Physics-Guided Optimizations

Inform and Inspire AI Models with Physics Principles



G. Grosso | gaiag795@mit.edu

Overview of the lecture

- PART A: Physics-**INFORMED** ML:
  - How to incorporate domain knowledge in the ML solution
  - Especially for data and task in the physics domain

— coffee break —

- PART B: Physics-**INSPIRED** ML:
  - How AI optimization mimics physics dynamics, naturally evolving to minimize their energy.
  - Dive into the first steps towards artificial intelligence and ML



PART A: Physics-informed AI



Why *informing* machine learning with physics?

Modeling complex physical systems behavior is a fundamental task for scientific and engineering progress

Top-down *(physics)*
strong modeling assumptions from physical laws

- PROS: *principled* and *interpretable* models
- CONS: Strong modeling *assumptions* may not fit real data well. Complex physical systems are *hard* to model (*intractable* theory)

Bottom-up *(ML)*
large amount of (labeled) training data to learn from

- PROS: High flexibility / *adaptability*
- CONS: Unconstrained model can lead to unphysical / *unreliable* outcomes

Physics-guided ML
Incorporating physics assumptions in very expressive models



Physics-informed AI

…the art of inductive bias

Relational inductive biases, deep learning, and graph networks

Box 2: Inductive biases

Learning is the process of apprehending useful knowledge by observing and interacting with the world. It involves searching a space of solutions for one expected to provide a better explanation of the data or to achieve higher rewards. But in many cases, there are multiple solutions which are equally good (Goodman, 1955). An *inductive bias* allows a learning algorithm to prioritize one solution (or interpretation) over another, independent of the observed data (Mitchell, 1980). In a Bayesian model, inductive biases are typically expressed through the choice and parameterization of the prior distribution (Griffiths et al., 2010). In other contexts, an inductive bias might be a regularization term (McClelland, 1994) added to avoid overfitting, or it might be encoded in the architecture of the algorithm itself. Inductive biases often trade flexibility for improved sample complexity and can be understood in terms of the bias-variance tradeoff (Geman et al., 1992). Ideally, inductive biases both improve the search for solutions without substantially diminishing performance, as well as help find solutions which generalize in a desirable way; however, mismatched inductive biases can also lead to suboptimal performance by introducing constraints that are too strong.

Inductive biases can express assumptions about either the data-generating process or the space of solutions. For example, when fitting a 1D function to data, linear least squares follows the constraint that the approximating function be a linear model, and approximation errors should be minimal under a quadratic penalty. This reflects an assumption that the data-generating process can be explained simply, as a line process corrupted by additive Gaussian noise. Similarly, $L2$ regularization prioritizes solutions whose parameters have small values, and can induce unique solutions and global structure to otherwise ill-posed problems. This can be interpreted as an assumption about the learning process: that searching for good solu is easier when there is less ambiguity among solutions. Note, these assumptions need explicit—they reflect interpretations of how a model or algorithm interfaces with the



Physics-informed AI

…the art of inductive bias

Algorithmic design has many analogies to *experimental design* in scientific domain:

- Once the objective of the algorithm is defined, one can formulate a priori *hypotheses* on properties of the **model solution** (e.g. which symmetries it should respect, its domain, which mathematical properties, …) and the optimal **data representation** (e.g. how to make relevant information readily available to the algorithm)
- *Assumptions* about the learning objective and expected solution can guide the design of effective ML algorithms
- Assumptions can be translated in *induced bias* on the algorithm, and their impact on the latter performances can be tested.

Adopting a *scientific approach* to model design is a principled way to explore the space of solution and can make the process of *model optimization* more *efficient*.



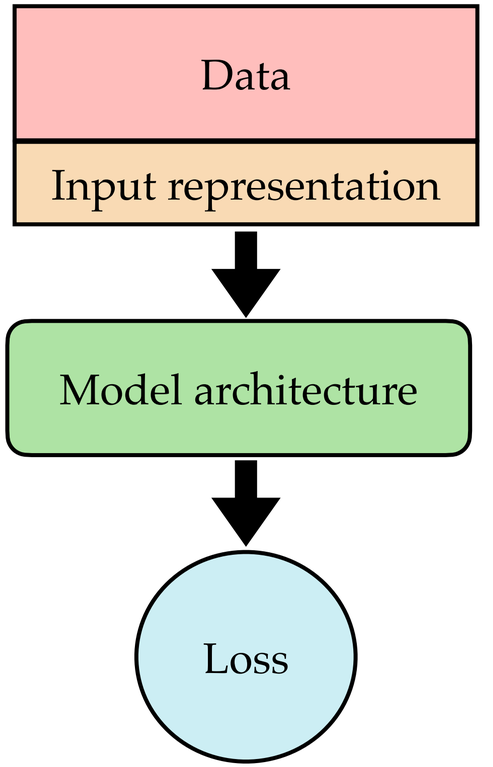
Physics-informed AI

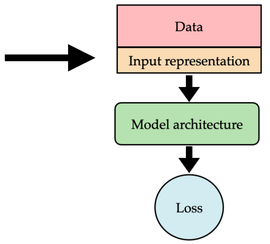
The art of inductive bias

There are multiple ways in which *assumptions* can be translated into effective *induced bias*:

- **Representation bias (data preparation)**: choose which information to feed into the algorithm and in which form (data structure and selection)
- **Hardware constraint (architecture design)**: hard-code symmetry constraints (invariance or equivariance) in the model architecture.
- **Software constraint (loss function design)**: soft-impose conservation laws adding penalty terms in the loss function.
- **Implicit bias (data augmentation)**: hand-craft training examples or their abundance to help the model learn specific features of the problem.

