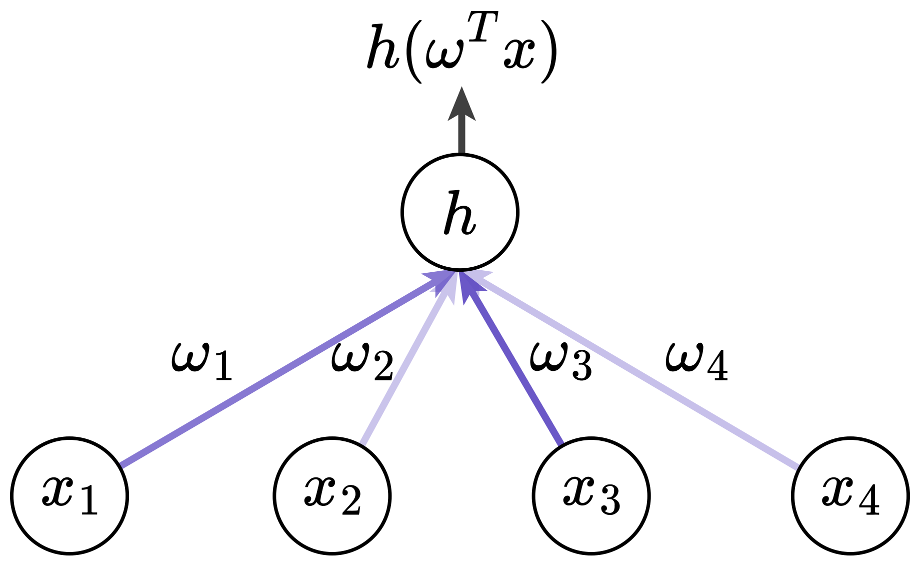
Gradient-based optimization

Università di Pisa

Exploiting gradient

The computational graph

We can generalize several gradient-based optimization to *computational graphs*, objects which compute a generic function $f(x)$. Functional form: $h(\omega^T x)$. Note: $\omega^T x$ is simply a linear combination of x.



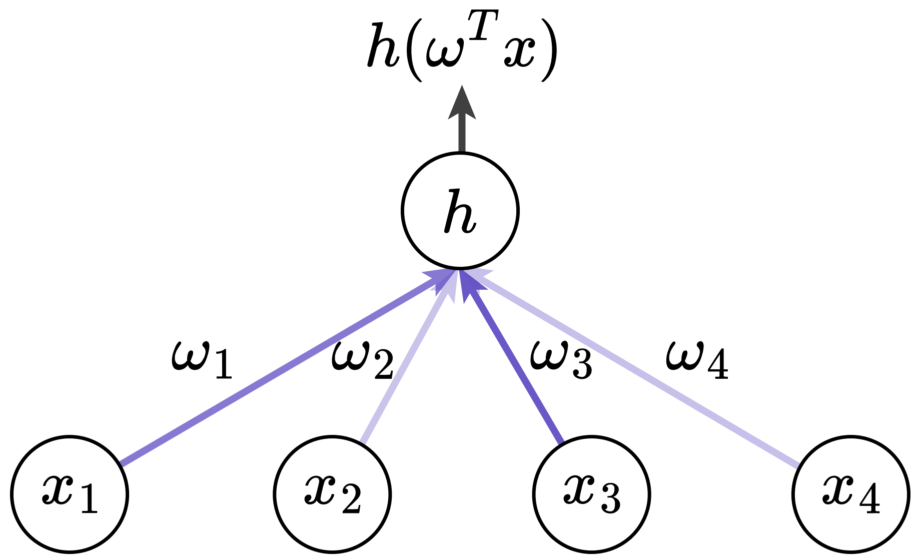
A computational graph: nodes indicate objects involved in the computation, and edges indicate their flow in the graph.

