} w_{j3} z_j\right)$$

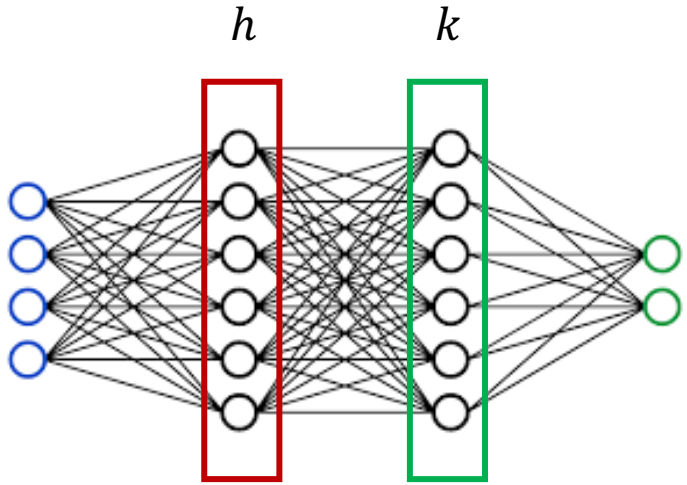
$$z_4 = f_4\left(b_4 + \sum_{j:j\to 4\in E} w_{j4} z_j\right)$$

$$z_5 = f_5\left(b_5 + \sum_{j:j\to 5\in E} w_{j5} z_j\right)$$

$$y = z_6 = f_6\left(b_6 + \sum_{j:j\to 6\in E} w_{j6} z_j\right)$$

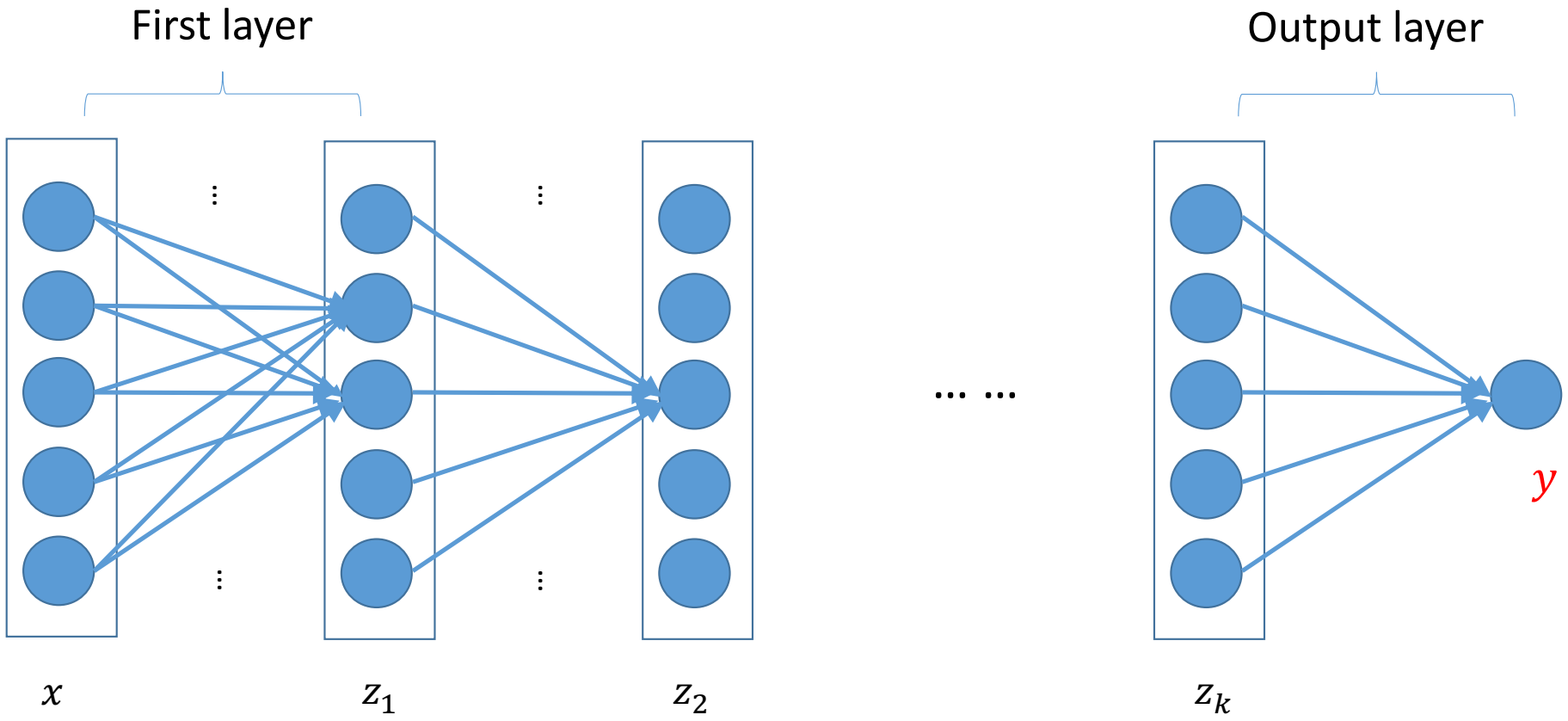
$$y = f_6\left(b_6 + w_{5,6} f_5\left(b_5 + w_{1,5}x_1 + w_{2,5}x_2\right) + w_{4,6} f_4\left(b_4 + w_{1,4}x_1 + w_{2,4}x_2\right) + w_{3,6} f_3\left(b_3 + w_{1,3}x_1 + w_{2,3}x_2\right)\right)$$

g Il grafo



- Notazione compatta:
  - Dati due layer consecutivi k e h:
  - $\boldsymbol{z_k} = f_h(\boldsymbol{b_h} + \boldsymbol{W z_h})$
- Nota: tutti i nodi condividono la stessa funzione di attivazione f_h
- W è la matrice dei pesi associate agli archi

Feed-Forward Networks



Input

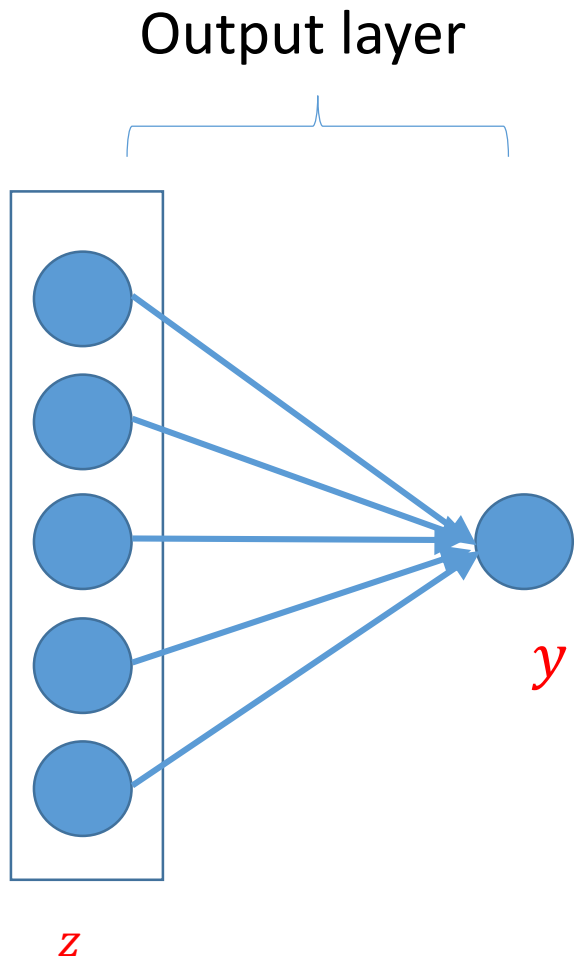
- Rappresentato come vettore

Expand

Output Layers

- Regressione:

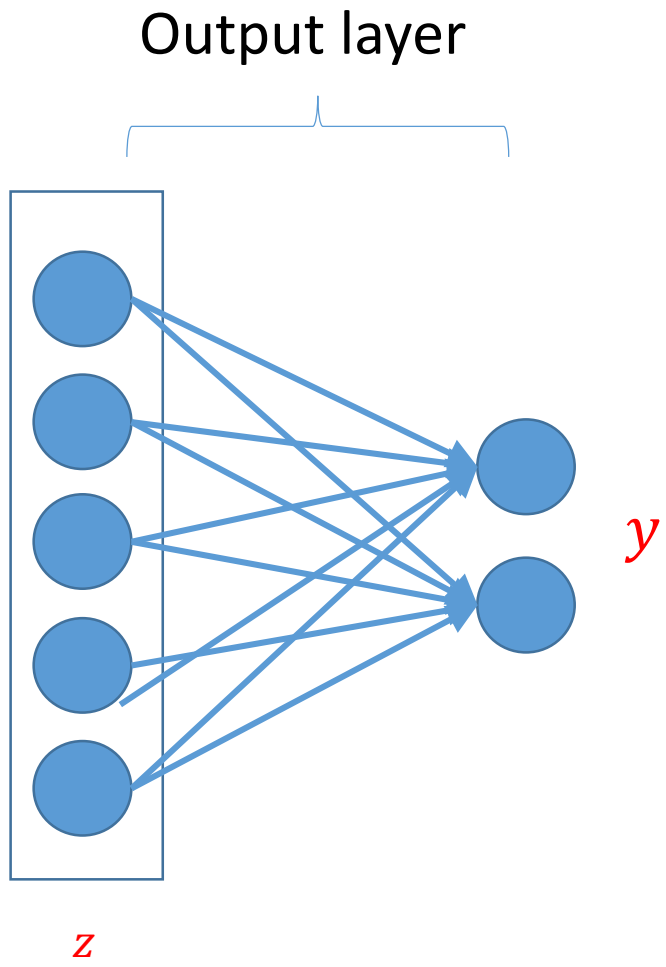
$$y = \boldsymbol{W}^T \boldsymbol{z} + \boldsymbol{b}$$



Output layers

- Regressione multidimensionale:

$$\boldsymbol{y = W^T z + b}$$

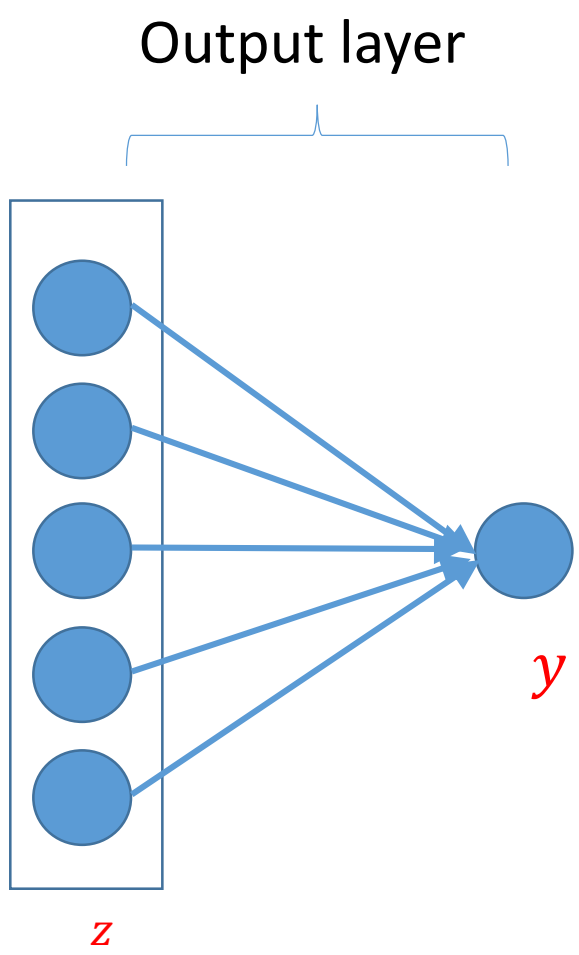


Output layers

- Classificazione binaria:

$$y = \sigma(\boldsymbol{W}^T \boldsymbol{z} + \boldsymbol{b})$$

  - Regressione logistica su z

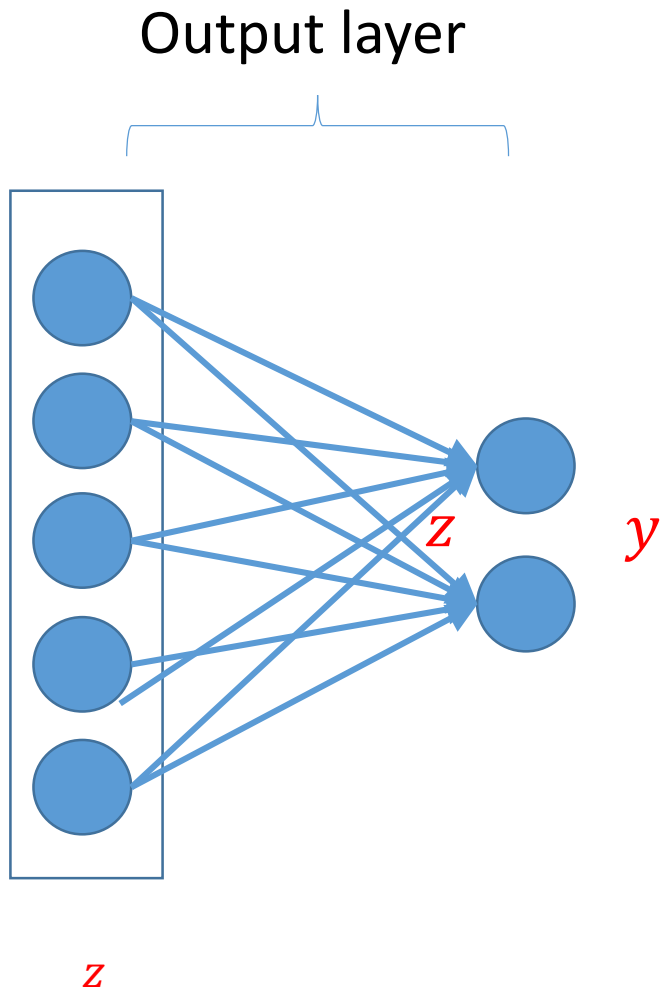


Output layers

- Classificazione multiclasse:

$$y = softmax(W^t z + b)$$

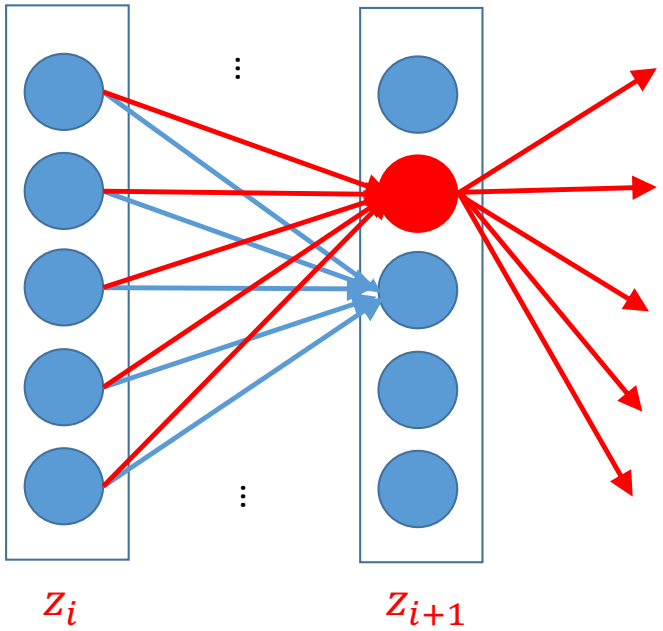
$$softmax_j(a) = \frac{e^{a_j}}{\sum_{i=1}^{k} e^{a_i}} = \frac{e^{a_j - a_{max}}}{\sum_{i=1}^{k} e^{a_i - a_{max}}}$$



