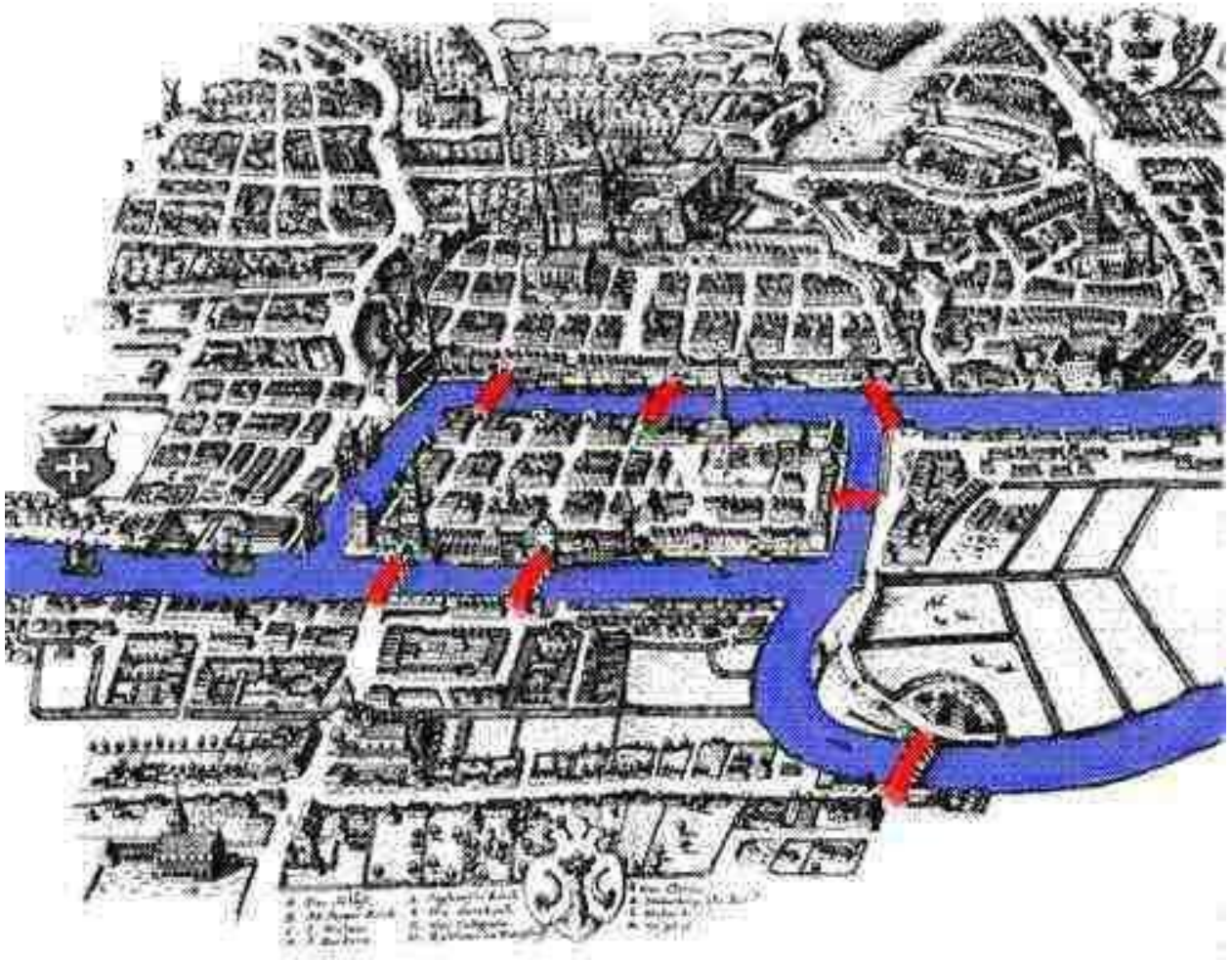
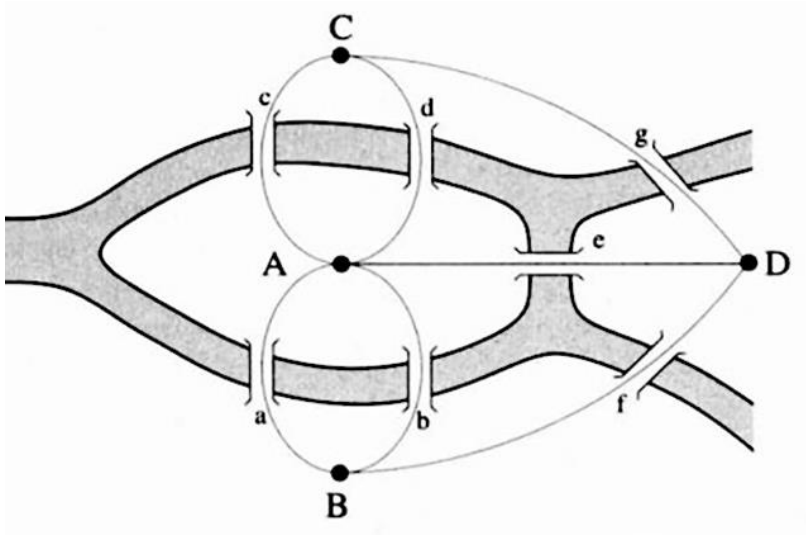
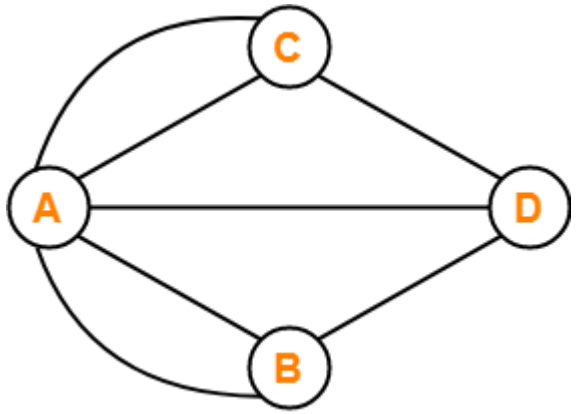
Computational graphs