General scheme of a ML model



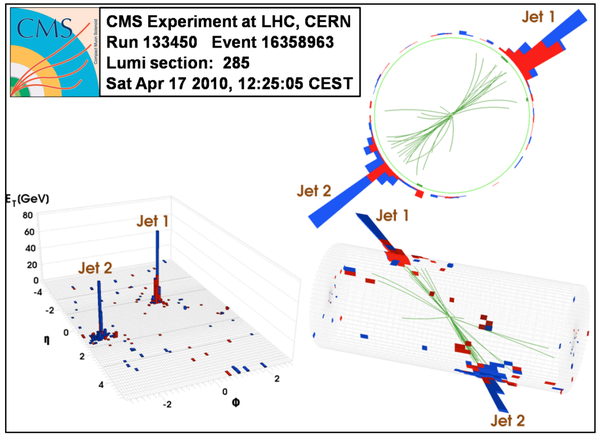


Data representation

Identifying relevant structures in data and preserve them in their representation is the first step for successfully solve a machine learning task

Physics data are typically highly *structured* (*t*ime series, images, point clouds, …)

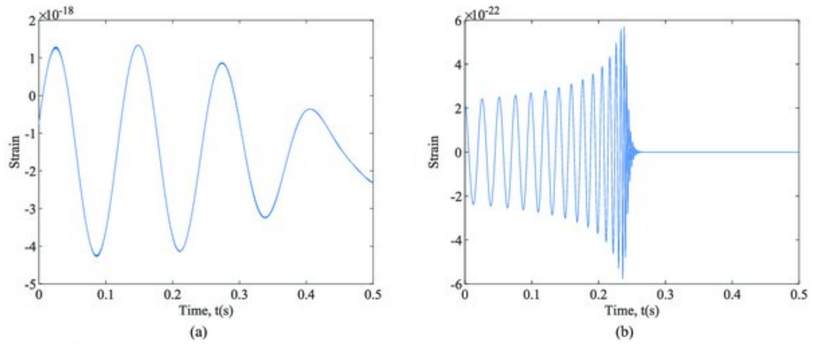
Different mathematical representations of the data capture different *relations* between pieces of information encoded in the data.

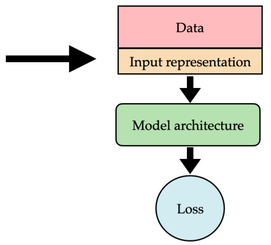




ESA/Hubble & NASA, A. Riess, D. Thilker, D. De Martin (ESA/Hubble), M. Zamani (ESA/Hubble)

10.1109/ACCESS.2021.3139850





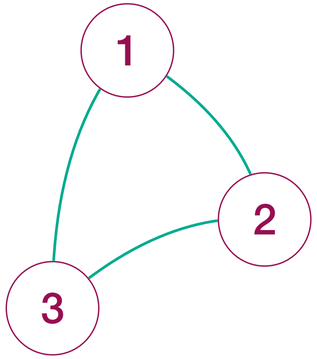
Graph-structured data

Generally, we can describe the *relations* between properties encoded in data using graphs.
Graphs are mathematical structures defined[1] as a 3-tuple

$$G = (\mathbf{u}, V, E)$$

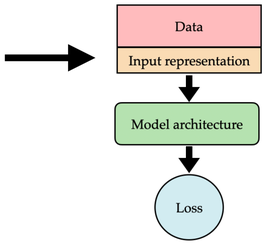
Where:

- $\mathbf{u}$ are the **global attributes** (e.g. feature of the system encoded in the graph)
- $V = \{\mathbf{v}_i\}_{i=1}^{N_v}$ is the set of **nodes** (of cardinality N_v) and each $\mathbf{v}_i$ represents a **node attributes**
- $E = \{(\mathbf{e}_k, s_k, r_k)\}_{k=1}^{N_e}$ is the set of **edges** (of cardinality N_e) and each $\mathbf{e}_k$ represents an **edge attributes**, s_k the index of the sender and r_k the index of the receiver.



Physics data can generally be encoded in graphs.
Depending of the specific properties of the data, special types of graph can be used.

[1] definition and conventions from Relational inductive biases, deep learning, and graph networks



Grid-organized data

In many physics applications collected data are organized in a *regular grid* in (Euclidean) space.
These data can be described using *graphs*.

Images:
Images are a special case of graph where nodes are organized in a *2D lattice* in Euclidean space.
Each pixel has a *fixed set of neighbors*. Spacial relations between neighbors define the edges of the graph.

Examples of images in experimental physics data are: astronomical images, 1-layer detector hits

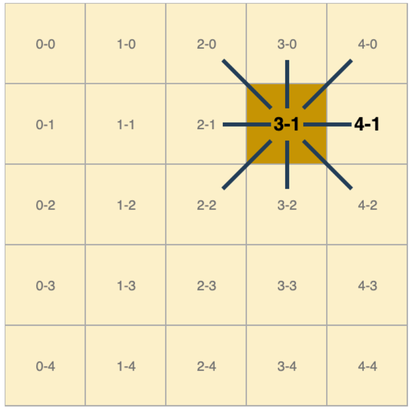
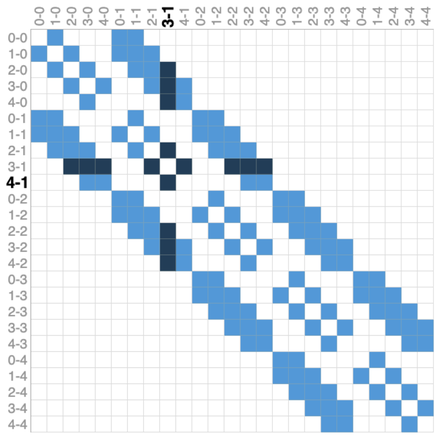
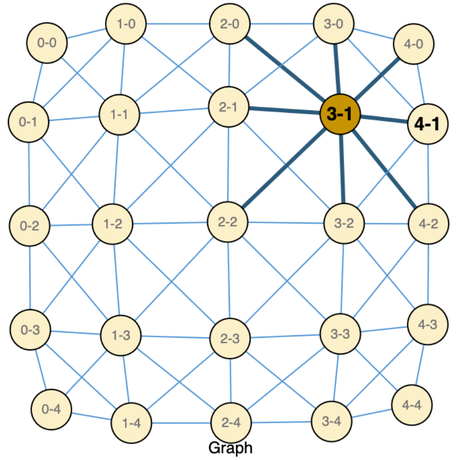
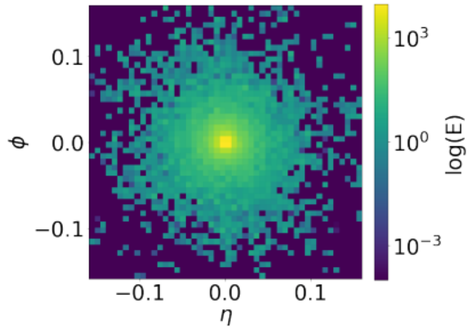