The forward pass: computing $f(x)$

Università di Pisa

To compute $f(x)$, we navigate the graph g bottom-up:

- Nodes are either
    - data: stores some data
    - functions: computes a function the sum of incoming edges
- Edges are weights in a weight space Θ, weighting the node traversing them

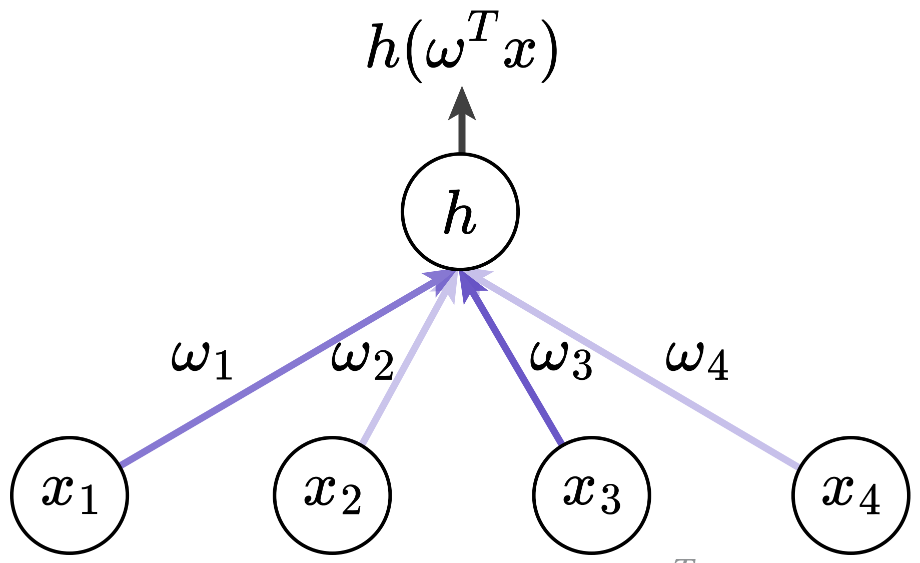


A forward pass in g: x_i is weighted by ω_i, then h is computed on their sum. Opacity of edges scaled with value of ω..

The forward pass: computing $f(x)$

In a computational graph, we have

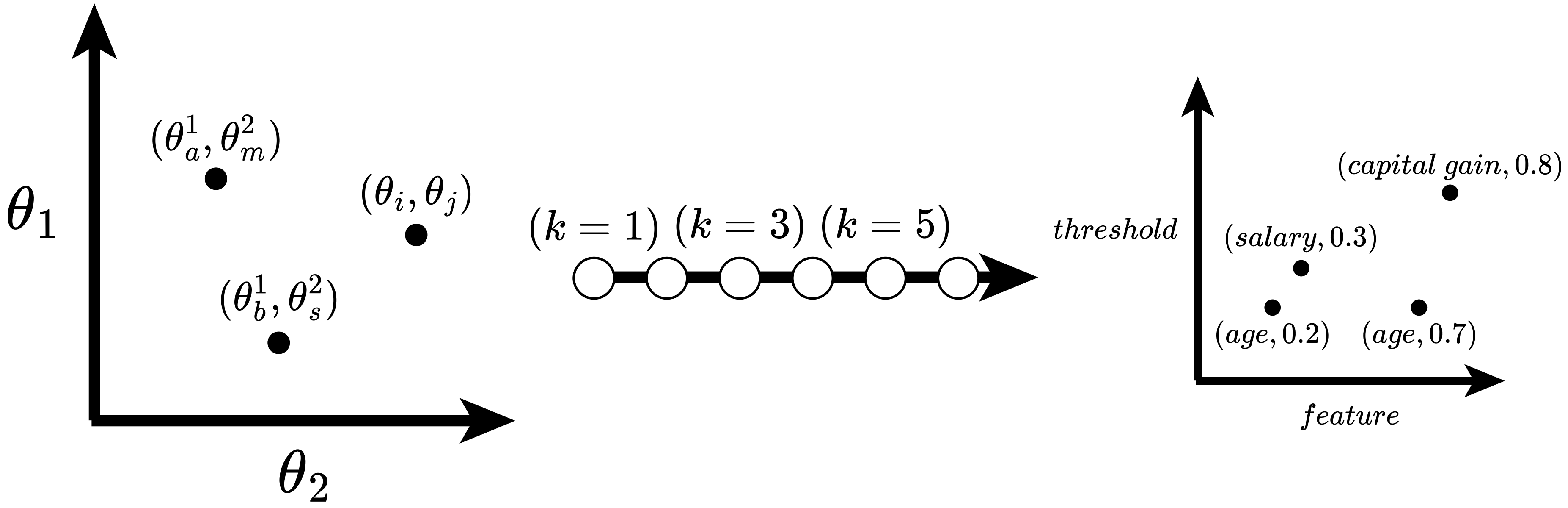
- Parameters θ given the weights on the nodes
- Structure given by the graph's architecture and functions



A computational graph g computing $h(w^T x)$. The structure is given by the operations $(h, \cdot)$, while the parameters ($\omega_1, \omega_2, \omega_3, \omega_4$).