Table of contents

History of graph theory. The Königsberg bridge problem

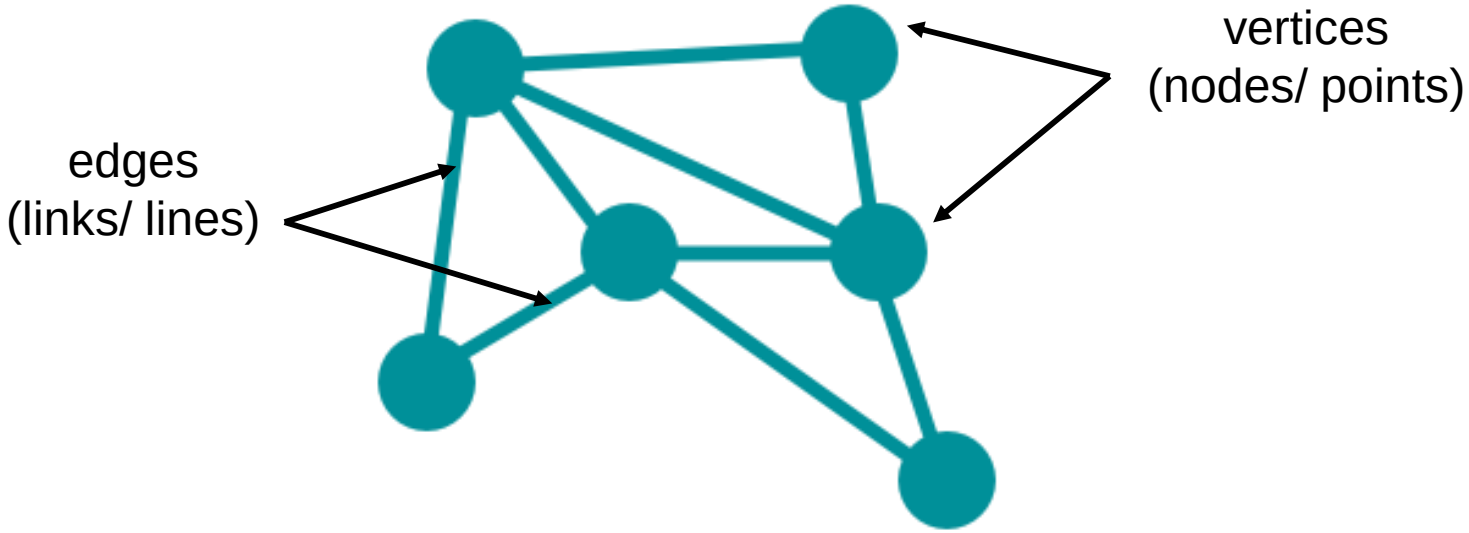






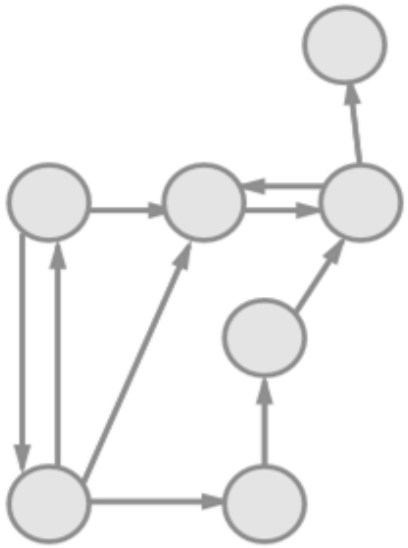
Graph theory. Definition

- a graph is a mathematical structure used to model **pairwise relations between objects**;
- a graph is made up of **vertices** , which are connected by **edges**.

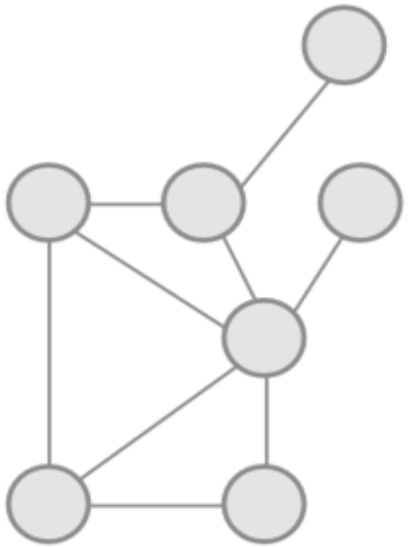


Graph theory. Classification

- graphs can have two types of edges:
    - edges that have a direction or flow (directed edges)
    - edges that have no direction (undirected edges)