Image Pixels


Adjacency Matrix

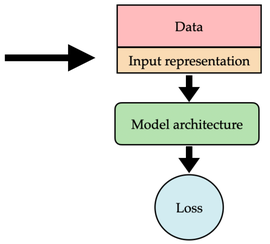
https://distill.pub/2021/gnn-intro/


Graph



Photon shower on EM calorimeter (Geant simulation) 2411.05996


ESA/Hubble & NASA, A. Riess, D. Thilker, D. De Martin (ESA/Hubble), M. Zamani (ESA/Hubble)

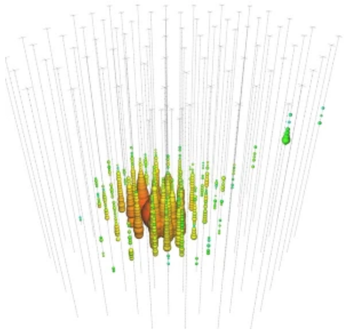


Grid-organized data

In many physics applications collected data are organized in a *regular grid* in (Euclidean) space.
These data can be described using *graphs*.

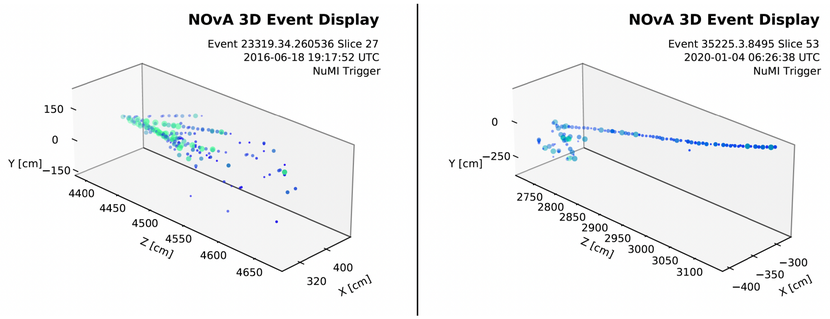
3D-lattice data:

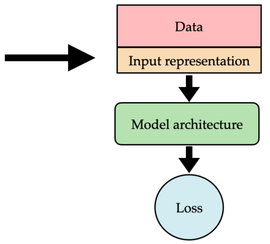
In experimental particle physics detectors often cover an extended volume in space to capture a particle trajectory or decay. The detector geometry is constant giving rise to 3D grid.



A reconstructed detection in the **IceCube** sensor array of a suspected **neutrino** with an energy of 252.7 trillion electron-volts. The strings of detectors descend from the surface of the ice to more than two kilometers below. Credit: Jakob van Santen doi.org/10.1038/nature.2013.13026

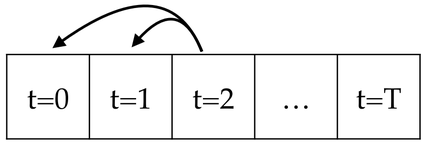
Depictions of particle traces at the far detector of the NOva experiment. https://www.physics.wisc.edu/2025/02/26/nova-study-sets-tighter-limit-on-sterile-neutrinos/

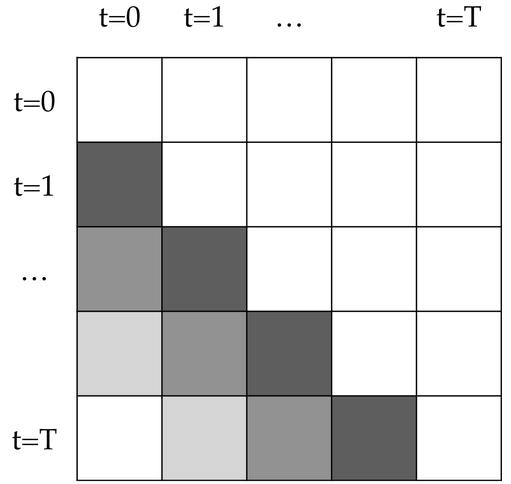




Grid-organized data

In many physics applications collected data are organized in a *regular grid* in (Euclidean) space.
These data can be described using graphs.





Time series:

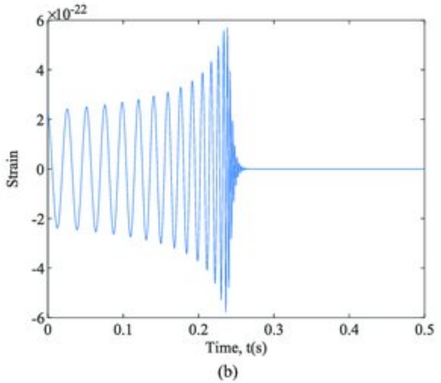
Constant rate sampling of sensory data also result in 1 or higher dimensional grids.

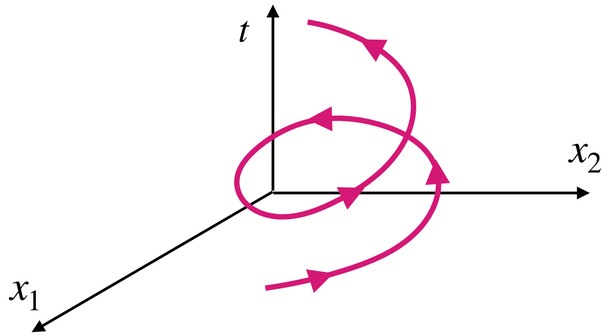
Due to causality, each time stamp depends only the previous steps.