Parameters and structure

The parameters-structure dichotomy is not unique to computational graphs, and we can find this also on other models. Note that pure algorithms with no learning involved, e.g., Naive Bayes, do not have this distinction, and everything is structure.

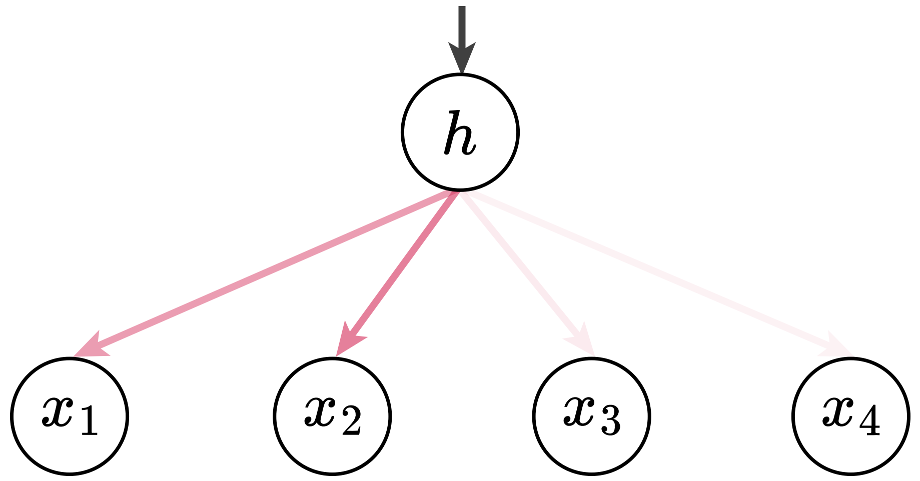


A generic parameter space Θ (left), one for k-NN (center), and a subset of one for Decision Trees (right).

The backward pass: improving θ

Given a loss L computed with a differentiable loss function l, as we have done for Gradient Boosting Machines, we can find directions in the model space where the L decreases.

Note: in computational graphs, we optimize over the parameter space, i.e., over Θ.



A backward pass in g: change parameters indicated by the gradient indicates the direction towards which move parameters to minimize l.

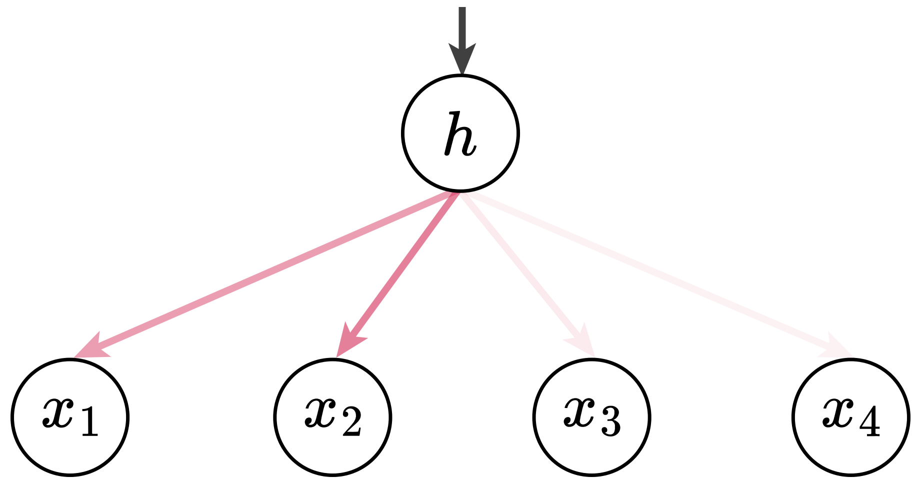
Scaling up: layering graphs

Università di Pisa

The backward pass is made possible by the differentiability of the loss... but this can be applied also to the functions within g: if a function h_i within g is differentiable, then I can **recursively compute gradients** on it!