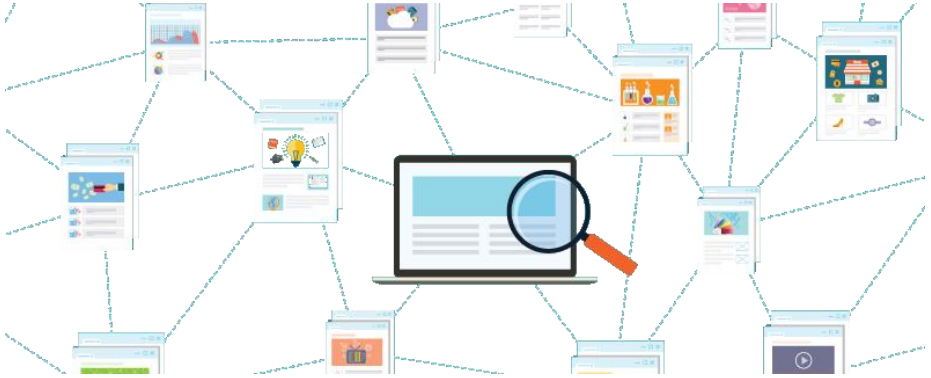
Directed graph



Undirected graph

Applications of graph theory

- in **computer science**, graphs are used to represent networks of communication, data organization, computational devices, the flow of computation, etc.;
- in **molecular biology**, scientists extrapolate prediction models for tracking the spread of diseases or breeding patterns;
- in **social media network analysis,** graph theory can be used as a way to describe whether people know each other, to predict how certain people can influence the behavior of others, etc.

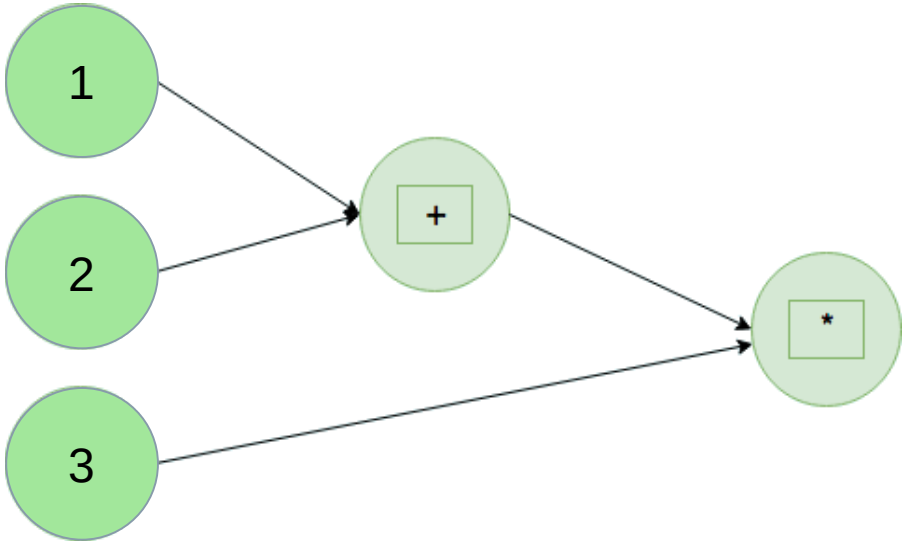


Computational graphs

- a computational graph is a way to represent **a math function in the language of graph theory**;
- a computational graph is a **directed graph** where the nodes correspond to operations or variables;
- the values that are fed into the nodes and come out of the nodes are called **tensors** (scalars, vectors and matrices);

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

Computational graph



Mathematical notation

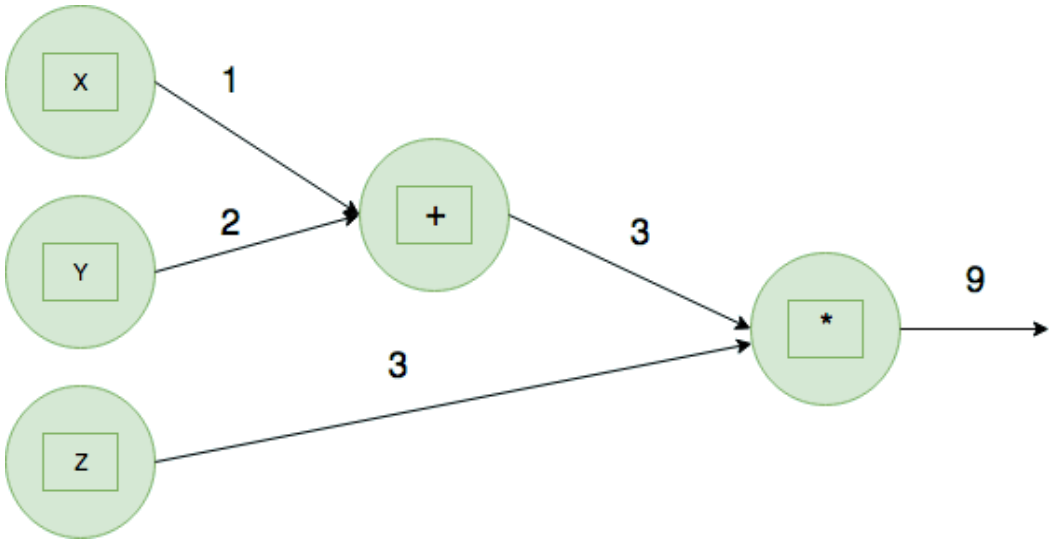
f(x, y, z) = h(g(x, y) , z)
g(i, j) = i + j
h(p, q) = p*q

Computational graphs

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

f(1, 2, 3) = ?