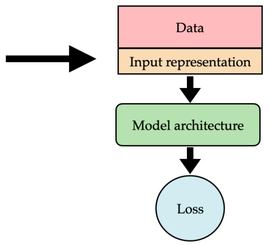
Dependence may decays the distance in times increases.

Example of time series in fundamental physics: gravitational waves

10.1109/ACCESS.2021.3139850







Grid-organized data

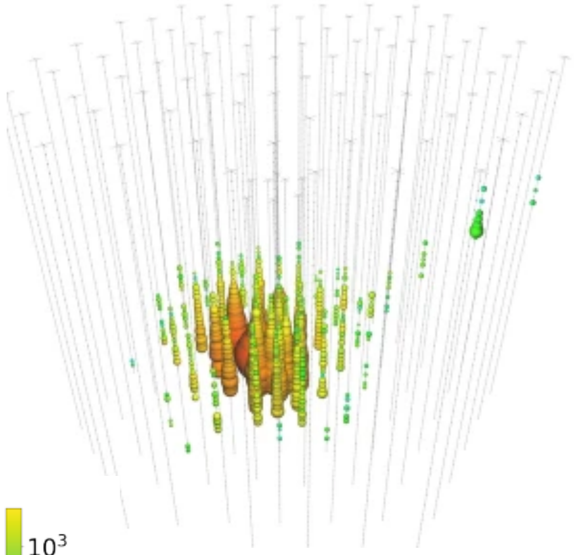
In many applications in **fundamental physics** nodes containing relevant information in regular grids are *sparse*.

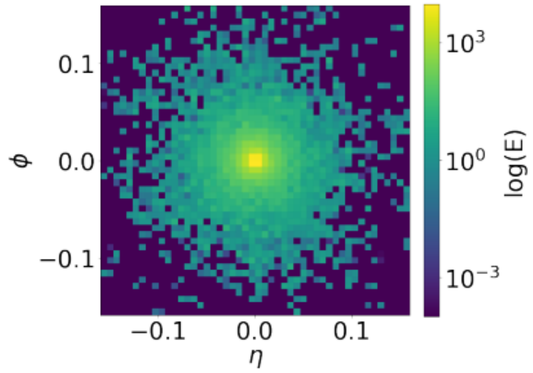
In these cases, regular graphs can be simplified by *dropping irrelevant nodes*, leading to a more *efficient* representation.

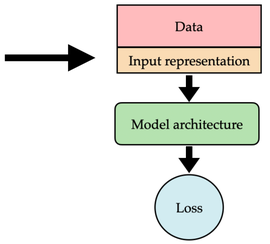
https://www.flickr.com/photos/nasahubble/54533174375/



doi.org/10.1038/nature.2013.13026







Spacial-organized data

In many applications in **fundamental** physics nodes containing relevant information in regular grids are *sparse*.

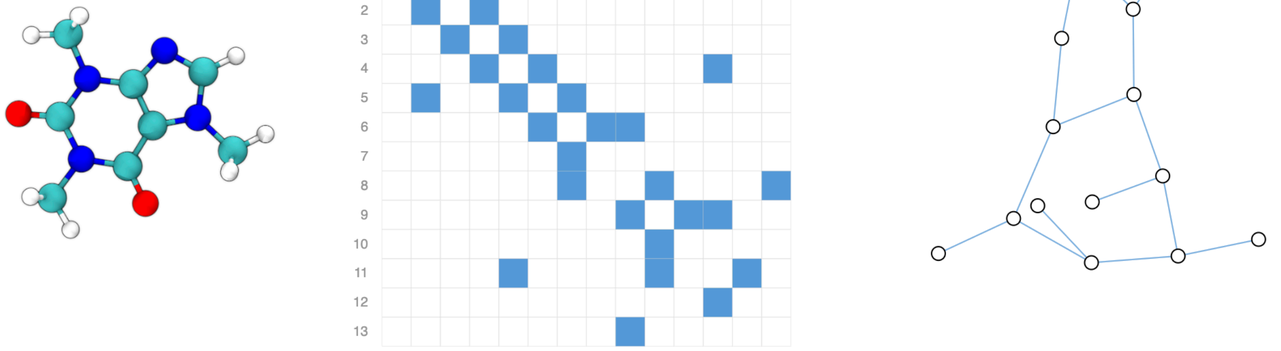
In these cases, regular graphs can be simplified by *dropping irrelevant nodes*, leading to a more *efficient* representation.

Molecules:

Covalent bonds between atoms in molecules define a stable 3*D geometry* for the molecule in Euclidean space.

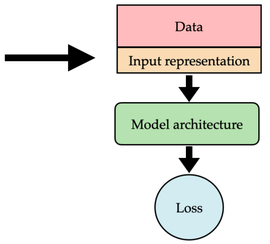
Different pairs of atoms and bonds result in different distances between atoms.

https://distill.pub/2021/gnn-intro/



(Left) 3d representation of the Caffeine molecule (Center) Adjacency matrix of the bonds in the molecule (Right) Graph representation of the molecule.





Spacial-organized data

In many applications in **fundamental** physics nodes containing relevant information in regular grids are *sparse*.