Chain rule of calculus
The derivative of a composite function $f = f_1 \circ f_2$ is

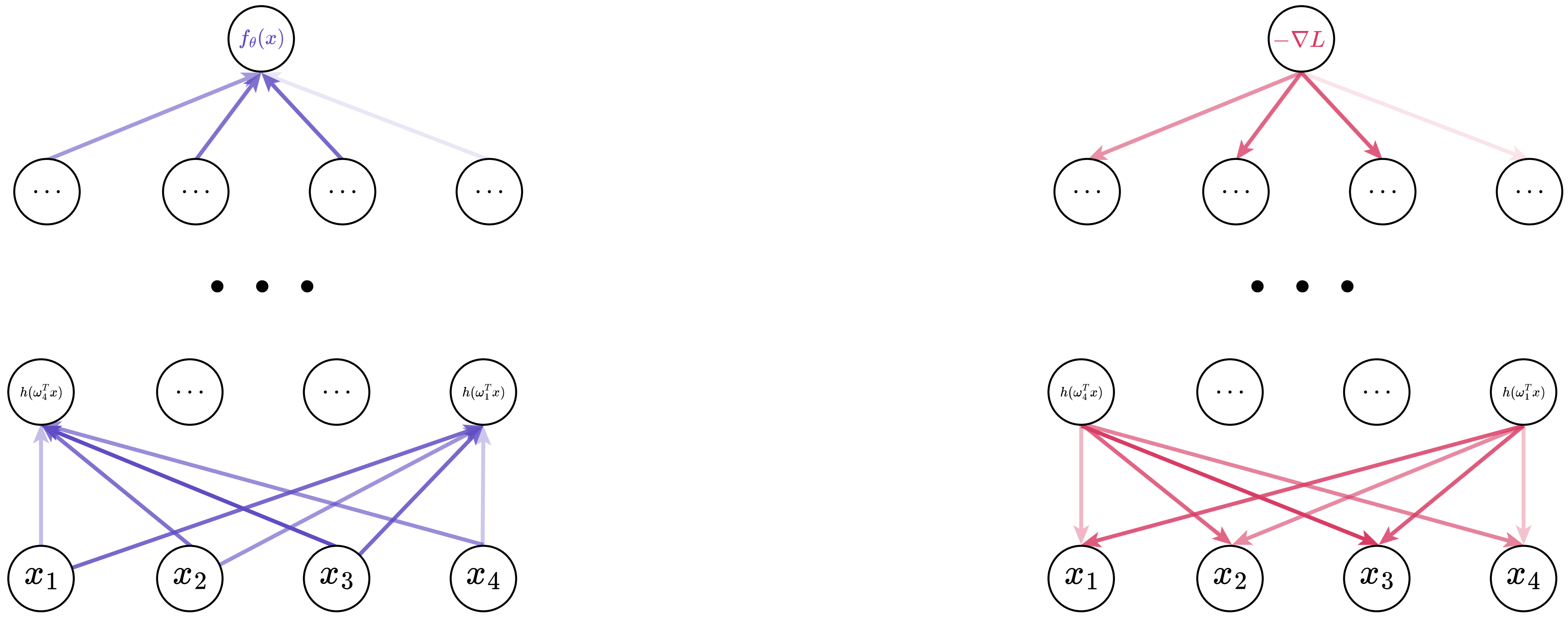
$$\frac{\partial f}{\partial x} = \frac{\partial f_1}{\partial f_2}\frac{\partial f_2}{\partial x}$$



A backward pass in g: change parameters indicated by the gradient indicates the direction towards which move parameters to minimize l.