Computational graph



Mathematical notation

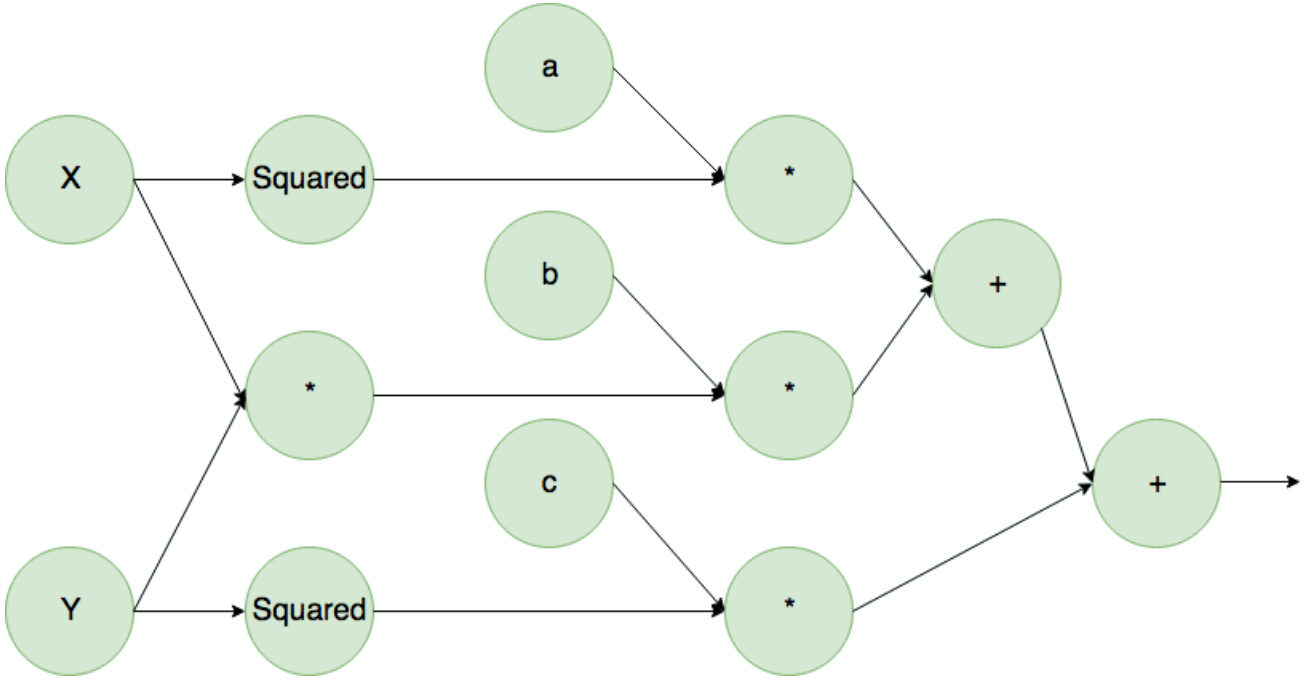
f(x, y, z) = h(g(x, y) , z)
g(i, j) = i + j
h(p, q) = p*q

f(1, 2, 3) = h(g(1, 2), 3)
g(1, 2) = 1 + 2 = 3
f(1, 2, 3) = h(3, 3)
h(3, 3) = 3*3 = 9
f(1, 2, 3) = 9

Neural networks as computational graphs

- every neural network represents a single mathematical function;
- when defining the architecture of a neural network it must be laid out the series of sub-functions and specified how they should be composed;
- training a neural network involves experimenting with the parameters of these sub-functions;

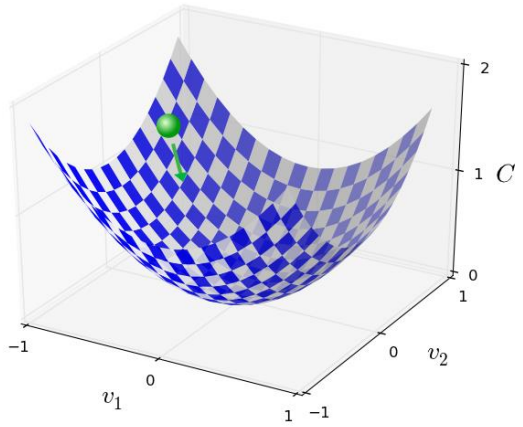
$$f(x, y) = ax^2 + bxy + cy^2$$

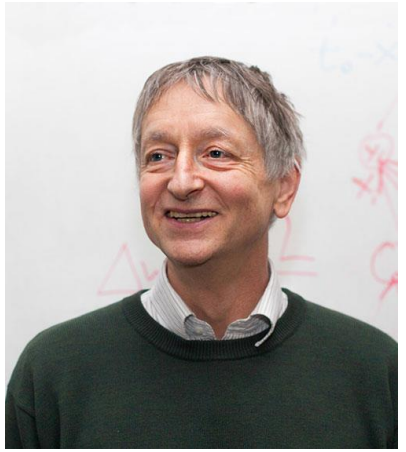


Computational graphs and backpropagation

- the goal in training a neural network is to find weights and biases which minimize a cost function **C(w,b)**;
- **Methods:** using calculus to find the minimum analytically;

  using gradient descent and backpropagation;
- backpropagation algorithm was originally introduced in the 1970s, but its importance wasn't fully appreciated until a famous 1968 paper by David Rumelhart, Geoffrey Hinton, and Ronald Williams **(“Learning representations by back-propagating errors”)**;
- the algorithm is called backpropagation because it computes the error vectors backward, starting from the final layer;





Geoffrey Hinton

Computational graphs and backpropagation

Why is backpropagation a fast algorithm?