Hidden layers

- Ogni neurone è una combinazione dei layer precedenti

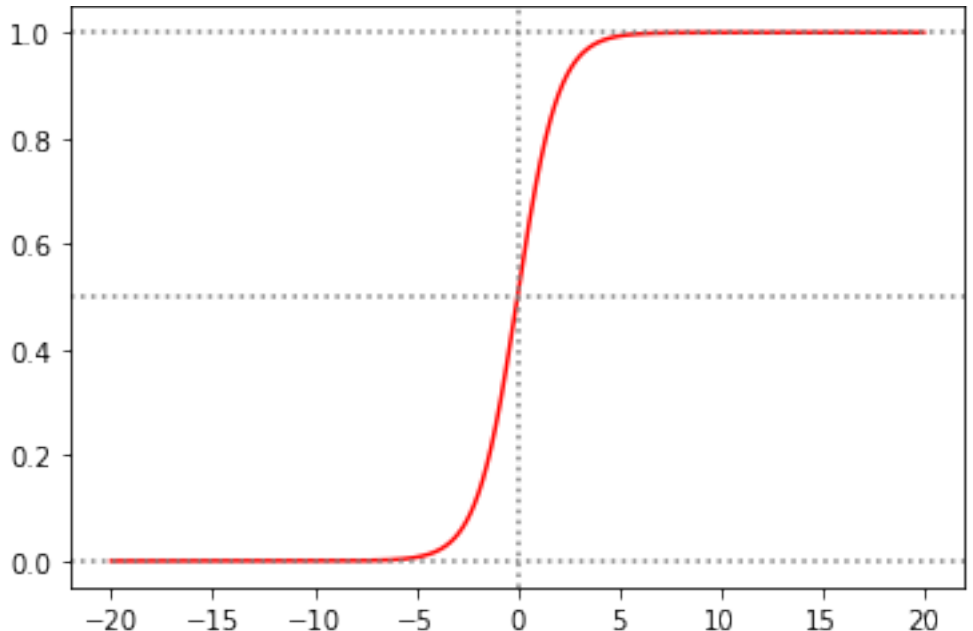
$$\boldsymbol{z_{i+1}} = f_i(\boldsymbol{W_i^{\mathrm{T}} z_i} + \boldsymbol{b_i})$$



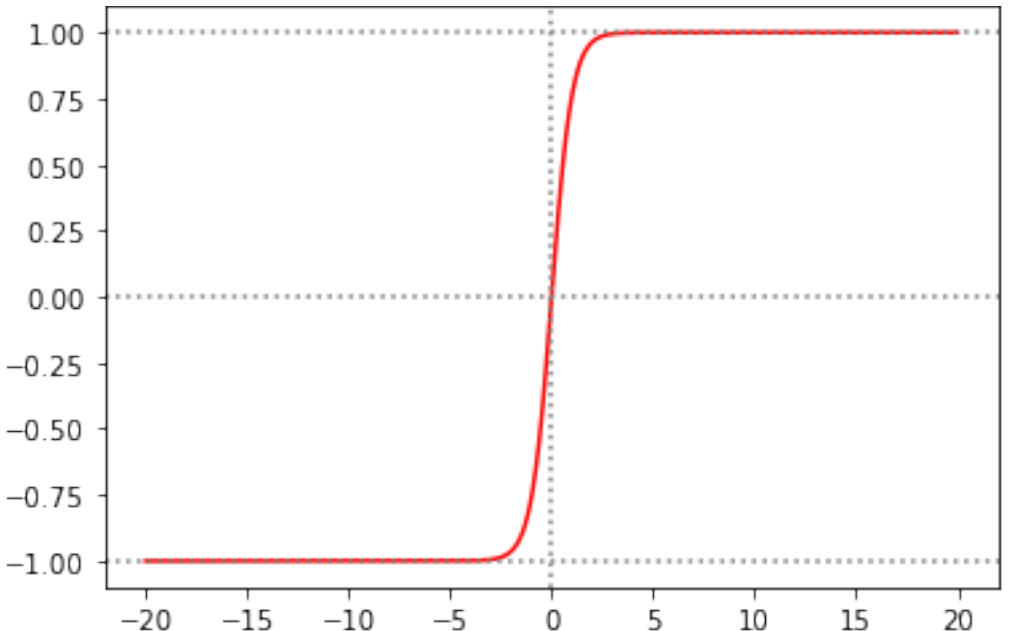
Funzioni di attivazione

- La forma di f_i ha una grossa influenza sui risultati
- Storicamente, $f(a)$ ha preso due forme

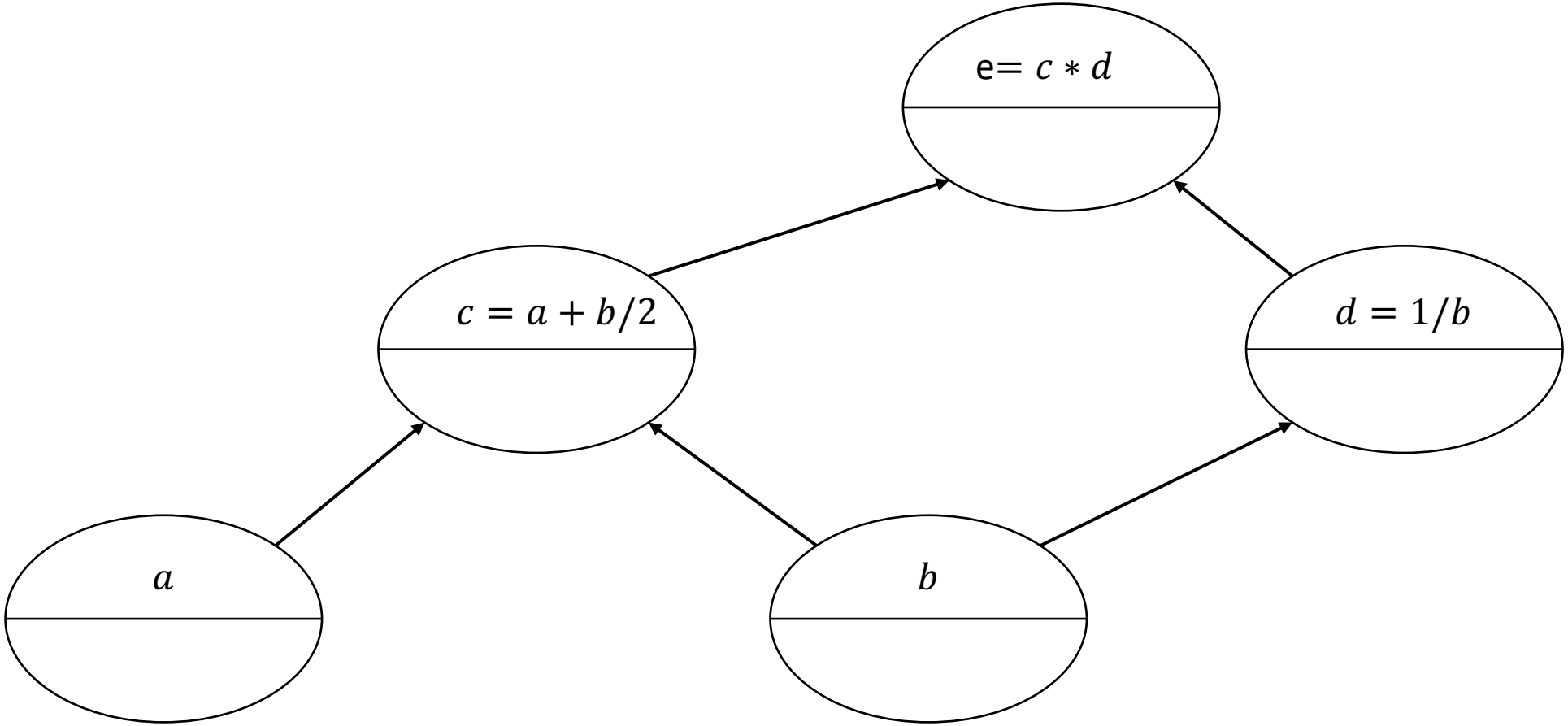
$$\sigma(a) = \frac{1}{1 + e^{-a}}$$



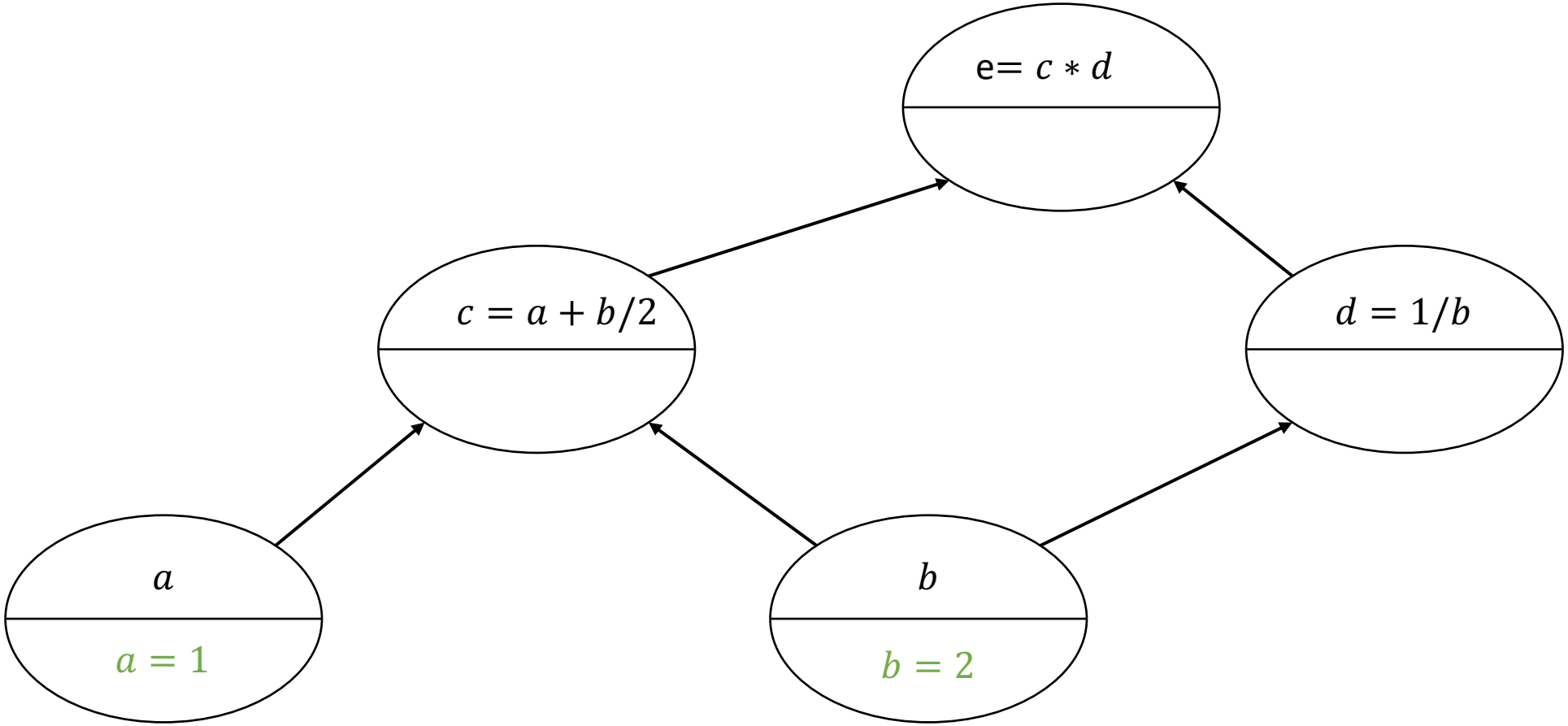
$$tanh(a) = \frac{1 - e^{-2a}}{1 + e^{-2a}}$$