Scaling up: layering graphs

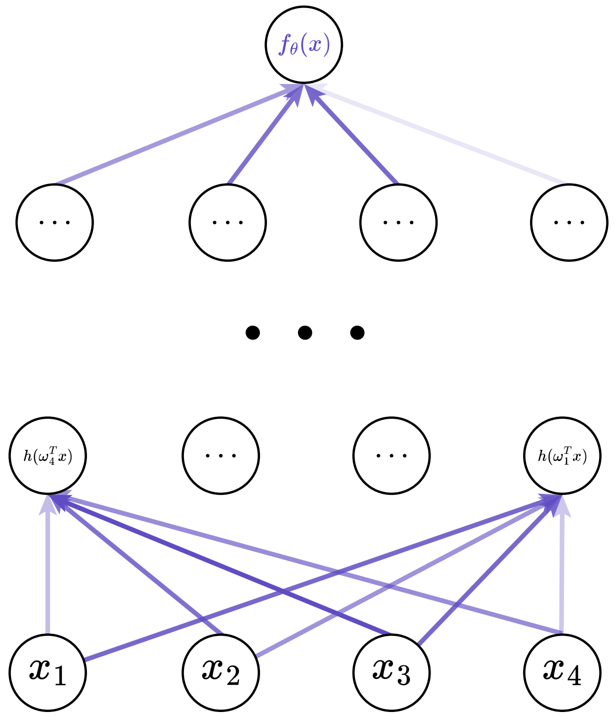
Applying the chain rule recursively, and going *down* the computational graph, we can compute optimization directions for all parameters! The algorithm chaining back the loss gradient is called *backpropagation*.



A multi-layered computational graph, and its forward (left) and backward (right) pass. The chain rule enables propagation of updates back and over the entire network.

Neural networks

Neural networks implement computational graphs.



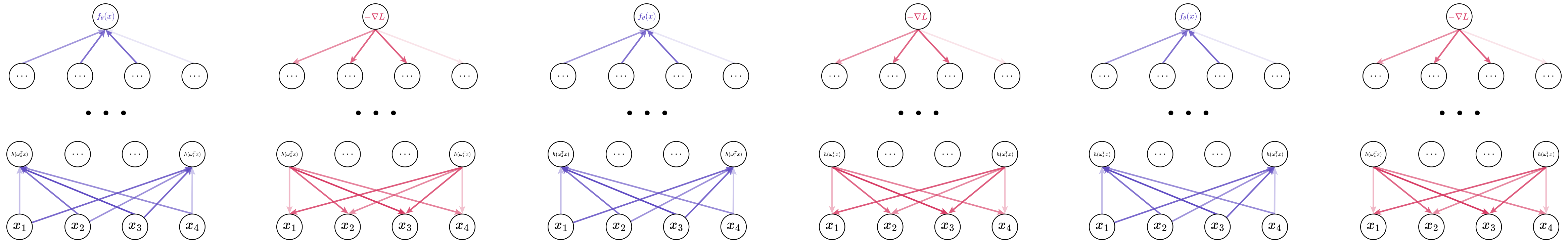
A visualization of a neural network.

Some nomenclature:

- Layer: set of all adjacent nodes
- Block: collection of consecutive layers
- Activation function: functions found in nodes
- Hidden layer: a layer, except the first or last one
- Output layer/nodes: layer/nodes yielding the computed $f_\theta(x)$

Neural networks: the training loop

Fitting a neural network consists in, at a very high level, repeatedly computing forward and backward passes, each pair improving on the current loss.



Consecutive foreward and backward passes on a network, each backward pass leverages gradients of the loss to find local directions of loss minimization, which are then used to fit the model.

Learning in neural networks

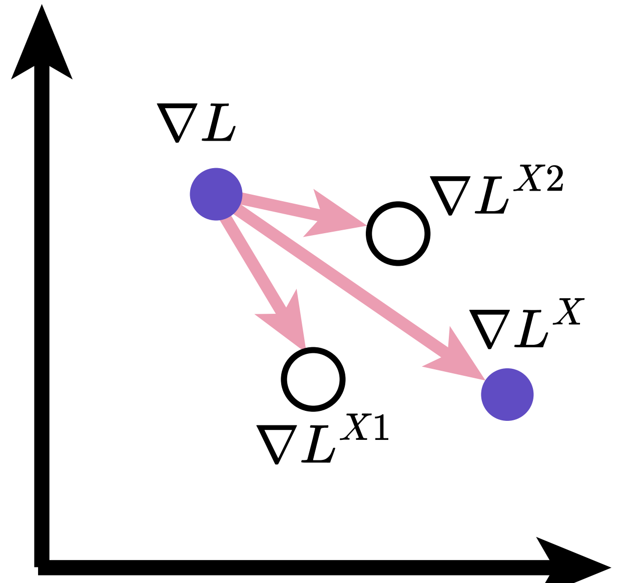
Stochastic learning

Università di Pisa

Backpropagation takes care of defining optimal directions for the parameters, but how do we estimate and leverage these directions?