- if we consider the cost as a function of the weights **C=C(w)**, we can compute the derivative of the cost function with respect to some particular weight with the following formula:

$$\frac{\partial C}{\partial w_j} \approx \frac{C(w + \epsilon e_j) - C(w)}{\epsilon}$$

  where **ϵ** is a small positive number, and $\mathbf{e}_j$ is the unit vector in the j^{th} direction;

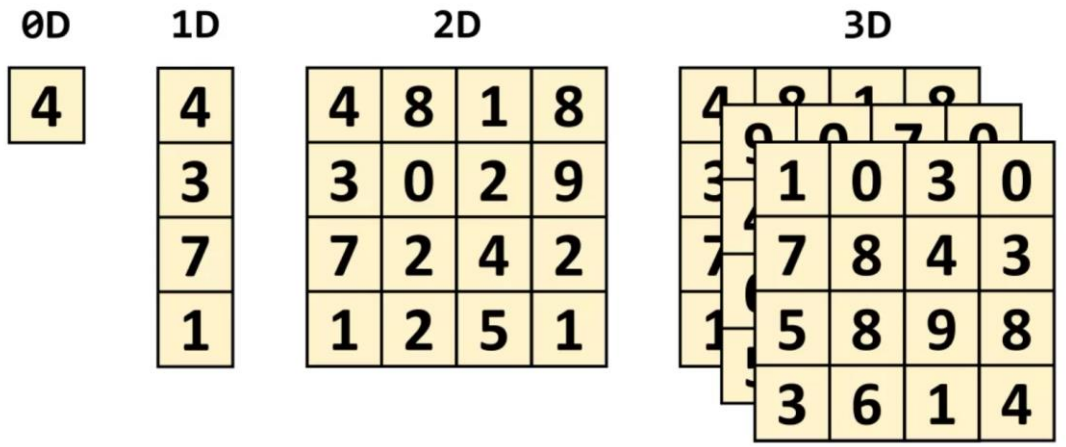
- this approach is simple conceptually and easy to implement, but it turns out to be extremely slow;

- backpropagation enables us to simultaneously compute all the partial derivatives using just one forward pass through the network, followed by one backward pass;

- so, even though backpropagation appears more complex, it's actually much faster.

Advantages of using tensors

- a **tensor** is a mathematical object that describes the relationship between other mathematical objects that are all linked together;
- they are commonly shown as an **array of numbers**;



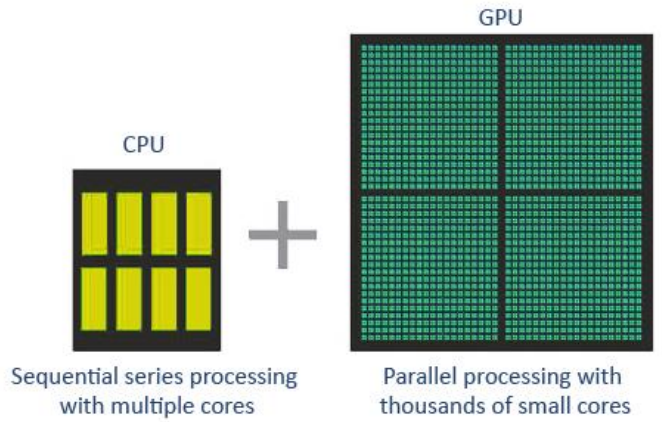
- backpropagation algorithm is based on common **linear algebraic operations**, like vector addition, multiplying a vector by a matrix, etc.

Advantages of using tensors

Matrices multiplication

$$\mathbf{C} = \mathbf{A} \times \mathbf{B}$$

$$\mathbf{C} = \begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ m & n & o & p \end{bmatrix} \times \begin{bmatrix} \alpha & \beta & \gamma & \delta \\ \varepsilon & \zeta & \eta & \theta \\ \kappa & \lambda & \mu & \nu \\ \pi & \tau & \psi & \omega \end{bmatrix}$$