Gradient Computing

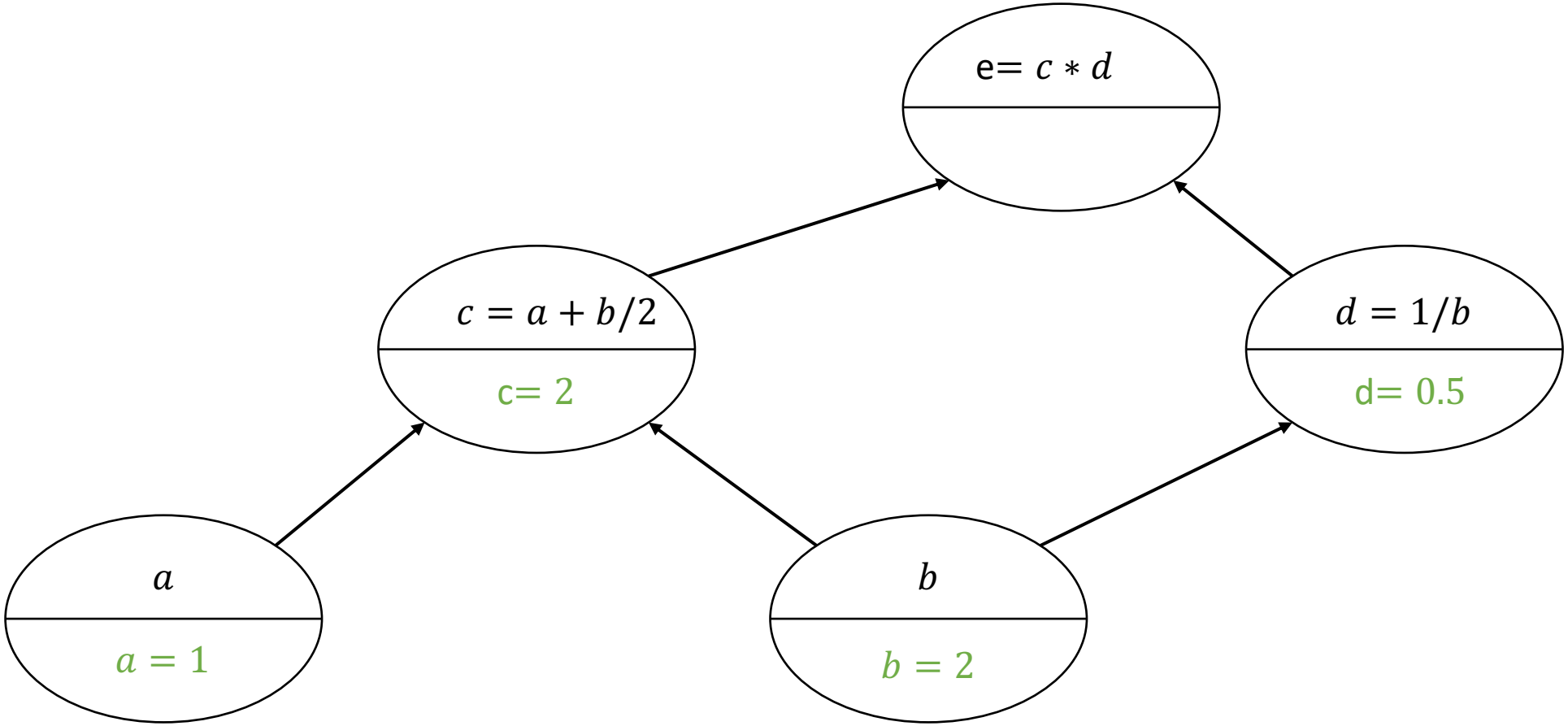


Gradient Computing



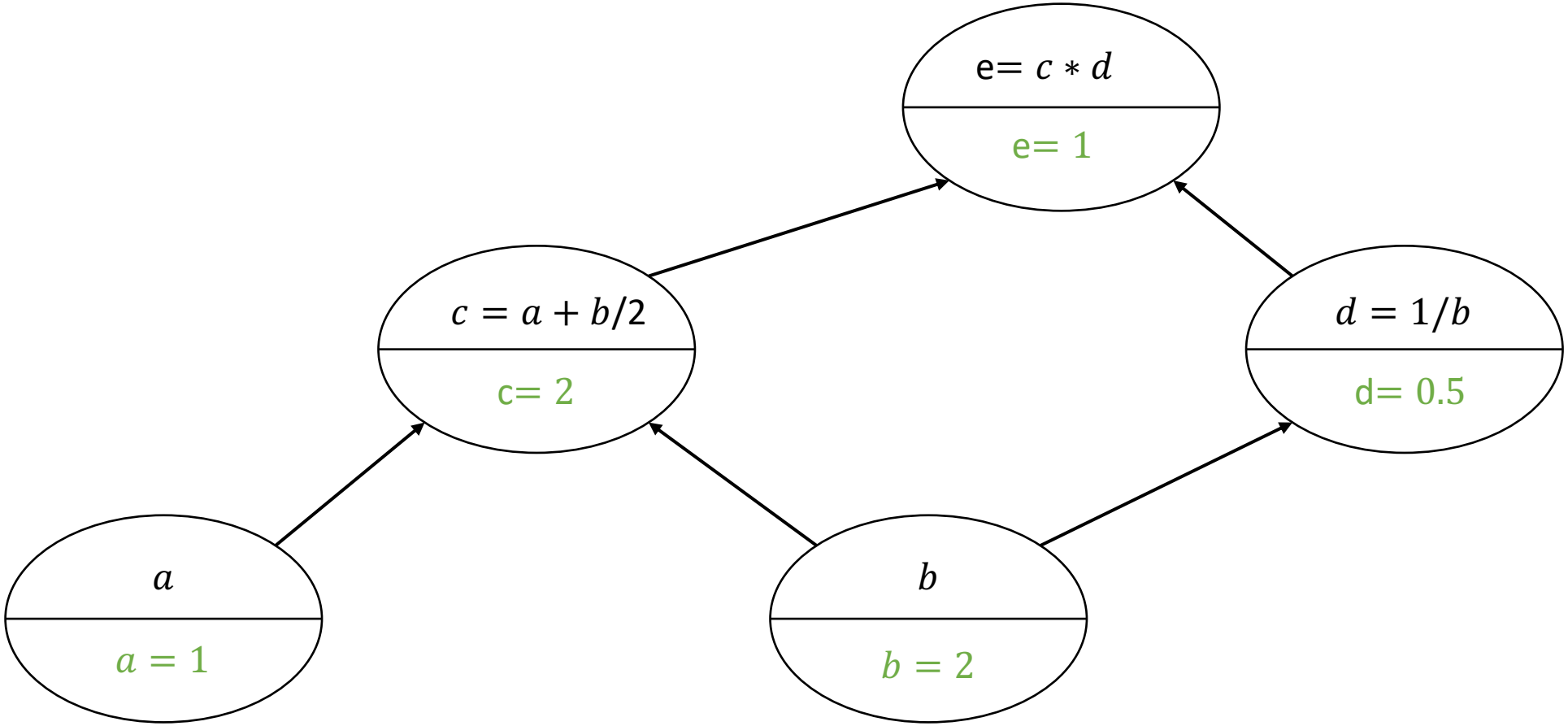
Forward pass

Gradient Computing



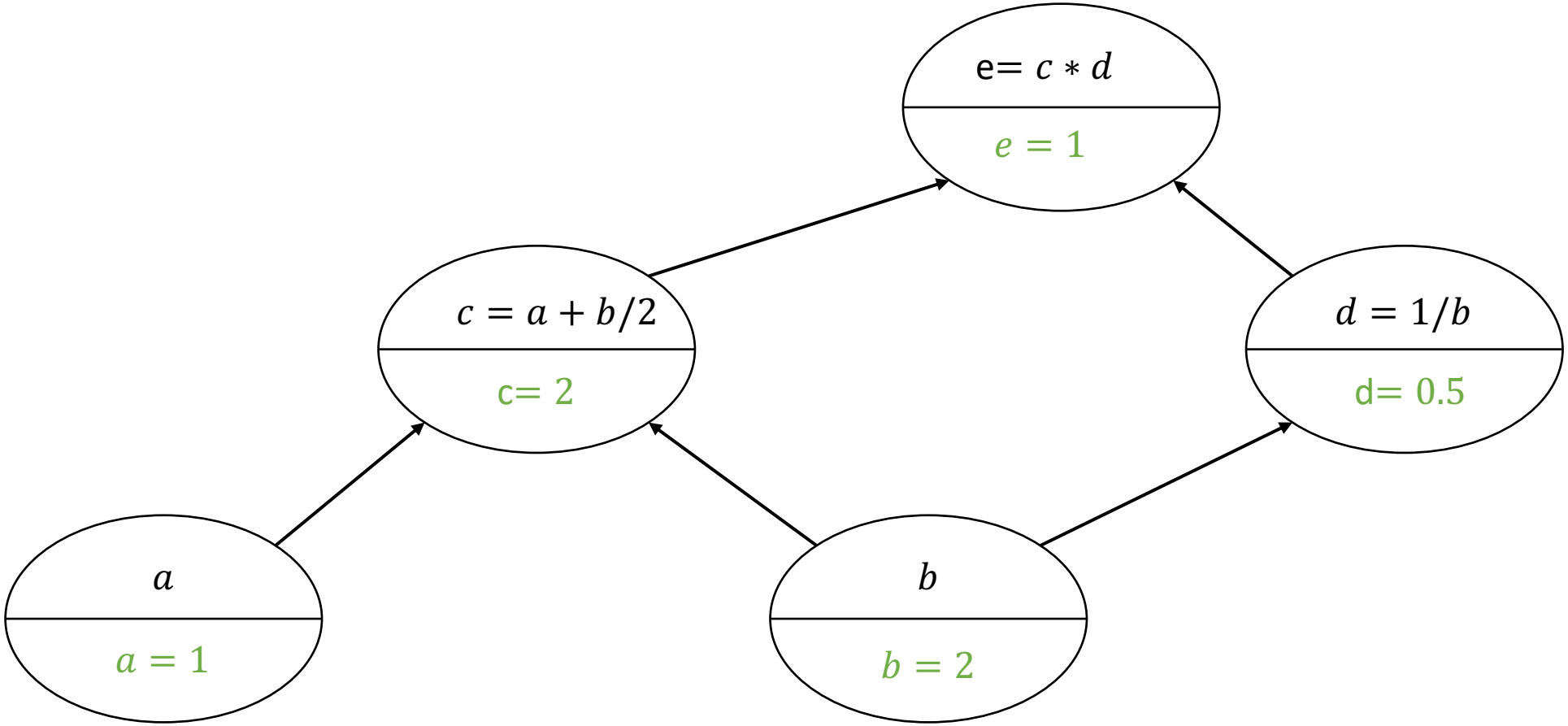
Forward pass

Gradient Computing



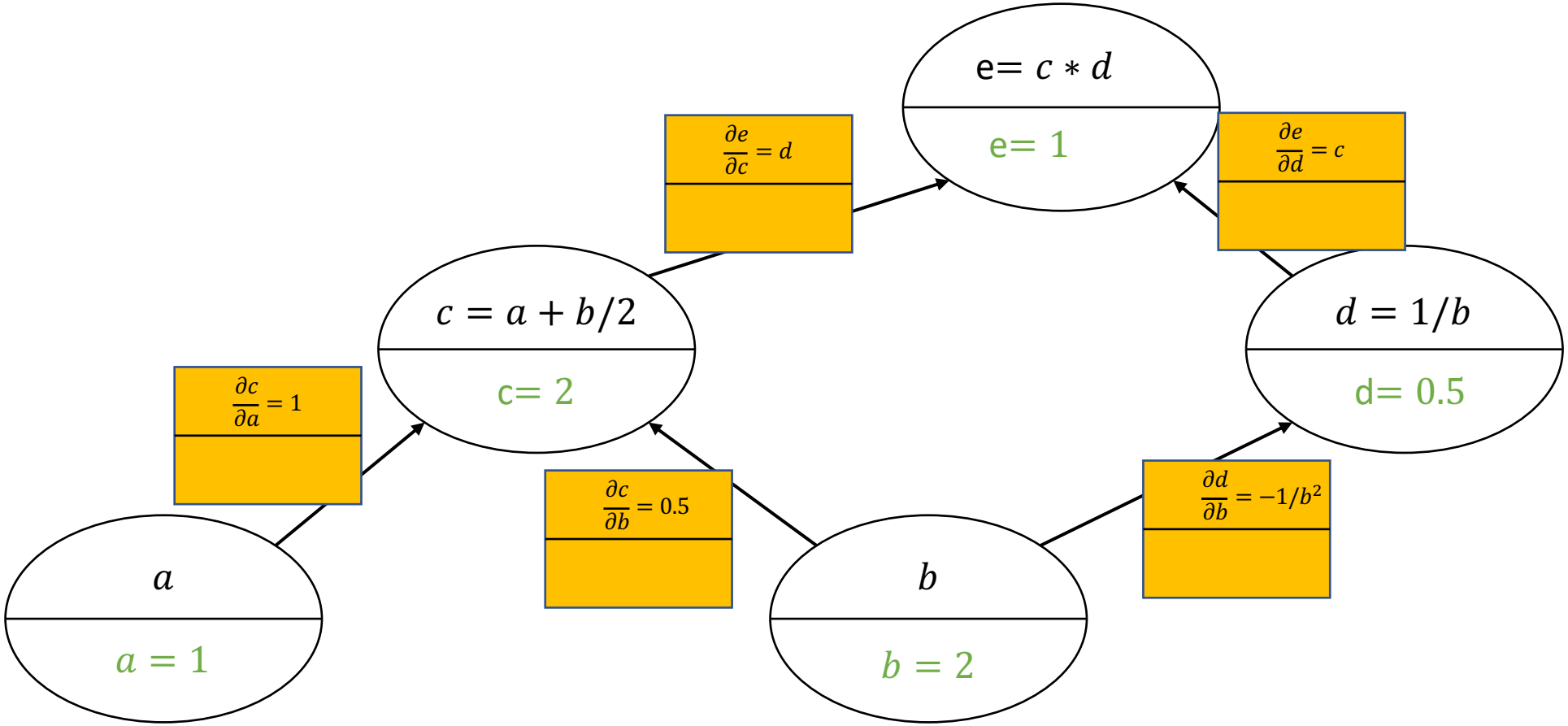
Forward pass

Gradient Computing



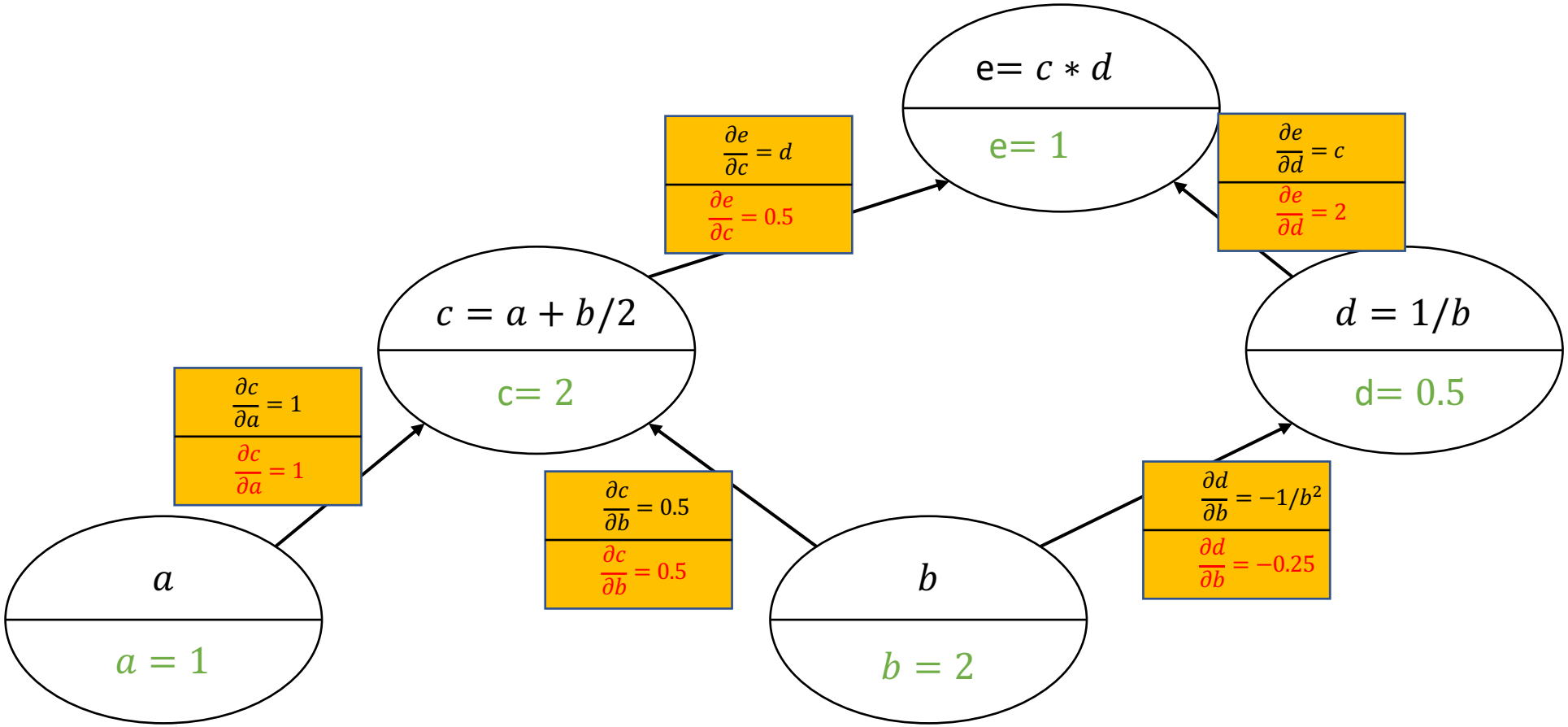