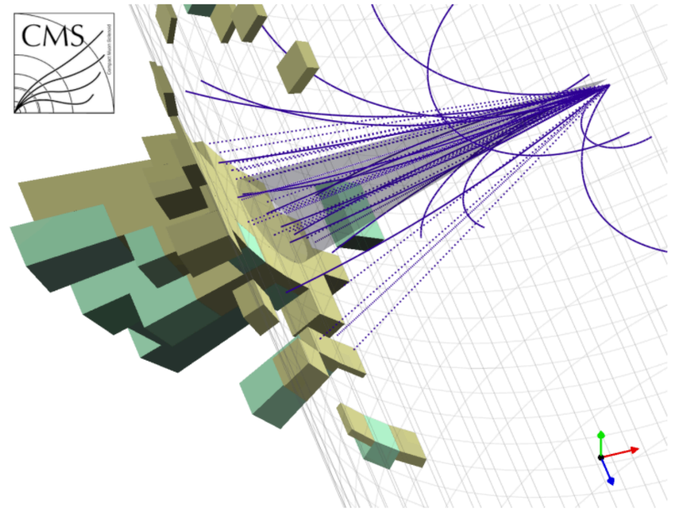
In these cases, regular graphs can be simplified by *dropping irrelevant nodes*, leading to a more *efficient* representation.

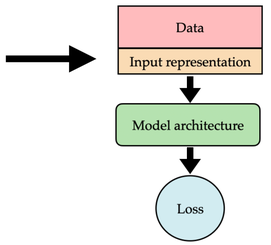
Particle Jets:

In high-energy collisions (e.g. at the LHC), energetic quarks or gluons produced in a hard scattering process cannot exist freely due to **color confinement**. Instead, they undergo a process known as a **parton shower**, followed by **hadronization**, resulting in a collimated spray of particles called a **jet**.

The resulting hadrons are detected in calorimeters or tracking systems.

Representation of a particle jet observed in the CMS experiment



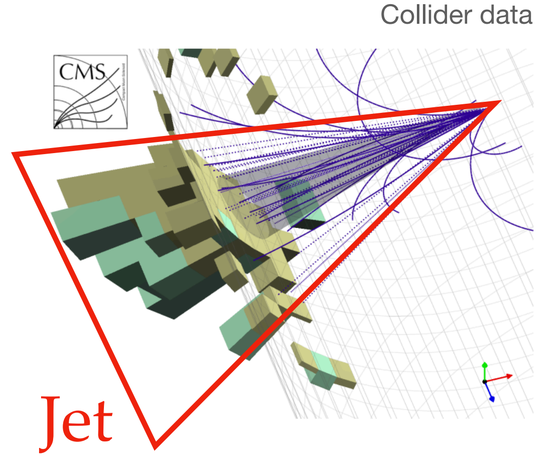


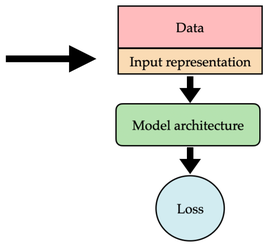
Feature engineering

Identifying relevant structures in data and preserve them in their representation is the first step for successfully solve a machine learning task

- Different representations can be optimal for different tasks
- *Feature engineering* exploiting domain knowledge can help to simplify some tasks and reach better solution with less data

From *sensor-level* representation we reconstruct the *object-level* information

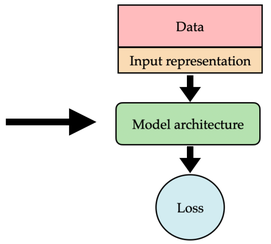




Conservation laws and symmetries

Identifying relevant structures in data and preserve them in their representation is the first step for successfully solve a machine learning task

- Physical systems obey fundamental *conservation laws* (e.g., energy, momentum, charge).
- According to **Noether's theorem**, every conservation law corresponds to a continuous *symmetry* of the system (e.g., time invariance → energy conservation).
- Symmetries define *constraints* on how physical quantities can transform under operations like translation, rotation, or reflection.
- Recognizing the relevant symmetry groups can help reduce the search space for learning.
- Enforcing expected physical invariance or equivariance can help increase **generalization** and **reliability** of the ML solution.



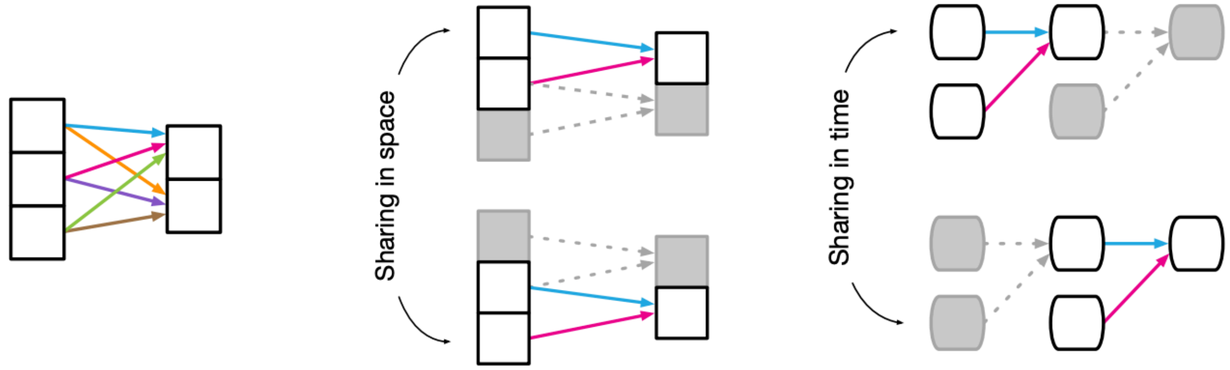
Architecture design

Data structures and learning objectives motivates architecture design

Architectures should be designed in such a way to guarantee that the relevant information flows from the input all the way to the output

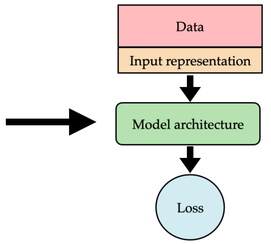
- Invariant/equivariant
- Preserve the data structure/topology
- Prefer models that respect priors on the expected solution