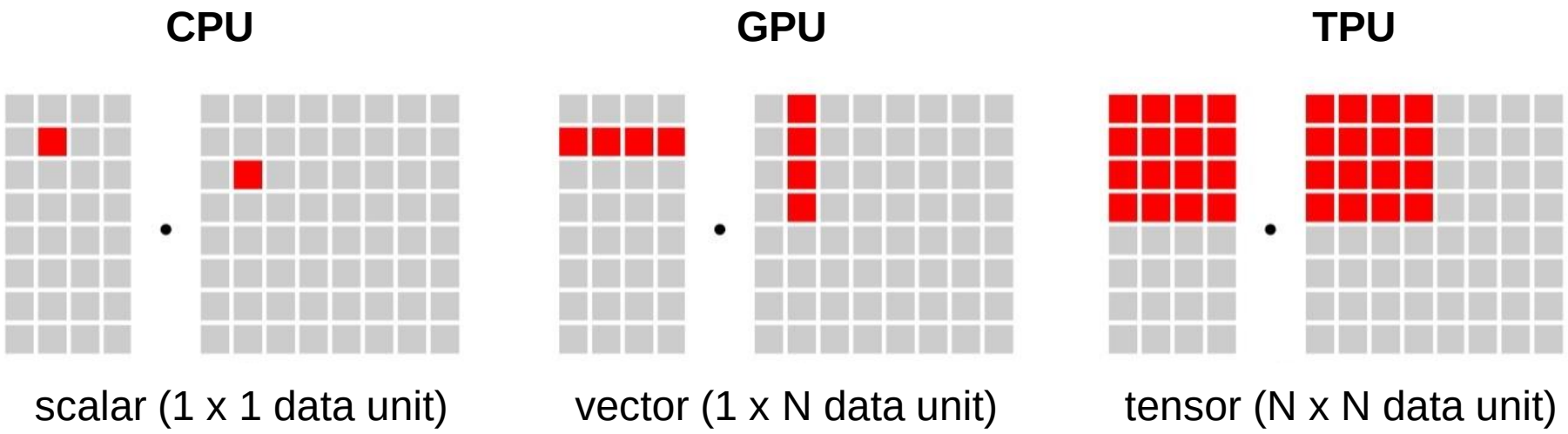


$$\mathbf{C} = \begin{bmatrix} (a \times \alpha) + (b \times \varepsilon) + (c \times \kappa) + (d \times \pi) & (a\beta) + (b\zeta) + (c\lambda) + (d\tau) & (a\gamma) + (b\eta) + (c\mu) + (d\psi) & (a\delta) + (b\theta) + (c\nu) + (d\omega) \\ (e\alpha) + (f\varepsilon) + (g\kappa) + (h\pi) & (e\beta) + (f\zeta) + (g\lambda) + (h\tau) & (e\gamma) + (f\eta) + (g\mu) + (h\psi) & (e\delta) + (f\theta) + (g\nu) + (h\omega) \\ (i\alpha) + (j\varepsilon) + (k\kappa) + (l\pi) & (i\beta) + (j\zeta) + (k\lambda) + (l\tau) & (i\gamma) + (j\eta) + (k\mu) + (l\psi) & (i\delta) + (j\theta) + (k\nu) + (l\omega) \\ (m\alpha) + (n\varepsilon) + (o\kappa) + (p\pi) & (m\beta) + (n\zeta) + (o\lambda) + (p\tau) & (m\gamma) + (n\eta) + (o\mu) + (p\psi) & (m\delta) + (n\theta) + (o\nu) + (p\omega) \end{bmatrix}$$

- operations with tensors can run on **graphics processing units** (GPUs) to speed up the computation process.

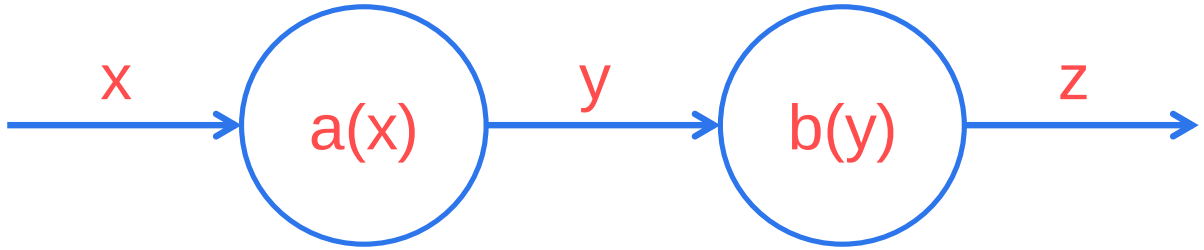
CPU vs. GPU vs. TPU

- the trade off between using a CPU vs a GPU consists in the fact that the CPU can do sequential and complex operations, whilst the GPU only performs basic operations but in parallel;
- Google announced their first **TPU** (tensor processing unit) in 2016, but these chips are so specialized they can't do anything other than matrix operations;
- TPUs' purpose is to accelerate deep learning tasks developed using TensorFlow;
- the image below summarizes the smallest unit in CPU, GPU and TPU:



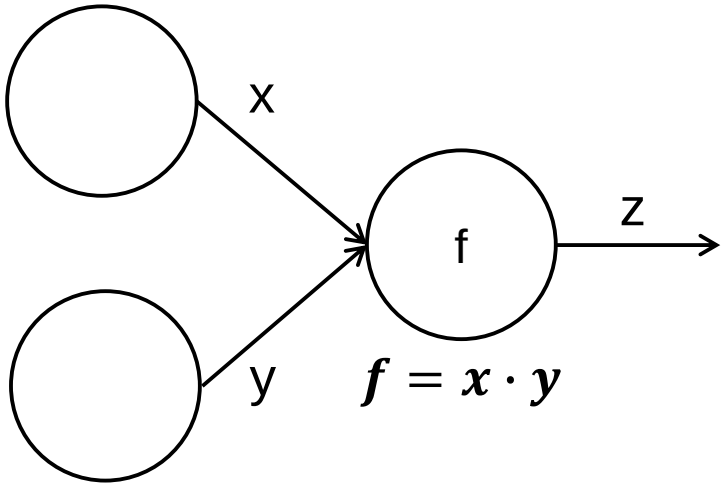
Theoretical example. Backpropagation

Chain rule:



$$\frac{dz}{dx} = \frac{dz}{dy} \cdot \frac{dy}{dx}$$

Computational graph:



Local gradients:

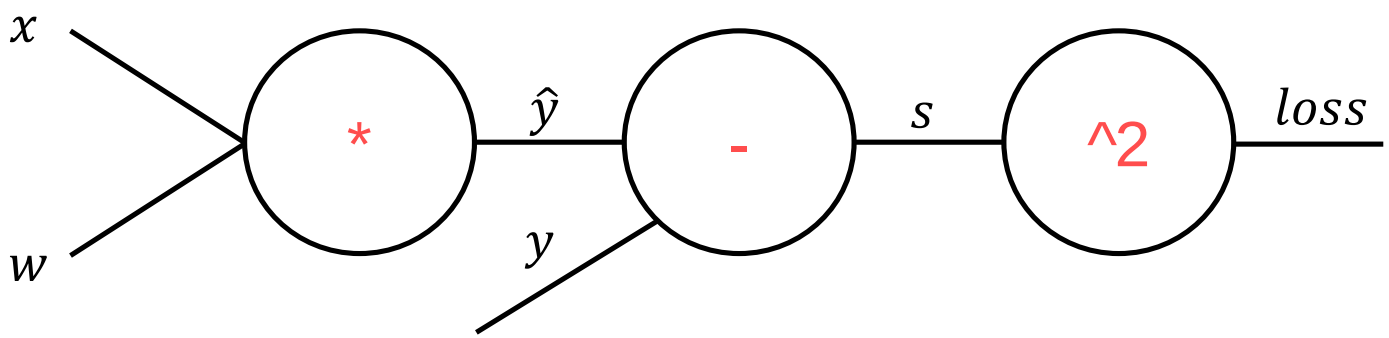
$$\frac{dz}{dx} = \frac{dxy}{dx} = y \qquad \frac{dz}{dy} = \frac{dxy}{dy} = x$$

$$\frac{dLoss}{dx} = \frac{dLoss}{dz} \cdot \frac{dz}{dx}$$

$$\frac{dLoss}{dy} = \frac{dLoss}{dz} \cdot \frac{dz}{dy}$$

Theoretical example. Backpropagation

1) **Forward pass:** compute Loss
2) **Compute local gradients**
3) **Backward pass:** compute dLoss/ dWeights using the chain rule