Backward pass

Gradient Computing



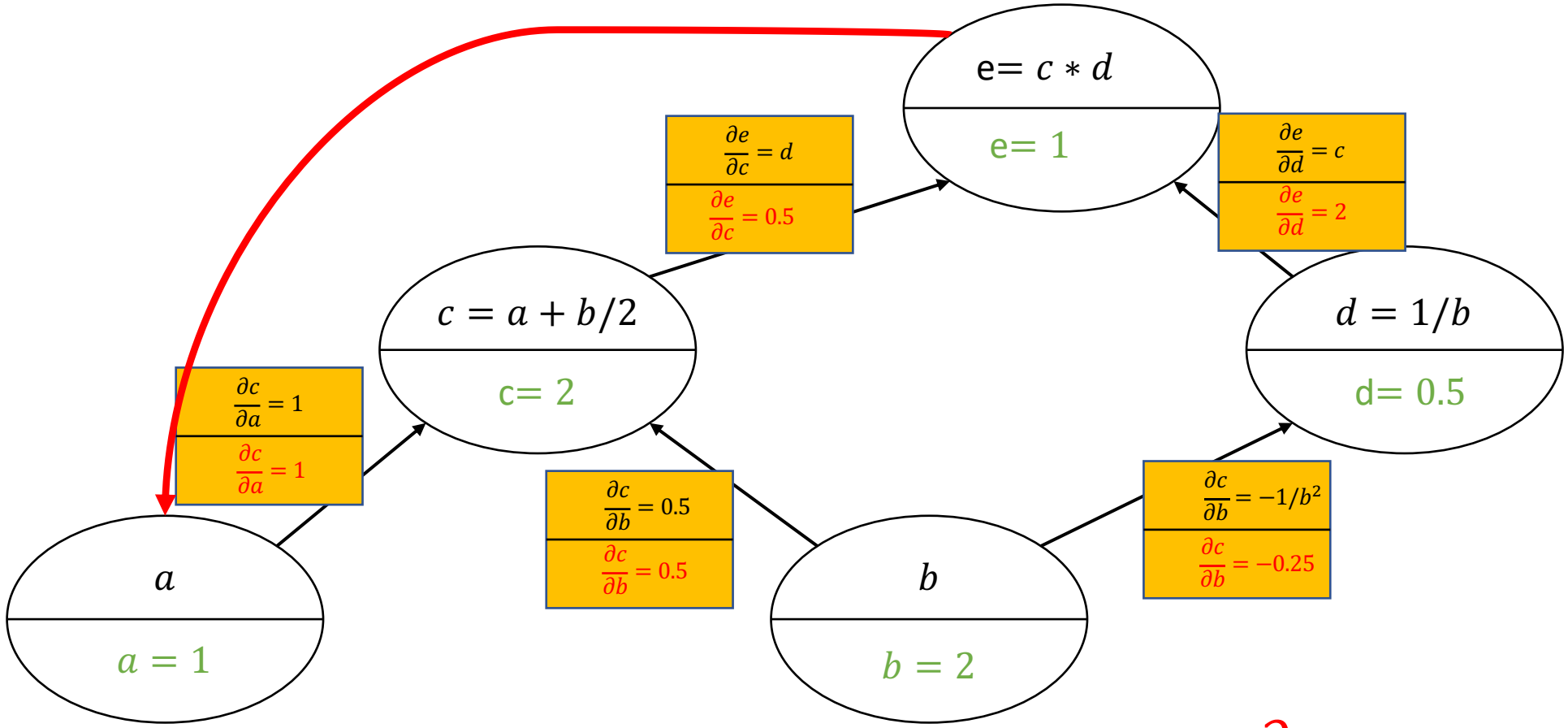
Backward pass

Gradient Computing



Backward pass

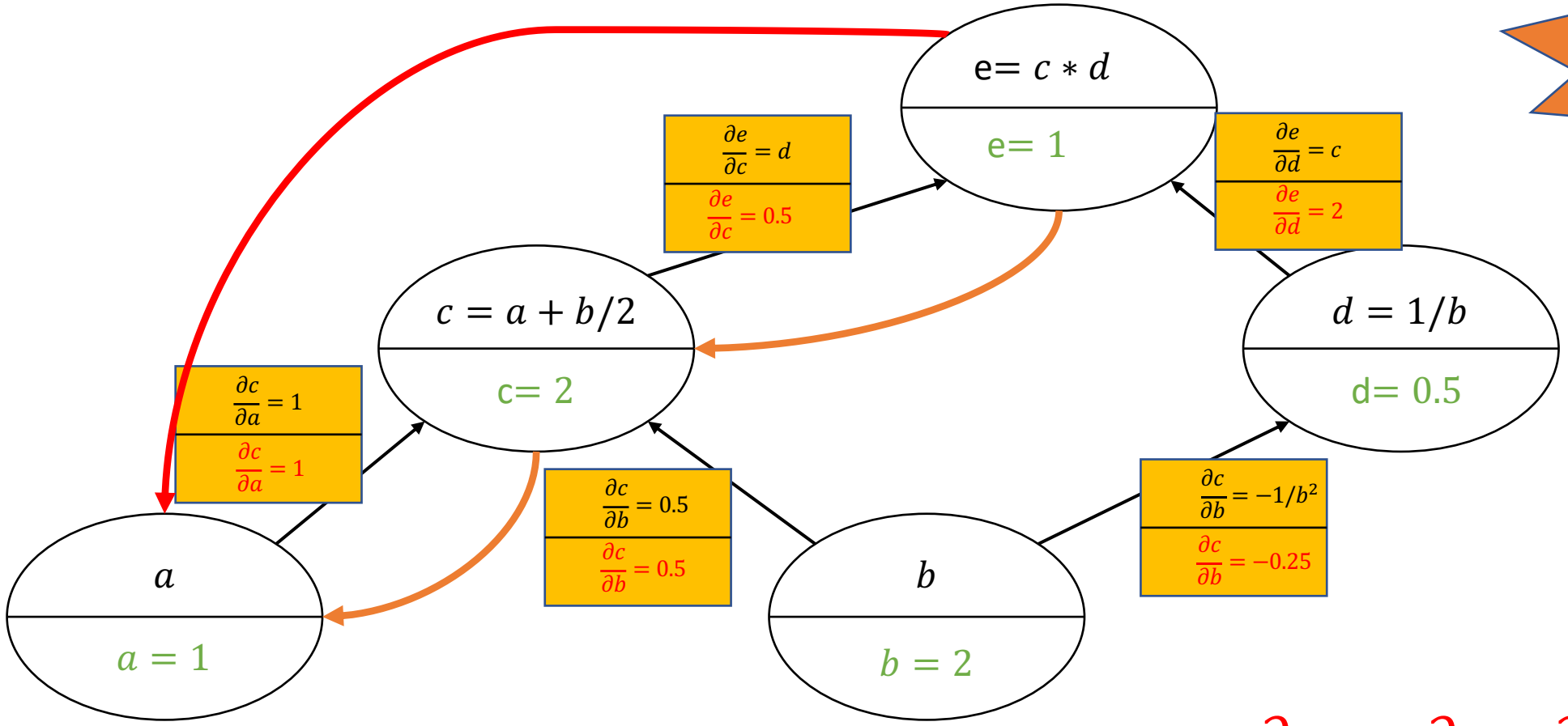
Gradient Computing



$$\frac{\partial e}{\partial a} = ?$$

Backward pass

Gradient Computing

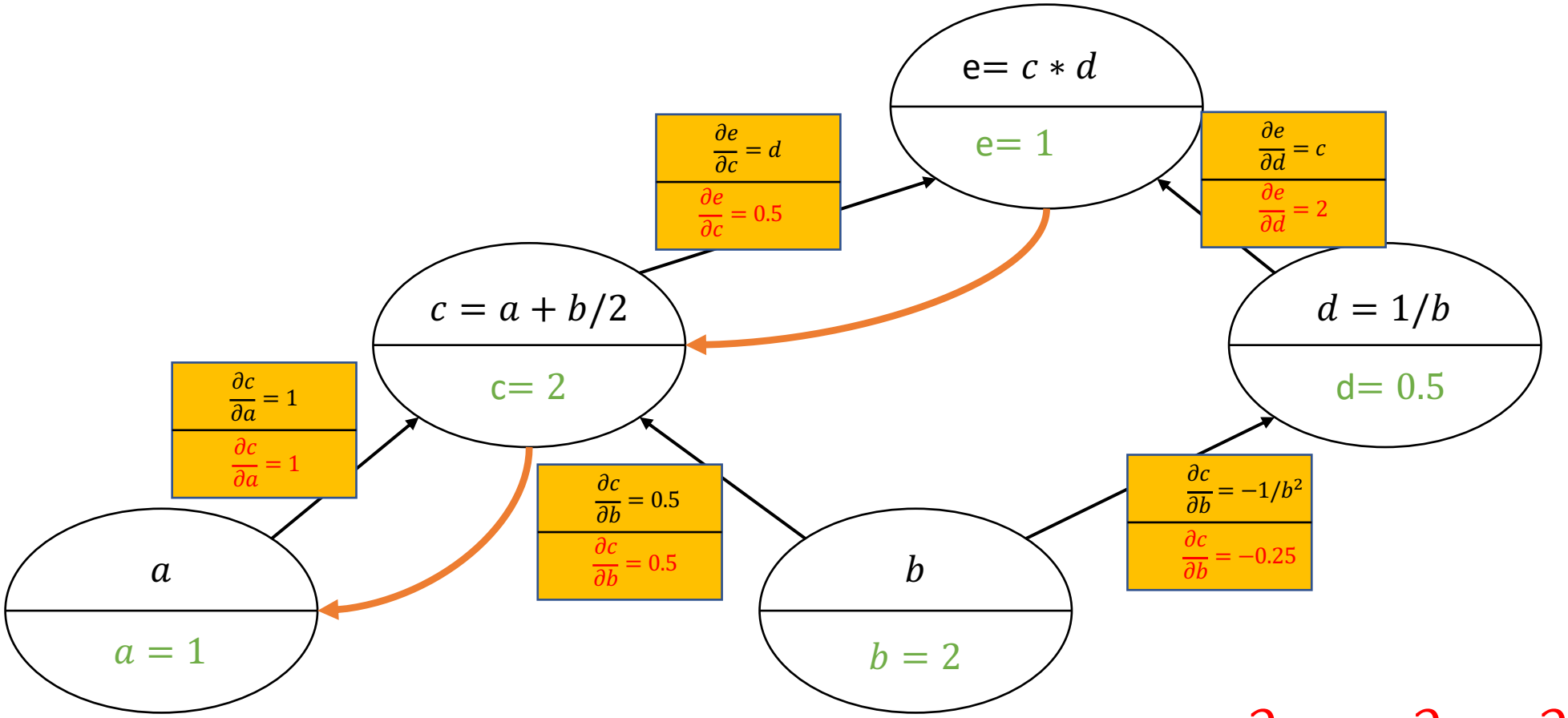


Chain rule!