Connectivity is a critical aspect of model design



(a) Fully connected (b) Convolutional (c) Recurrent

Not very informative! → Encode assumptions on relational structure in the data



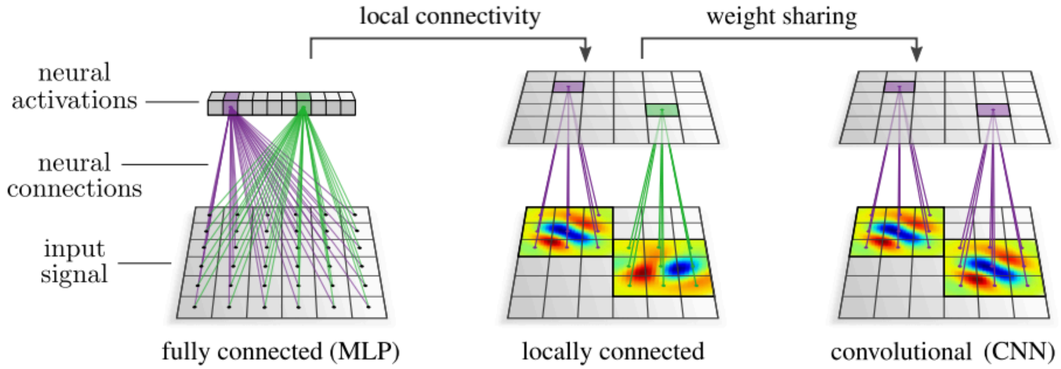
Architecture design

Convolutional NNs:

Strong Inductive Bias: *spacial locality* and *space translation equivariance*

CNNs embed **prior knowledge** about image-like structures:

- *Spacial locality*: nearby pixels are more related than distant ones
  → Convolutional layers connect each neuron to a **local receptive field** of neighboring pixels.
- *Translation equivariance*: patterns recur across space
  → The same filter is applied across all spatial locations (weight sharing), encouraging learning of **position-independent features**

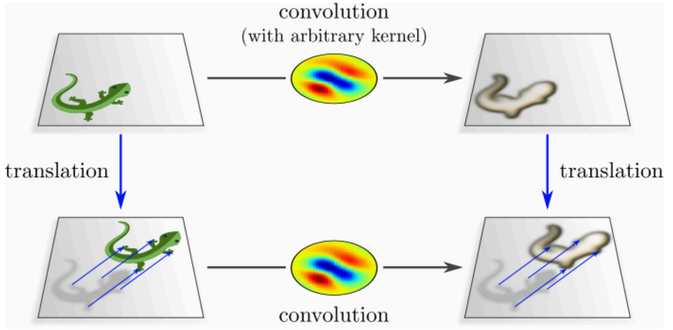


Basic CNN update

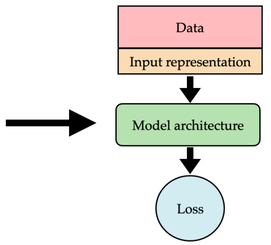
The core operation is a *convolution*:

$$y_{i,j}^{(k)} = \sigma\left(\sum_{c=1}^{C}\sum_{u,v} W_{u,v}^{(k,c)}\, x_{i+u,j+v}^{(c)} + b^{(k)}\right)$$

$x_{i,j}^{(c)}$: input value at position (i, j) in channel c
$W^{(k,c)}$: convolutional kernel weights for output channel k, input channel c
σ: non-linear activation function (e.g., ReLU)
$y_{i,j}^{(k)}$: output feature at position (i, j) in channel k



M. Weiler "Equivariant And Coordinate Independent CNNs"



Architecture design

Convolutional NNs:

Parameter Efficiency

- **Weight sharing** reduces the number of parameters (CNNs are more efficient than DNNs for images).

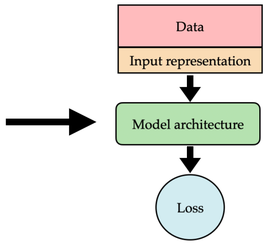
Hierarchical Feature Learning

- Initial layers learn simple patterns (edges, gradients).
- Deeper layers capture complex structures (shapes, object parts).
- Mimics the **hierarchical processing** in the human visual system.
- Enables **scalable architectures** like ResNet or EfficientNet.

Pooling and Invariance

- Pooling layers (e.g., max or average pooling) downsample the feature maps.
- Provide **translation invariance** to small shifts and reduce spatial resolution.
- Helps in **summarizing features** and reducing computational cost.





Architecture design

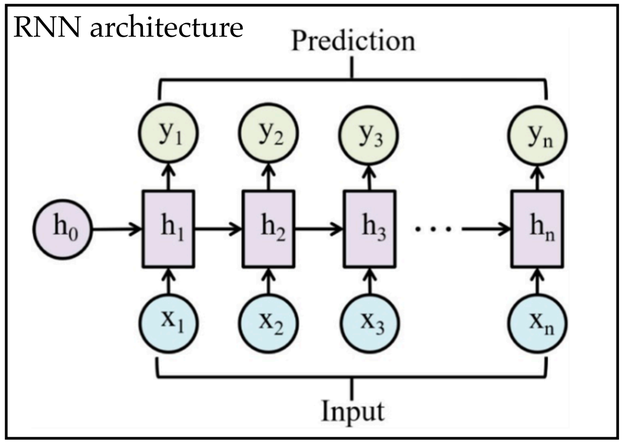
Recurrent NNs:

Strong Inductive Bias: ***sequential correlation*** **and** ***time translation equivariance***

RNNs embed **prior knowledge** about time-series:

- *Sequential correlation*: input is processed **sequentially**, state at time t depends on all previous ones.
  → Recurrent layers are unrolled over time: state at time t takes as input the state at time $t-1$.
- *Time translation equivariance*: patterns recur across time
  → The same weight matrices are applied across all time steps (weight sharing), encouraging learning of **time-independent features**

https://doi.org/10.3390/info15090517



Basic RNN cell update

Time series: $x = (x_1, \ldots, x_T)$ with $x_t \in \mathbb{R}^d$ the input vector at time t

Hidden state at time t: $h_t \in \mathbb{R}^k$

The update rule is: $h_t = \sigma(W_{xh}x_t + W_{hh}h_{t-1} + b_h)$

$W_{xh} \in \mathbb{R}^{d\times k}$ input-to-hidden weight matrix

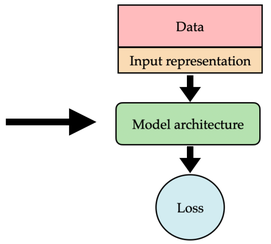
$W_{hh} \in \mathbb{R}^{k\times k}$ hidden-to-hidden (recurrent) weight matrix

$b_h \in \mathbb{R}^k$ hidden bias

The hidden state h_{t-1} encodes the summary of all past inputs $(x_1, \ldots, x_{t-1})$

Each new input x_t modulates the current memory h_{t-1} to produce the update state h_t

This recurrence allows the network to **learn temporal correlations** without needing to explicitly store all past inputs.