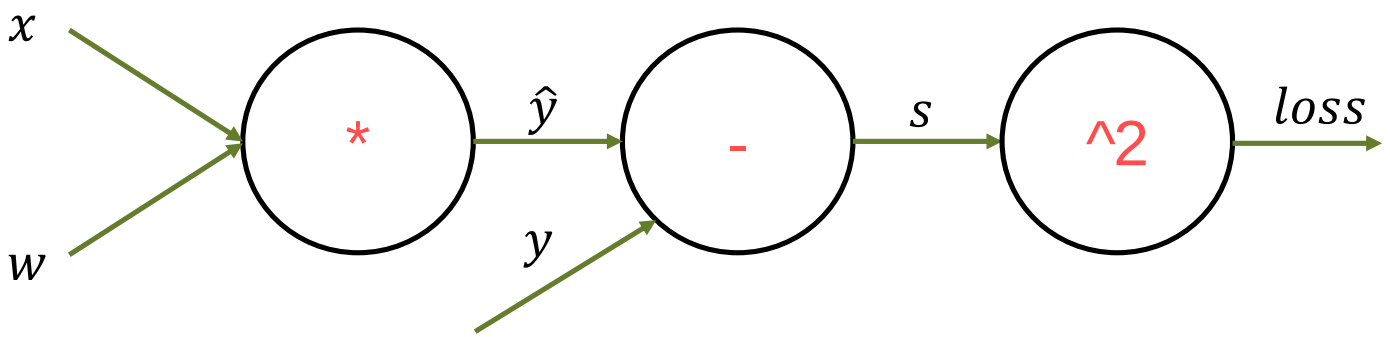
Linear regression: $\hat{\boldsymbol{y}} = \boldsymbol{w} \cdot \boldsymbol{x}$

$$\boldsymbol{loss} = (\hat{\boldsymbol{y}} - \boldsymbol{y})^{\boldsymbol{2}} = (\boldsymbol{wx} - \boldsymbol{y})^{\boldsymbol{2}}$$

$$\boldsymbol{Minimize\ loss} \rightarrow \frac{\boldsymbol{dloss}}{\boldsymbol{dw}} = ?$$