Estimating directions: Through stochastic gradient, by computing gradient on a *batch* of data, rather than the whole dataset.

Provides a good approximation and a much faster computation.

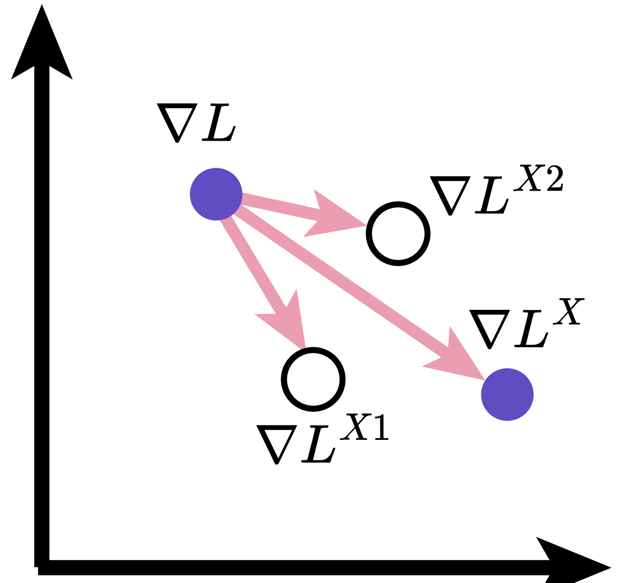


The gradient ∇L^X estimated on the some data X, and gradients ∇L^{X1}, ∇L^{X2} estimated on subsets of $X1$, $X2$ of X.

Stochastic learning

Backpropagation takes care of defining optimal directions for the parameters, but how do we estimate and leverage these directions?

Batches can be aggregated, the directions they yield combined: a training-specific approach to combat overfit.

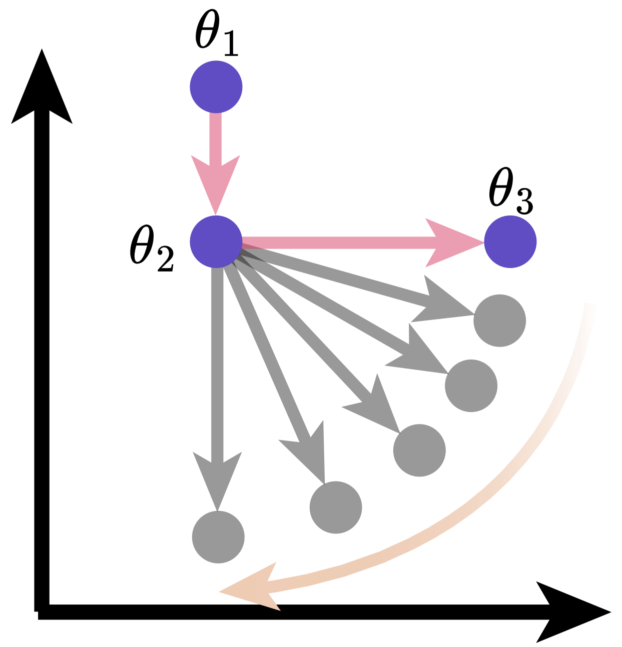


The gradient ∇L^X estimated on the some data X, and gradients $\nabla L^{X1}, \nabla L^{X2}$ estimated on subsets of $X1, X2$ of X.

Momentum

Backpropagation takes care of defining optimal directions for the parameters, but how do we estimate and leverage these directions?

Estimating weight update: Update directions are weighted by a possibly decaying learning rate η, and possibly weighted by past updates (Adam, RMSProp, Nesterov momentum).



Momentum controls the weight of the parameter updates: the lower, the lesser the impact of the direction, and vice versa.

Regularization

Regularization of neural networks can take many forms.

Weight regularization

The loss includes a term $|| \Omega ||_2$ to penalize large weights in the network, as higher weights tend to increase the network capacity.

Dropout

Random deletion of network connections, aims to create networks less reliant on a small number of neurons.