Architecture design

Recurrent NNs:

Strong Inductive Bias: ***sequential correlation*** **and** ***time translation equivariance***
RNNs embed **prior knowledge** about time-series:

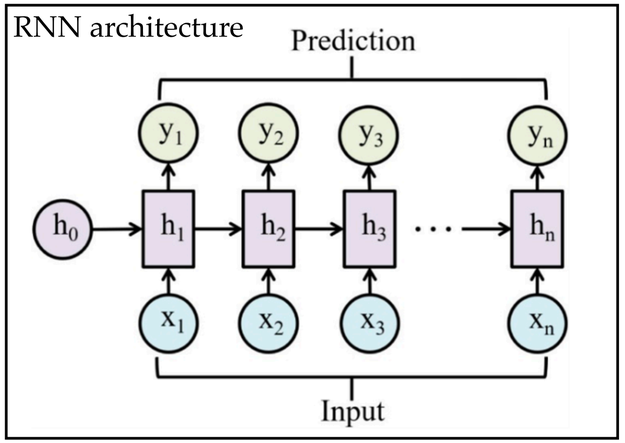
- *Sequential correlation*: input is processed **sequentially**, state at time t depends on all previous ones.
  → Recurrent layers are unrolled over time: state at time t takes as input the state at time $t-1$.
- *Time translation equivariance*: patterns recur across time
  → The same weight matrices are applied across all time steps (weight sharing), encouraging learning of **time-independent features**

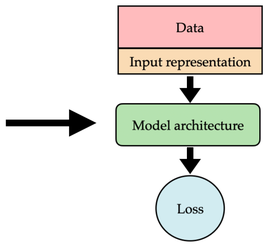
Sequence unrolling:
The RNN is typically unrolled over time to form a chain:

$$h_1 = \sigma(W_{xh}x_1 + W_{hh}h_0 + b_h)$$
$$h_2 = \sigma(W_{xh}x_2 + W_{hh}h_1 + b_h)$$
$$\dots$$
$$h_T = \sigma(W_{xh}x_T + W_{hh}h_{T-1} + b_h)$$

This allows **backpropagation through time** to train the weights using gradient descent.

https://doi.org/10.3390/info15090517





Architecture design

Recurrent NNs:

Limitations and extensions:

- In RNNs, **gradients either vanish or explode** as they are propagated backward through many time steps.
- **Memory-augmented RNNs** introduce gating mechanisms that regulate information flow across time steps

Example — Long short term memory (LSTM):

In LSTMs, a **cell state** c_t stores in *memory* the relevant information; **gated units** control how much of the previous memory to retain, how much new information to add, and what to output.

Cell state update:

$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t$$

Forget gate: $f_t = \sigma(W_f[h_{t-1}, x_t] + b_f)$

Input gate: $i_t = \sigma(W_i[h_{t-1}, x_t] + b_i)$

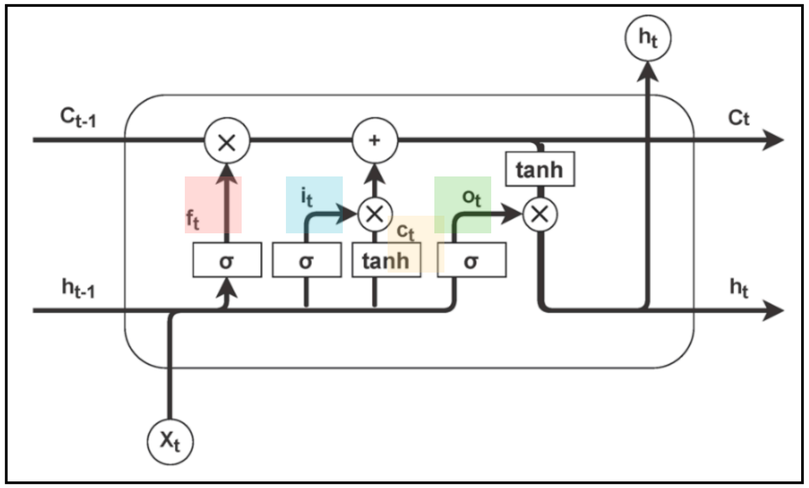
Candidate cell state: $\tilde{c}_t = \tanh(W_c[h_{t-1}, x_t] + b_c)$

Hidden state update:

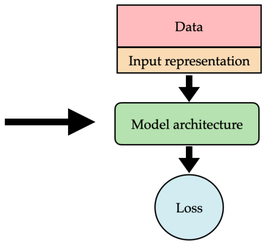
$$h_t = o_t \odot \tanh(c_t)$$

Output gate: $o_t = \sigma(W_o[h_{t-1}, x_t] + b_o)$

https://doi.org/10.3390/info15090517



These mechanisms allow LSTMs to **selectively retain or discard information**, making them ideal for modeling physical processes with **delayed or cumulative effects**.