$$\frac{\partial e}{\partial a} = \frac{\partial e}{\partial c} \cdot \frac{\partial c}{\partial a}$$

Backward pass

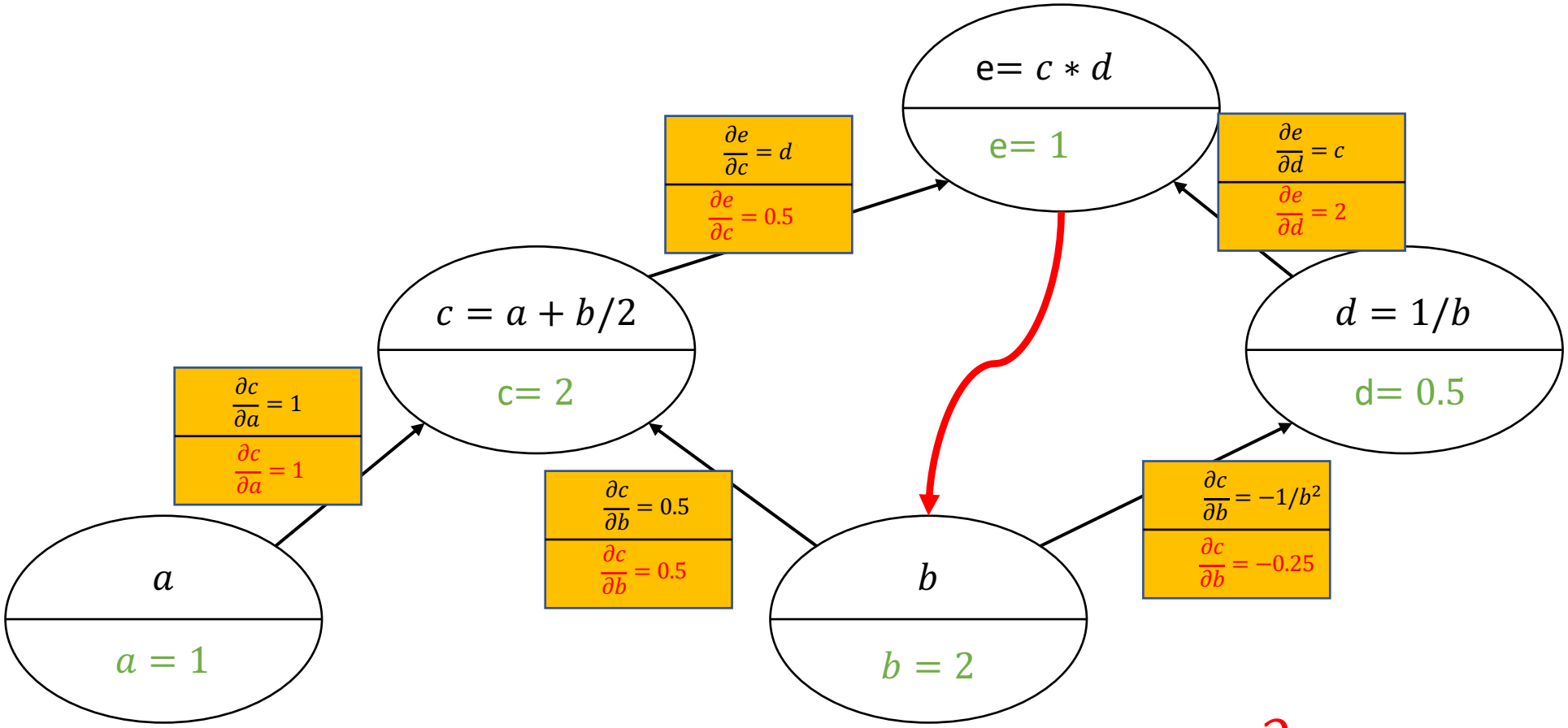
Gradient Computing



Backward pass

$$\frac{\partial e}{\partial a} = \frac{\partial e}{\partial c} \cdot \frac{\partial c}{\partial a} = 0.5$$