Early stopping

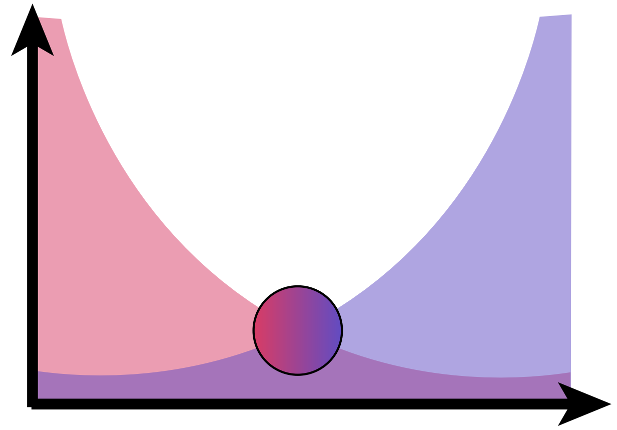
Università di Pisa

As fitting goes on, we adapt the capacity of the network. How do we prevent the model to overfit? Early stopping!

We keep two rolling statistics:

1. The gap between train and validation error
2. Train error

We stop when 1. grows larger (to avoid overfit), or 2. does not lower (to avoid unnecessary training).



Model error on training (blue) and validation (red) data: as iterations go by, the model lowers its loss, possibly incurring in overfit.

Structure

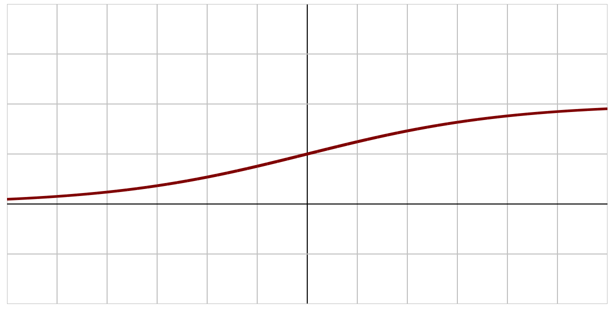
Università di Pisa

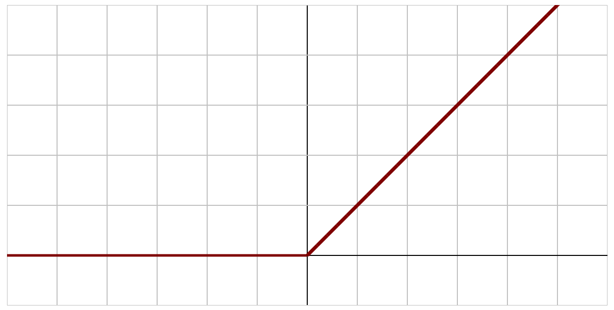
Activation functions

Activation functions impact the flow of data throughout the network, and their outputs are called *activations*. Different activations define different *representations* of the data, which, unlike in PCA, are dependent on learned parameters.

	Formulation	**Heads**
Identity	$\omega^T x$	1
Logistic	$\frac{1}{1+e^{-\omega^T x}}$	1
ReLU	$\max\{0, \omega^T x\}$	1
Softmax	$\frac{exp(\omega_i x_i)}{exp(w^T x)}$	≥ 1
...	...	...

Remember, $\omega^T x$ is the sum of the incoming edges in the node.





Logistic (top) and ReLU (bottom) activation functions.

Activation functions

As the last operation in the network, activation functions play different roles.

	Formulation	**Heads**	**Task**
Identity	$\omega^T x$	1	Regression
Logistic	$\frac{1}{1 + e^{-\omega^T x}}$	1	Classification*
ReLU	$\max\{0, \omega^T x\}$	1	Regression*
Softmax	$\frac{exp(\omega_i x_i)}{exp(w^T x)}$	≥ 1	Classification*

*Indirectly: classification is often rendered as a continuous value (to turn into discrete), while regression may be bounded, e.g., to be positive by ReLU.

Architectures

Università di Pisa

To fit or not to fit, this is the question

Given their high capacity, and innate ability to encode data, networks are often not fit from scratch. Rather, a network is fit on a task, then *adapted* to other tasks.