Theoretical example. Backpropagation

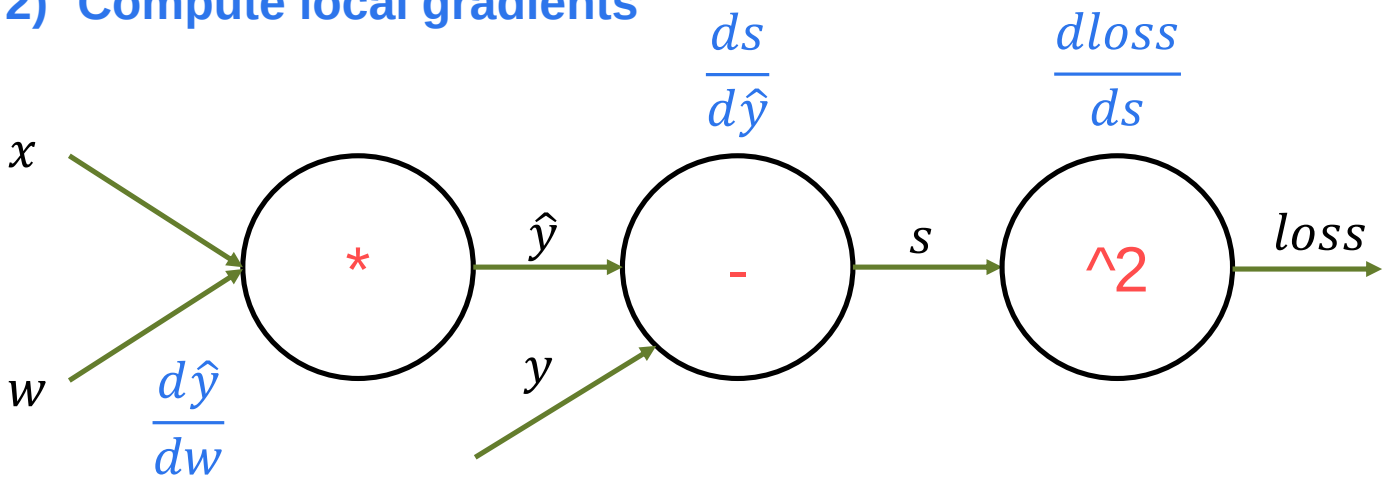
1) Forward pass: compute Loss



Linear regression: $\hat{\boldsymbol{y}} = \boldsymbol{w} \cdot \boldsymbol{x}$

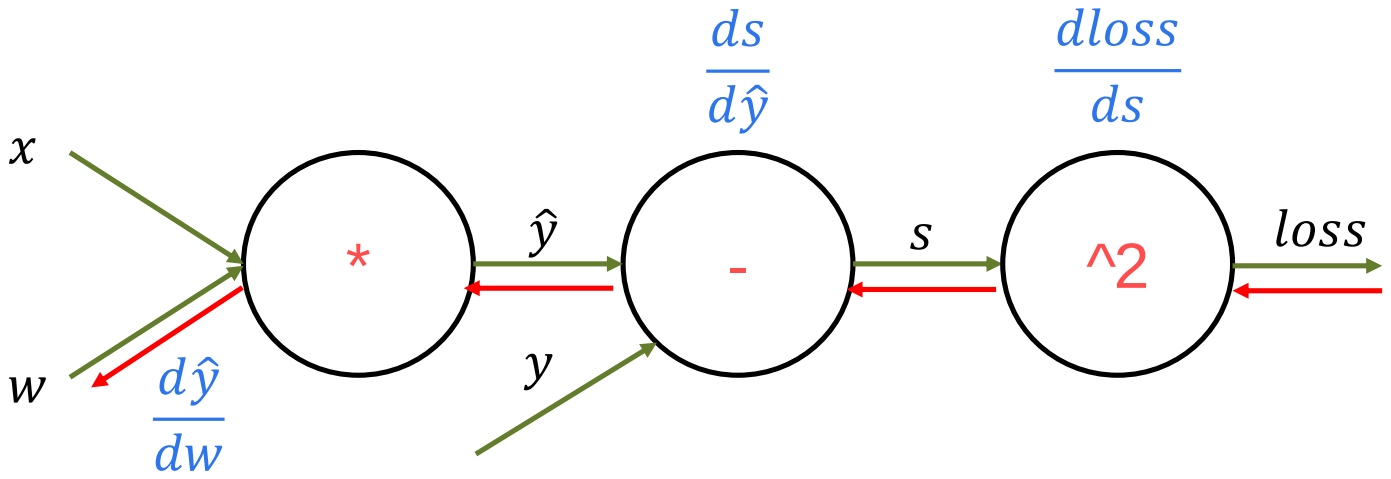
$$\boldsymbol{loss} = (\hat{\boldsymbol{y}} - \boldsymbol{y})^2 = (\boldsymbol{wx} - \boldsymbol{y})^2$$

2) Compute local gradients



Theoretical example. Backpropagation

3) Backward pass: compute dLoss/ dWeights using the chain rule

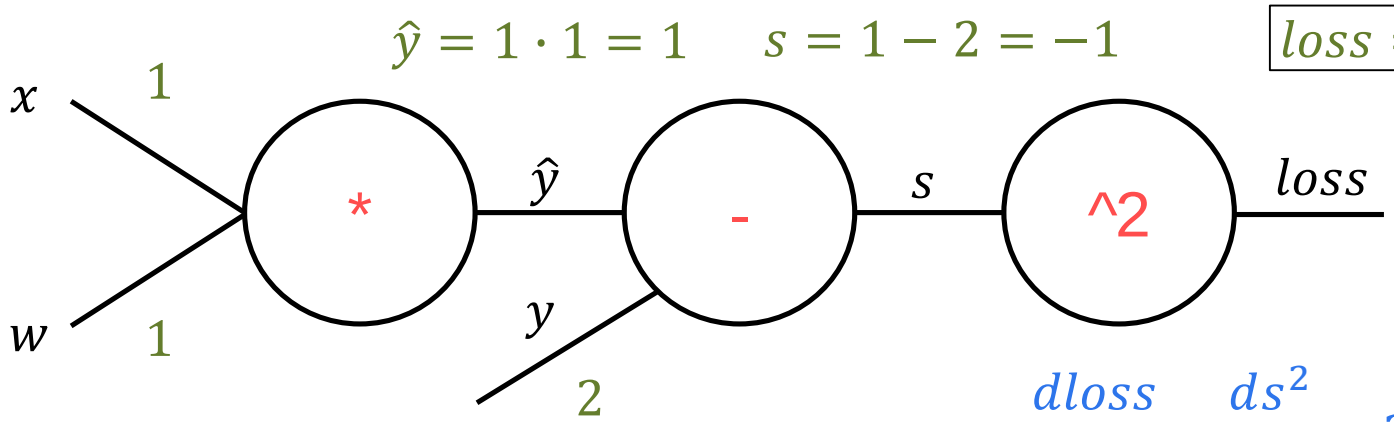


$$\frac{dloss}{dw} \longleftarrow \frac{dloss}{d\hat{y}} \longleftarrow \frac{dloss}{ds}$$

$$\frac{dloss}{dw} = \frac{dloss}{ds} \cdot \frac{ds}{d\hat{y}} \cdot \frac{d\hat{y}}{dw}$$

Numerical example. Backpropagation

- **x=1, y=2, w=1**



$$\frac{d\hat{y}}{dw} = \frac{dwx}{dw} = x$$

$$\frac{ds}{d\hat{y}} = \frac{d(\hat{y} - y)}{d\hat{y}} = 1$$

$$\frac{dloss}{ds} = \frac{ds^2}{ds} = 2s$$

$$\frac{dloss}{d\hat{y}} = \frac{dloss}{ds} \cdot \frac{ds}{d\hat{y}} = 2 \cdot s \cdot 1 = -2$$

$$\frac{dloss}{dw} = \frac{dloss}{d\hat{y}} \cdot \frac{d\hat{y}}{dw} = -2 \cdot x = -2$$

Conclusions

1. A computational graph is a way to represent a math function, using vertices which are connected by edges;

2. Every neural network represents a mathematical function, that can be represented as a series of sub-functions. The way these sub-functions should be composed could be specified through a computational graph;

3. Training a neural network involves experimenting with a huge number of parameters. A fast algorithm to determine the values of all these parameters is backpropagation.

4. Because backpropagation algorithm is based on common linear algebraic operations that make use of tensors, we can train neural networks on GPUs or even TPUs to speed up the computation process.

5. Computational graphs are compatible with backpropagation algorithm since each node knows how to compute its value and the value of its derivative w.r.t each argument (edge).