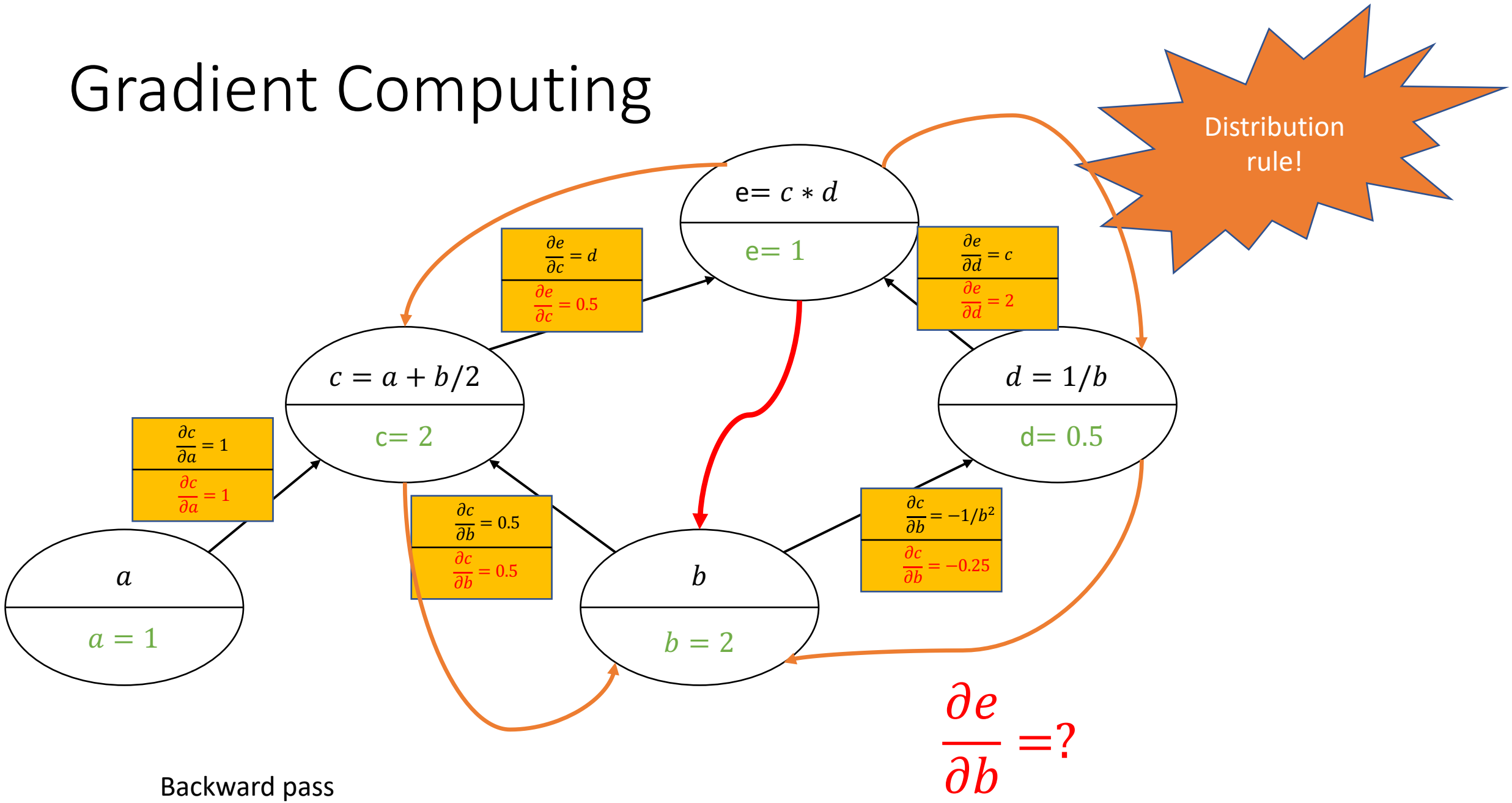
Gradient Computing



$$\frac{\partial e}{\partial b} = ?$$

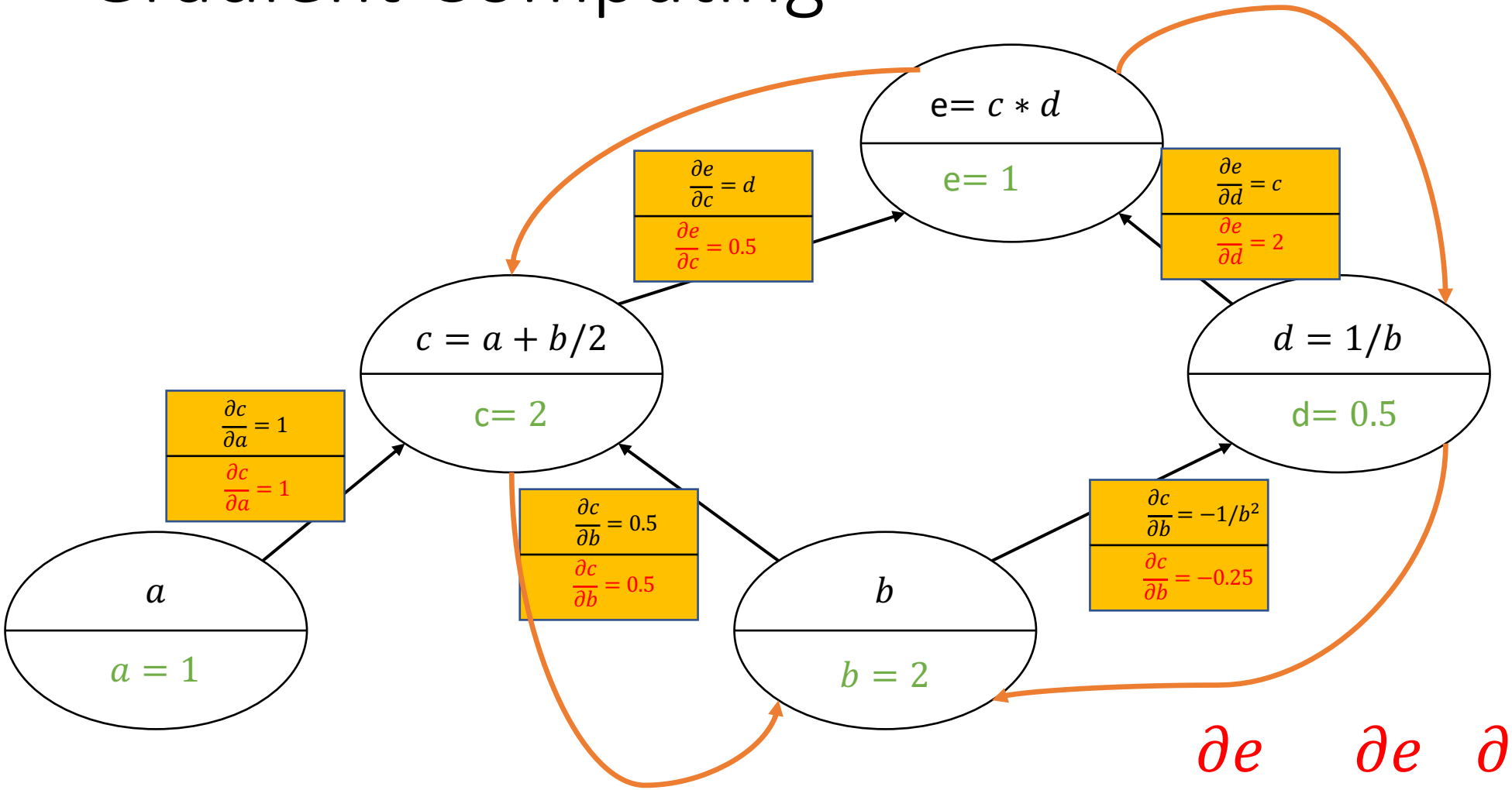
Backward pass

Gradient Computing



Backward pass

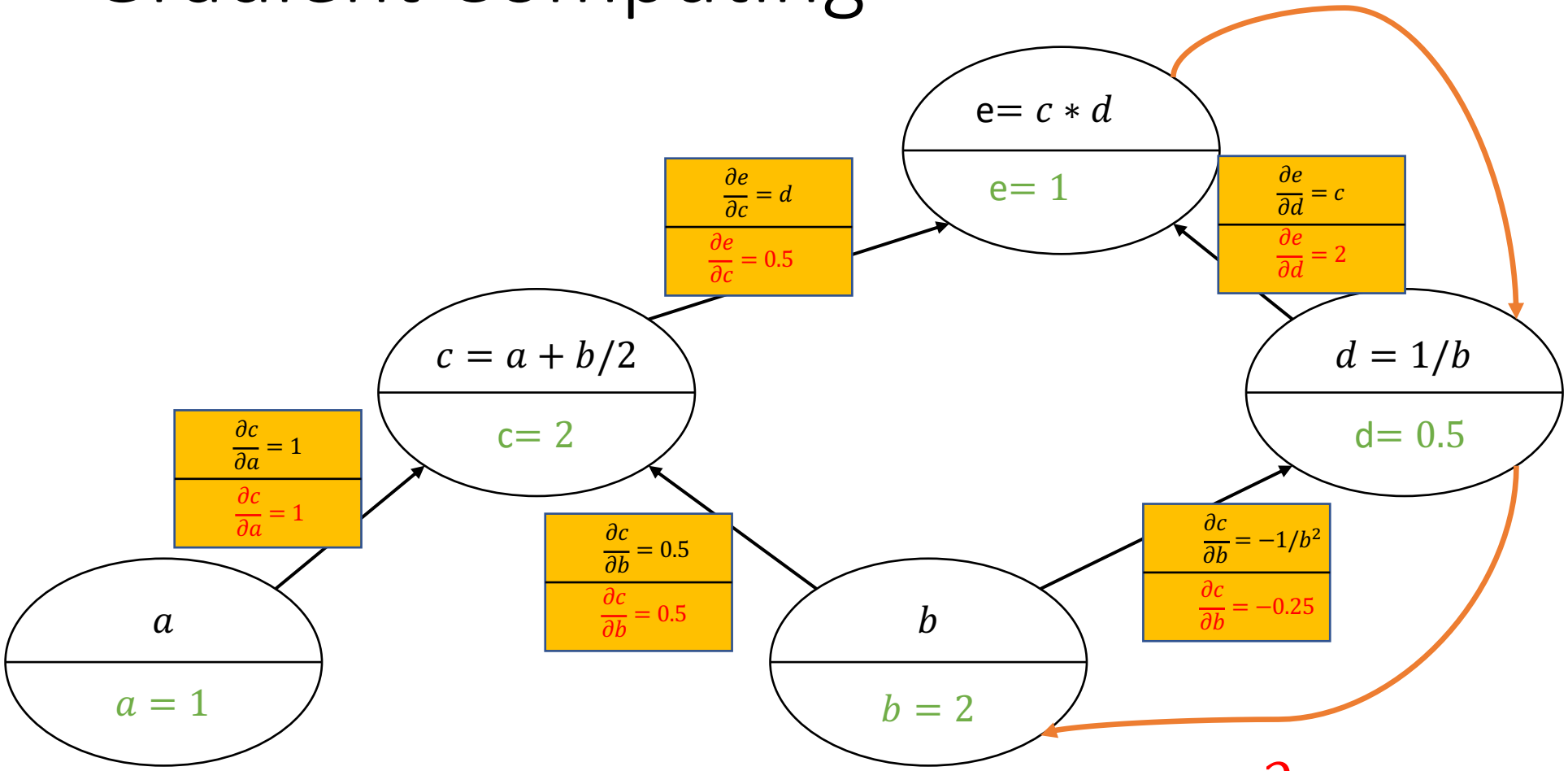
Gradient Computing



Backward pass

$$\frac{\partial e}{\partial b} = \frac{\partial e}{\partial c} \cdot \frac{\partial c}{\partial b} + \frac{\partial e}{\partial d} \cdot \frac{\partial d}{\partial b}$$

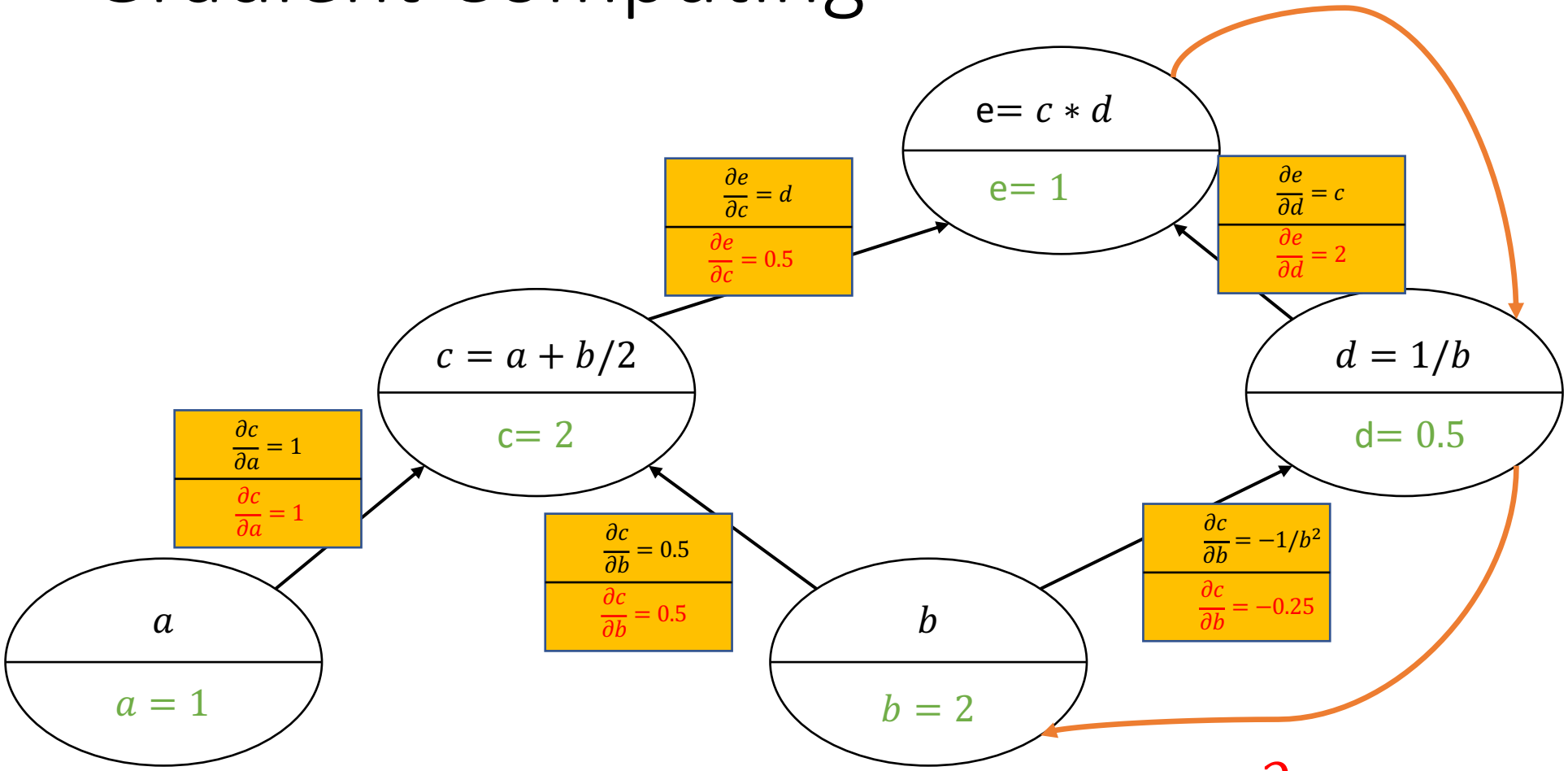
Gradient Computing



$$\frac{\partial e}{\partial b} = 0.5 \cdot 0.5 - 2 \cdot 0.25$$

Backward pass

Gradient Computing

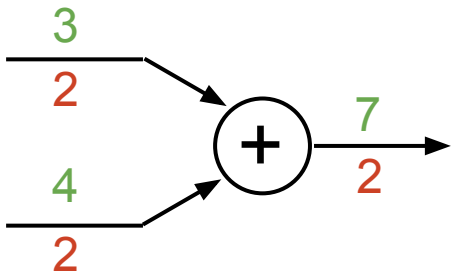


$$\frac{\partial e}{\partial b} = -0.75$$

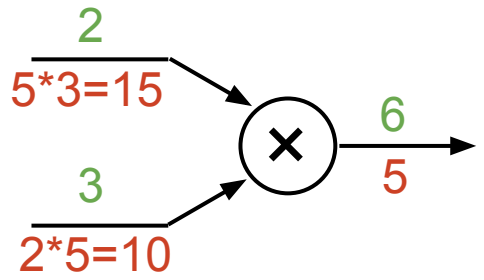
Backward pass

Patterns nel flow dei gradienti

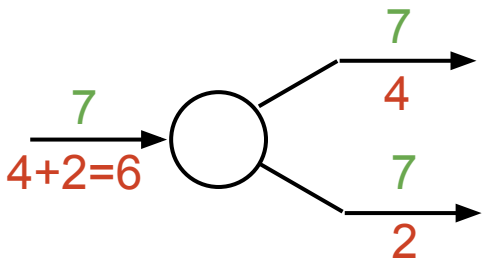
add gate: gradient distributor



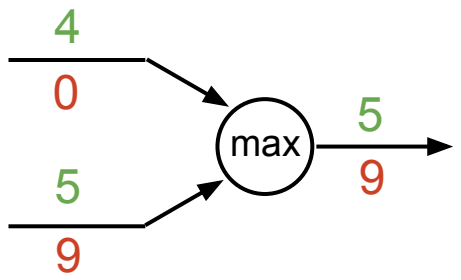
mul gate: “swap multiplier”



copy gate: gradient adder



max gate: gradient router



Derivate vettoriali sui nodi

- Input scalare, output scalare

$$x \in \mathbb{R}, y \in \mathbb{R}$$

- Derivata

$$\frac{\partial y}{\partial x} \in \mathbb{R}$$

Derivate vettoriali sui nodi

- Input vettore, output reale

$$x \in \mathbb{R}^N, y \in \mathbb{R}$$

- Gradiente

$$\frac{\partial y}{\partial x} \in \mathbb{R}^n$$

$$\left(\frac{\partial y}{\partial x}\right)_n = \frac{\partial y}{\partial x_n}$$

Derivate vettoriali sui nodi

- Input vettore, output vettore

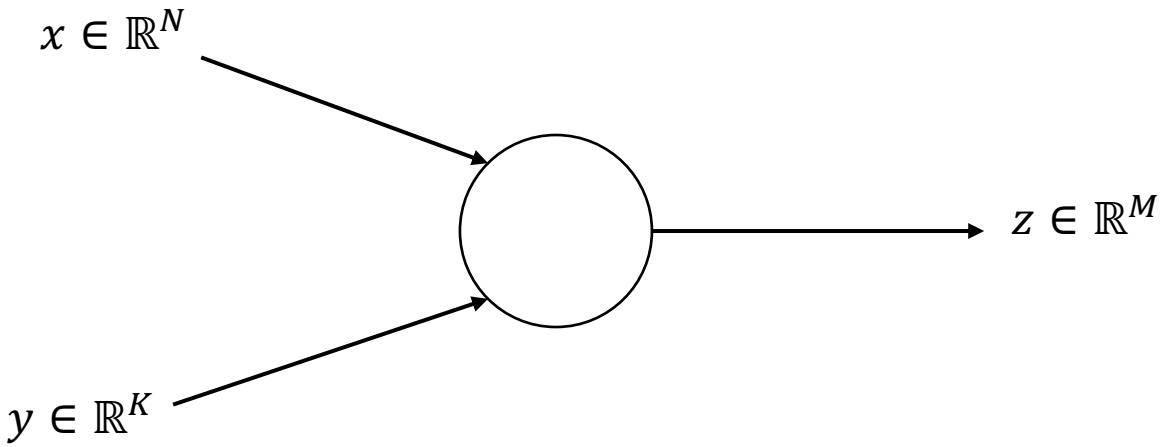
$$x \in \mathbb{R}^N, y \in \mathbb{R}^M$$

- Jacobiano

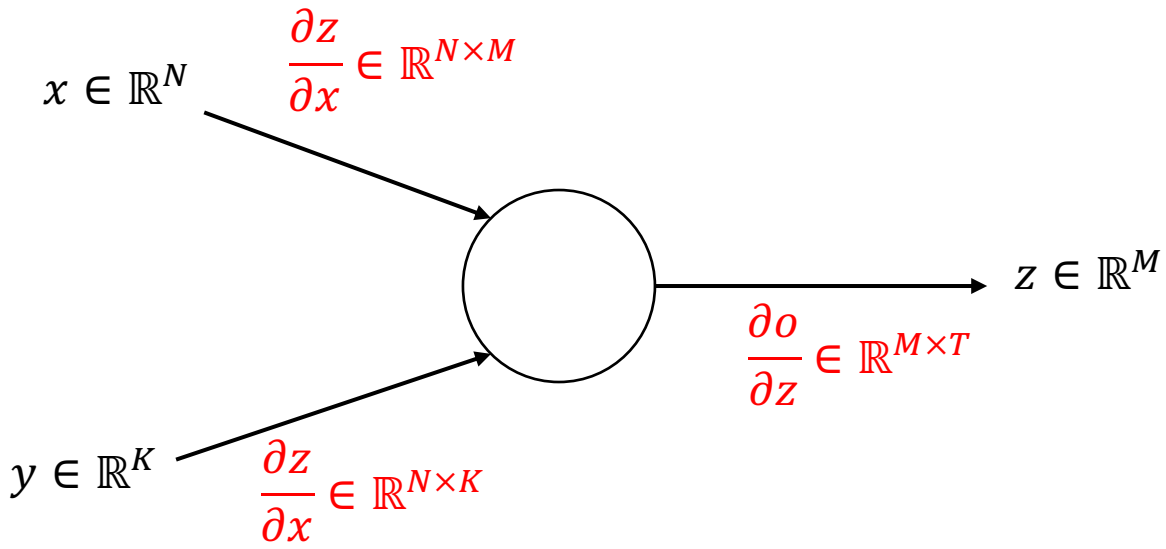
$$\frac{\partial y}{\partial x} \in \mathbb{R}^{N \times M}$$

$$\left(\frac{\partial y}{\partial x}\right)_{n,m} = \frac{\partial y_m}{\partial x_n}$$