Fine-tuning

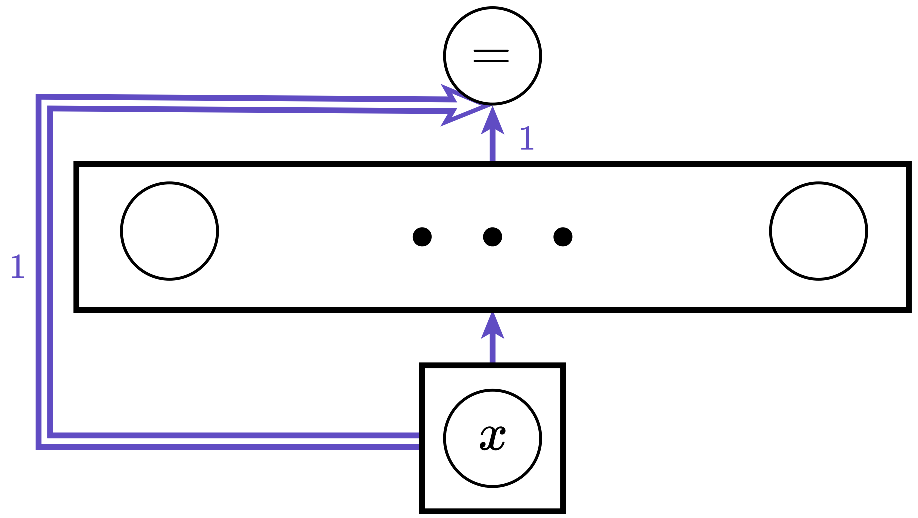
A network is first trained on a general task and a large dataset, then some more training is performed on the smaller, specialized task.

Adapters

Small networks are fit to "steer" the parameters of a larger network to solve a specific task. They are then included in the desired architecture.

Network architectures: ResNet

Architectures have proven to be extremely important, and in several cases, the application dictates the network architecture. On tabular dataset, residual networks (ResNets) are a particularly strong baseline. They have a functional form $f(x) = x + h(x)$.



A residual block: a node is forwarded to the next layer, and to the one after it as well. Residual (also called *skip*) connections allow a more effective backpropagation. Weights on the skip connection are set to 1 to preserve data.

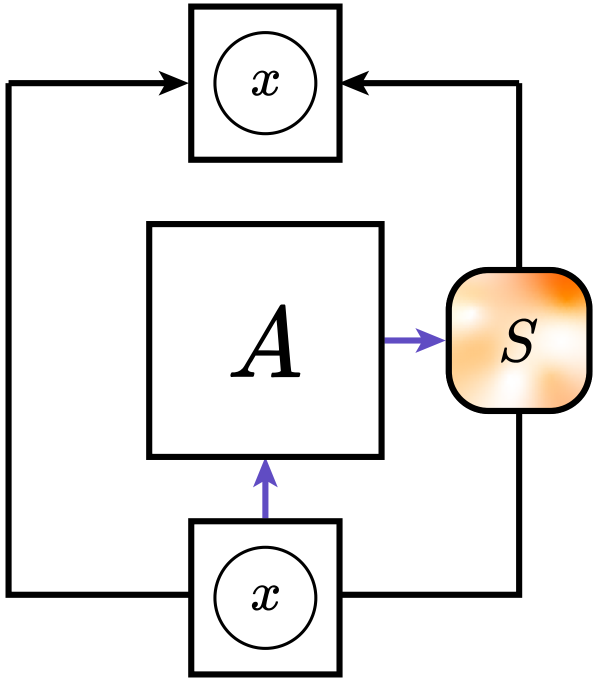
Network architectures: FeatureTransformer

Università di Pisa

Blocks constructs circuits of similarity, computing degrees of "attention" between features, and representations. Similarity then weighs on skip connections.

Functional form:

$f(x^i) = f(x^i) + \sum_{j \neq i} \alpha_{i,j} x^j.$



An attention block: similarities between representations are computed through a (scaled) multiplication. Addition through a residual connection allows representations to explicitly influence each other

References

Università di Pisa

	Reference
Neural Networks	Deep Learning. I. Goodfellow, Y. Bengio, A. Courville. Sections 6.1-6.5, 7.1, 7.8
Feature Transformer	On Embeddings for Numerical Features in Tabular Deep Learning
ResNet	Deep Residual Learning for Image Recognition