Derivate vettoriali sui nodi

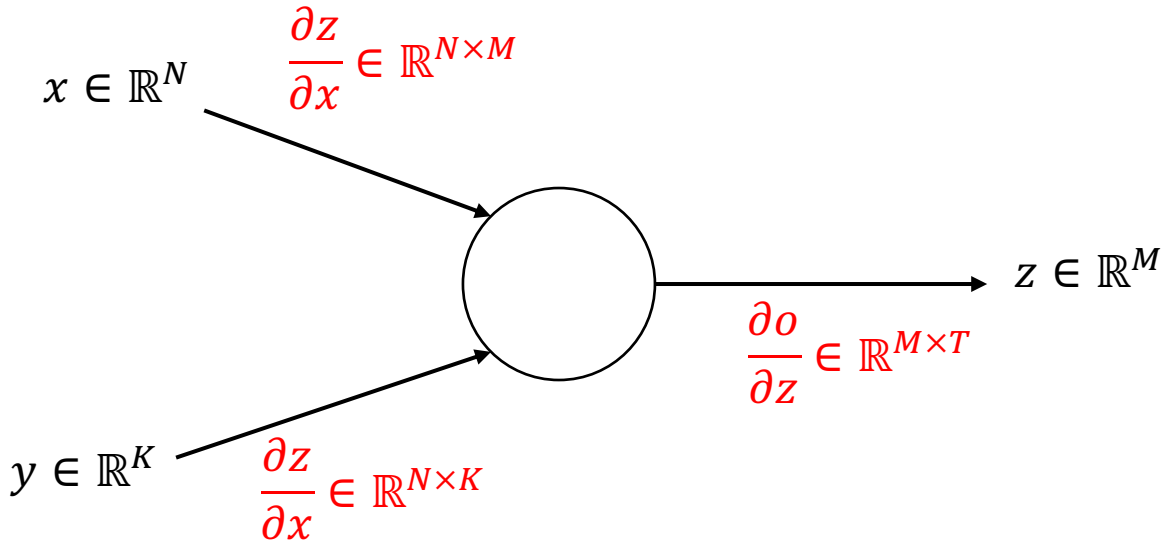


Derivate vettoriali sui nodi



Derivate vettoriali sui nodi

$$\frac{\partial o}{\partial x} = \frac{\partial z}{\partial x} \cdot \frac{\partial o}{\partial z} \in \mathbb{R}^{N \times T}$$



$$\frac{\partial o}{\partial y} = \frac{\partial z}{\partial y} \cdot \frac{\partial o}{\partial z} \in \mathbb{R}^{K \times T}$$

Esercizio

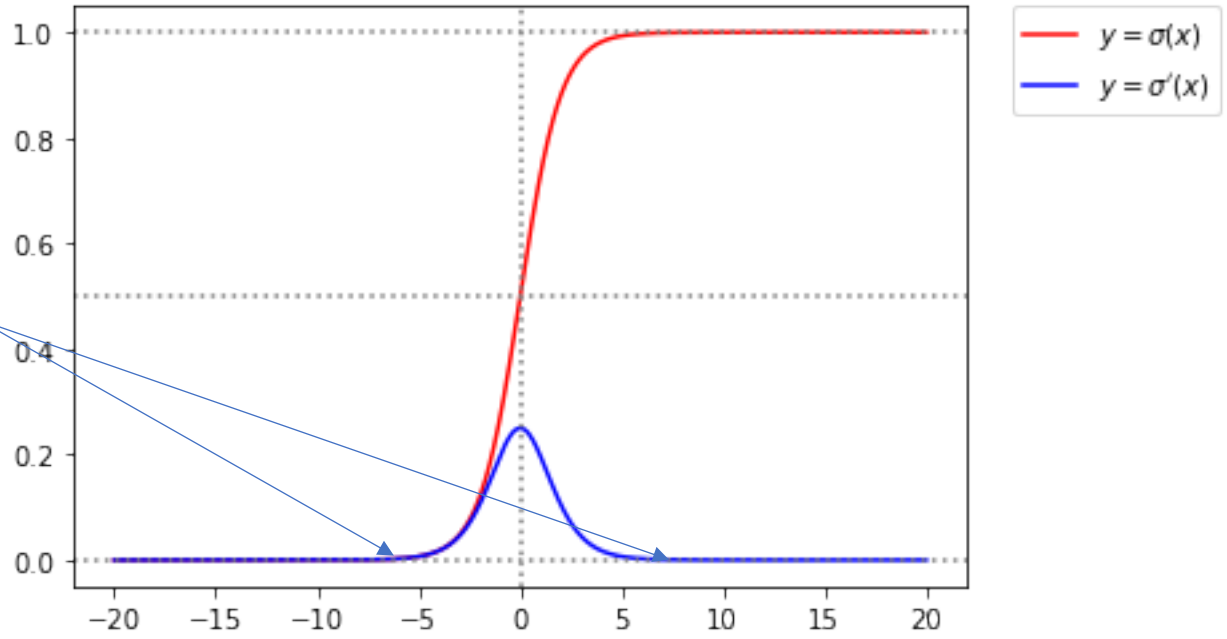
```
x = torch.ones((3,2),requires_grad = True)

y = torch.ones((2,2),requires_grad = True)*0.5

z = torch.ones((3,2),requires_grad = True)*0.25


z1 = torch.mm(x,y)

z2 = z1 + z

y = z2*torch.tensor([[1,0],[2,1],[0,1]]) + z

o = torch.sum(y)
```

- Disegnare il grafo
- Calcolare i gradienti

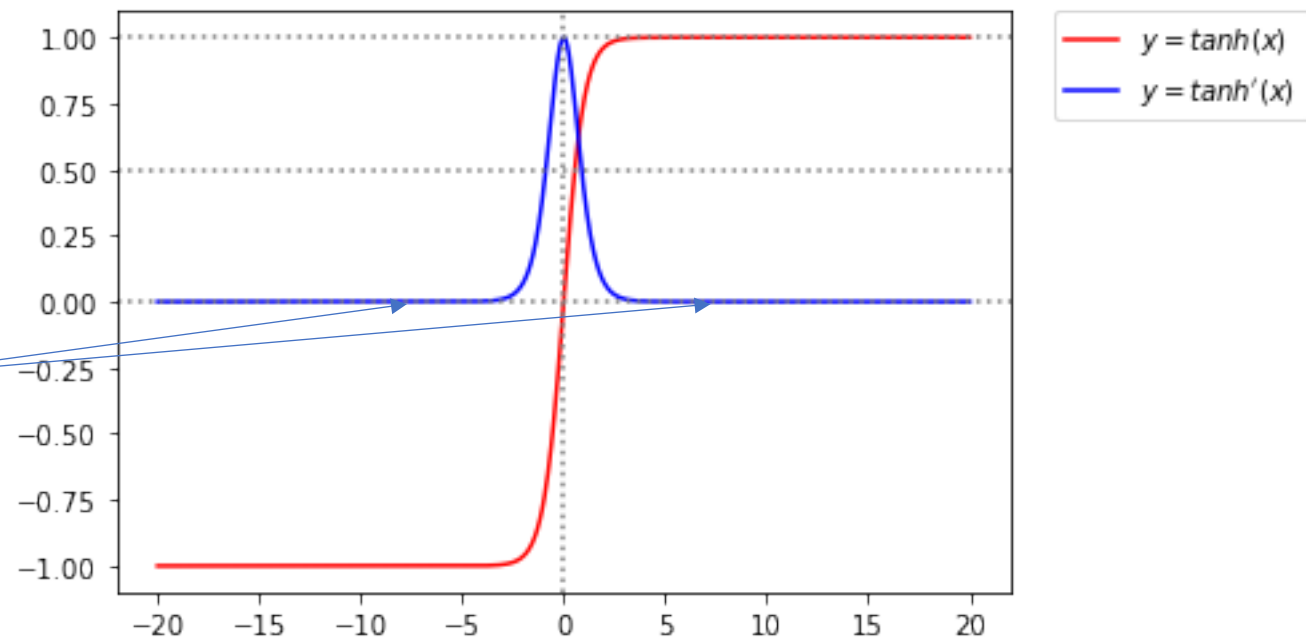
Funzioni di attivazione e gradienti

- Problema: saturazione

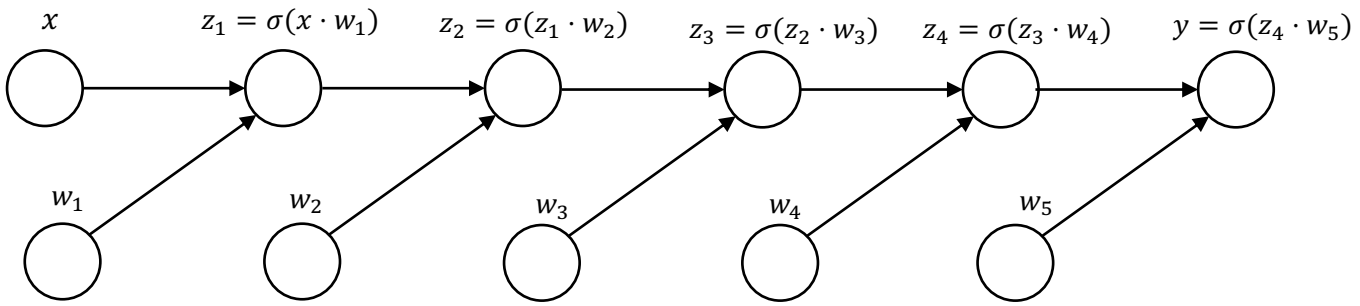
Gradiente nullo!



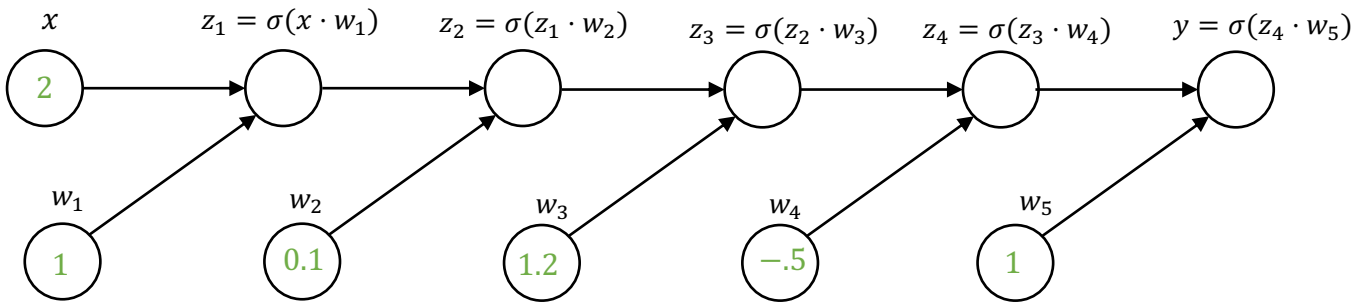
Gradiente nullo



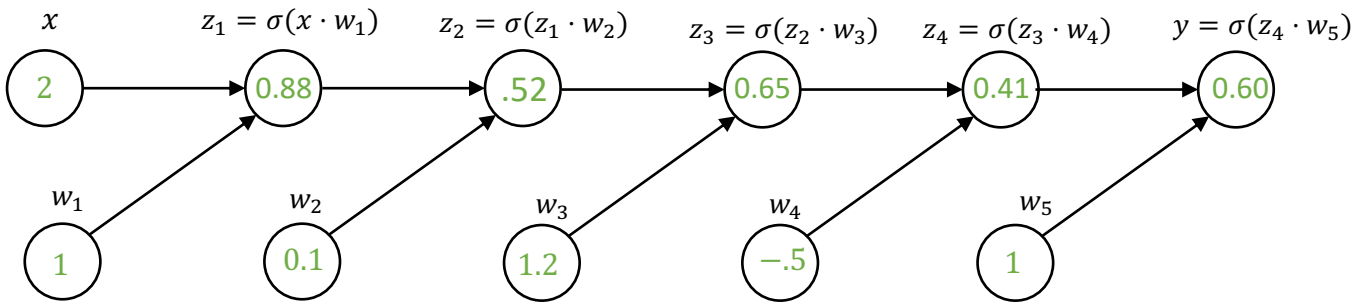
Gradiente evanescente



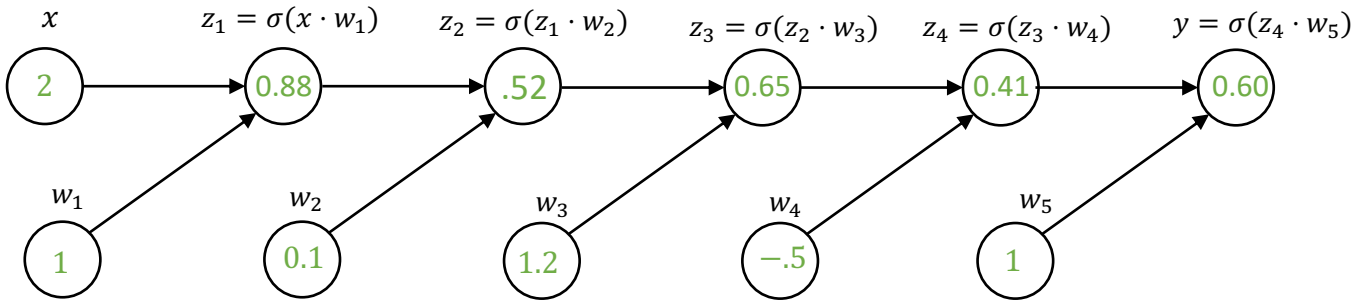
Gradiente evanescente



Gradiente evanescente

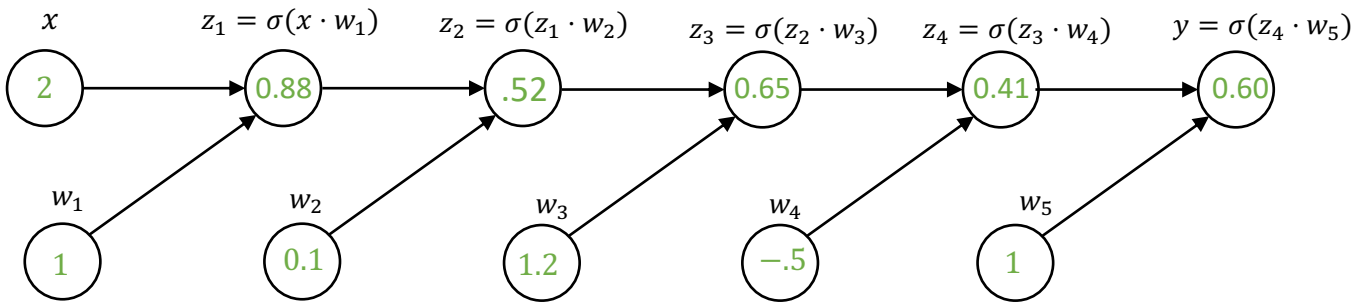


Gradiente evanescente



$$\frac{\partial y}{\partial w_1} = ?$$

Gradiente evanescente



$$\frac{\partial y}{\partial w_1} = \sigma'(z_4)w_5\sigma'(z_3)w_4\sigma'(z_2)w_3\sigma'(z_1)w_2\sigma'(x \cdot w_1)x$$

Vanishing Gradient

- Forward pass

$$\mathbf{a}^{(h+1)} = \mathbf{W}^{(h)}\mathbf{z}^{(h)}$$
$$\mathbf{z}^{(h+1)} = \sigma\left(\mathbf{a}^{(h+1)}\right)$$
$$\mathbf{z}^{(0)} = \mathbf{x}$$

- Backward pass

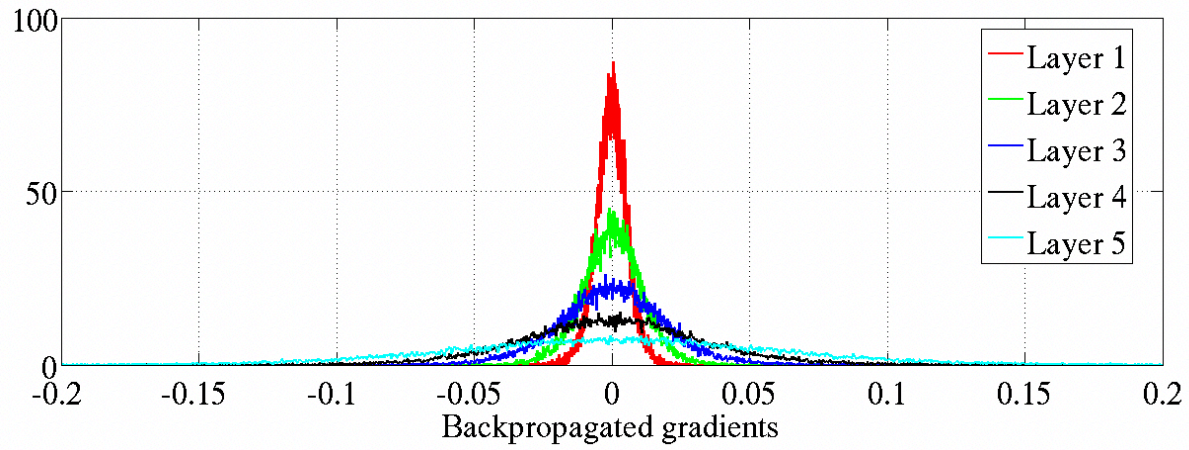
$$\frac{\partial \ell}{\partial \mathbf{z}^{(h)}} = \left(\mathbf{W}^{(h)}\right)^T \frac{\partial \ell}{\partial \mathbf{a}^{(h+1)}}$$
$$\frac{\partial \ell}{\partial \mathbf{a}^{(h)}} = \frac{\partial \ell}{\partial \mathbf{z}^{(h)}} \odot \sigma'(\mathbf{a}^{(h)})$$

Vanishing gradient

- Conseguenza:

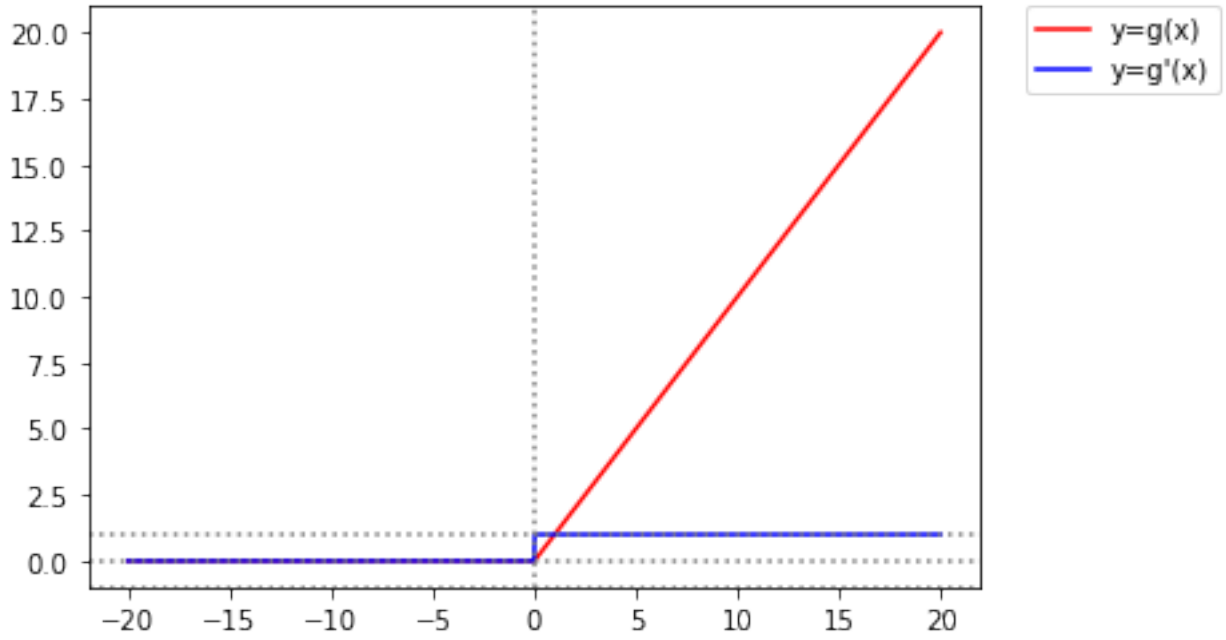
$$\frac{\partial \ell}{\partial \mathbf{z}^{(h)}} = \left(\mathbf{W}^{(h)}\right)^T \left(\sigma'(\mathbf{a}^{(h)}) \odot \frac{\partial \ell}{\partial \mathbf{z}^{(h+1)}}\right)$$

  - Il gradiente «svanisce» esponenzialmente con la profondità della rete se i pesi sono ill-conditioned o le attivazioni sono nel dominio di saturazione di σ.



Hidden layers

- ReLU



l funzione di loss

… defining the network goal

l funzione di loss

- g è un operatore algebrico non lineare
- L'operatore è parametrico rispetto ai pesi:
  - La matrice W e il bias b
- La fase di learning aspira a trovare i migliori valori di W e b

l funzione di loss

- Problema di ottimizzazione
  - Qual’è l’output desiderato?
  - Quando è differente dall’output prodotto?

l funzione di loss

- La loss misura la discrepanza tra l'output predetto e quello desiderato

- La funzione obiettivo:

$$\underset{W,B}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} loss[y_i, g(x_i|W, B)]$$

l funzione di loss

- Se l'output è una classe:
  - Binary Cross Entropy (BCE) – $y_i \in \{0; 1\}$, $g(\vec{x}_i|W, B) \in [0; 1]$

$$\mathrm{BCE} = -\frac{1}{n}\sum_{i=1}^{n} y_i \ln g(x_i|W, B) - (1 - y_i) \ln[1 - g(x_i|W, B)]$$

  - Categorical Cross Entropy (CCE) – K classes, $y_{i,k} \in \{0; 1\}$, $g(x_i|W, B) \in [0; 1]$

$$\mathrm{CCE} = -\frac{1}{n}\sum_{i=1}^{n}\sum_{k=1}^{K} y_{i,k} \ln g(x_i|W, B)_k$$

  - Hinge – $y_i \in \{-1; 1\}$

$$\mathrm{Hinge} = \frac{1}{n}\sum_{i}^{n} \max\{0; 1 - y_i \cdot g(x_i|W, B)\}$$

l funzione di loss

- BCE:
  - Classi binarie
  - Pesa gli errori allo stesso modo
- CCE:
  - Classi multiple
  - Pesa gli errori allo stesso modo
- Hinge
  - Classi binarie
  - Pesa gli errori allo stesso modo
  - Non differenziabile
  - Penalizza le predizioni con confidenza bassa
    - Vicina a 0 quando i segni coincidono e la predizione è vicina a 1

o L’ottimizzatore

… finding optimal solutions

o l'ottimizzatore

- Problema di ottimizzazione

$$\underset{W,B}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} loss[y_i, g(x_i|W, B)]$$

- Stochastic Gradient Descent
  - E sue varianti

o l'ottimizzatore

- Più controllo sugli update
  - Momentum update

$$W_{t+1}^{*} = m \cdot W_{t}^{*} - \eta \nabla l_i$$

$$l_{i+1} = l_{i+1} + W_{t+1}^{*}$$

  - Annealing
    - Learning rate adattivo
    - E.g. Exponential decay $\lambda_t = \lambda_0 \cdot e^{-\alpha t}$

o l’ottimizzatore

- Varianti di SGD
- Considerano
  - Raggiungibilità
  - Convergence speed
  - Overfitting

o l'ottimizzatore

- Varianti di SGD
  - AdaGrad

$$W_{t+1}^{*} = W_{t}^{*} - \frac{\eta}{\sqrt{\epsilon \cdot I + \mathrm{diag}\left(\nabla l_i \cdot \nabla l_i^{T}\right)}} \nabla l_i$$

  - I pesi con gradiente alto hanno un learning rate ridotto
  - Pesi con gradiente piccolo hanno un learning rate ampliato

o l'ottimizzatore

- Varianti di SGD
  - RMSprop

$$\zeta_{t+1} = \alpha \cdot \zeta_t + (1 - \alpha) \cdot (\nabla l_i)^2$$

$$W_{t+1}^* = W_t^* - \frac{\eta}{\sqrt{\epsilon \cdot I + \zeta_{t+1}}} \nabla l_i$$

  - Riduce la policy aggressive di AdaGrad sulla riduzione del learning rate

o l'ottimizzatore

- Varianti di SGD
  - Adam

$$\zeta_{t+1} = \alpha \cdot \zeta_t + (1-\alpha) \cdot (\nabla l_i)^2 \qquad \rightarrow \qquad \zeta^*_{t+1} = \frac{\zeta_{t+1}}{1-(\alpha)^{t+1}}$$

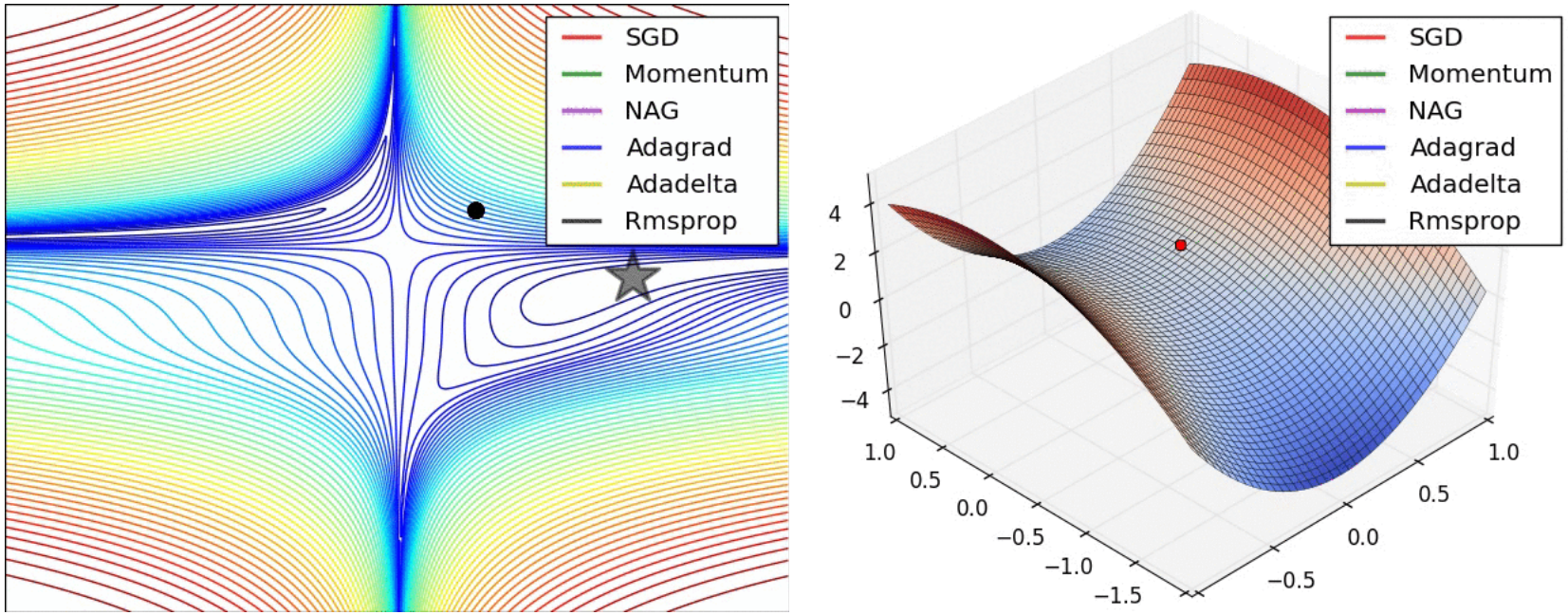
$$m_{t+1} = \beta \cdot m_t + (1-\beta) \cdot \nabla l_i \qquad \rightarrow \qquad m^*_{t+1} = \frac{m_{t+1}}{1-(\beta)^{t+1}}$$

$$W^*_{t+1} = W^*_t - \frac{\eta \cdot m_{t+1}}{\sqrt{\epsilon \cdot I + \zeta_{t+1}}} \nabla l_i$$

  - RMSprop con smoothing

o l'ottimizzatore

- Confronto



(Source: Stanford class CS231n, MIT License, Image credit: Alec Radford)

i l’inizializzazione

… well begun is half done

i l’inizializzazione

- I pesi necessitano un valore iniziale
- L’inizializzazione ha un effetto significativo sul risultato finale

i l’inizializzazione

- Zero initialization
  - Bad
    - Tutti I nodi hanno lo stesso gradiente
    - Non c’è diversificazione
      - Simmetria

i l’inizializzazione

- Random initialization