Architecture design

Graph NNs:

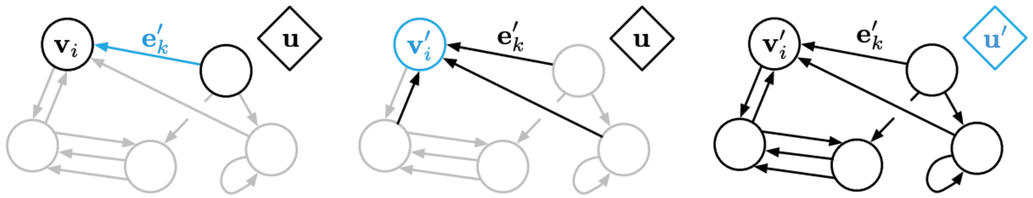
Strong inductive bias: *arbitrary relations* and *permutation invariance*

GNNs embed **prior knowledge** about any arbitrary relational data structure:

- *Arbitrary relations*: the connectivity map is hard coded in the graph.
  → GNNs determine how representation interact and are isolated, rather than those choices being determined by the fixed architecture
- *Permutation invariance*: properties of the system don't depend on the elements order in the representation
  → attributes aggregation methods are permutation invariant

Basic GNN update:

Graph Neural Networks operate on graph-structured data by iteratively updating edges, nodes and global attributes.

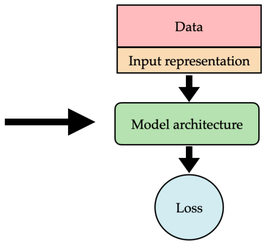


(a) Edge update (b) Node update (c) Global update

Algorithm 1 Steps of computation in a full GN block.

function GRAPHNETWORK(E, V, $\mathbf{u}$)
  for $k \in \{1 \dots N^e\}$ **do**
    $\mathbf{e}'_k \leftarrow \phi^e(\mathbf{e}_k, \mathbf{v}_{r_k}, \mathbf{v}_{s_k}, \mathbf{u})$ ▷ 1. Compute updated edge attributes
  end for
  for $i \in \{1 \dots N^n\}$ **do**
    let $E'_i = \{(\mathbf{e}'_k, r_k, s_k)\}_{r_k=i,\ k=1:N^e}$
    $\bar{\mathbf{e}}'_i \leftarrow \rho^{e\to v}(E'_i)$ ▷ 2. Aggregate edge attributes per node
    $\mathbf{v}'_i \leftarrow \phi^v(\bar{\mathbf{e}}'_i, \mathbf{v}_i, \mathbf{u})$ ▷ 3. Compute updated node attributes
  end for
  let $V' = \{\mathbf{v}'\}_{i=1:N^v}$
  let $E' = \{(\mathbf{e}'_k, r_k, s_k)\}_{k=1:N^e}$
  $\bar{\mathbf{e}}' \leftarrow \rho^{e\to u}(E')$ ▷ 4. Aggregate edge attributes globally
  $\bar{\mathbf{v}}' \leftarrow \rho^{v\to u}(V')$ ▷ 5. Aggregate node attributes globally
  $\mathbf{u}' \leftarrow \phi^u(\bar{\mathbf{e}}', \bar{\mathbf{v}}', \mathbf{u})$ ▷ 6. Compute updated global attribute
  return $(E', V', \mathbf{u}')$
end function

Relational inductive biases, deep learning, and graph networks



Architecture design

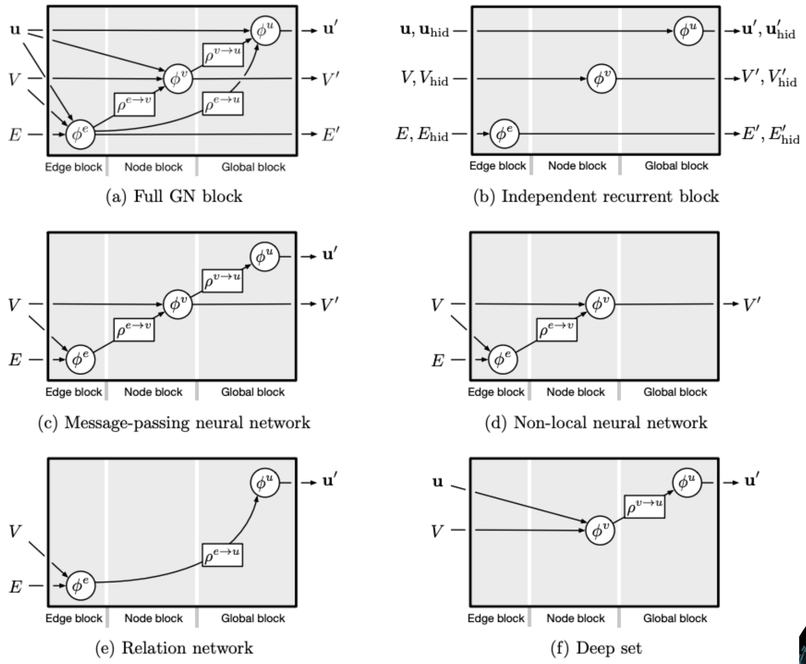
Graph NNs:

Strong inductive bias: *arbitrary relations* and *permutation invariance*

GNNs embed **prior knowledge** about any arbitrary relational data structure:

- *Arbitrary relations*: the connectivity map is hard coded in the graph.
  → GNNs determine how representation interact and are isolated, rather than those choices being determined by the fixed architecture
- *Permutation invariance*: properties of the system don't depend on the elements order in the representation
  → attributes aggregation methods are permutation invariant

Several adaptations of the basic update block showed before can be designed, depending on the nature of the data and task



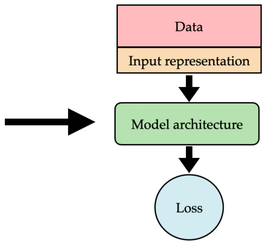
(a) Full GN block (b) Independent recurrent block

(c) Message-passing neural network (d) Non-local neural network

(e) Relation network (f) Deep set

Relational inductive biases, deep learning, and graph networks

Architecture design



Transformers:

Relatively bias-free:

- No assumptions of locality (unlike CNNs)
- No sequential order assumption (unlike RNNs)
- ***Permutation equivariance*** in self-attention allows flexible reasoning over arbitrary sets or graphs.

- Input: list of elements with d features: $x = (x_1, \ldots, x_n) \in \mathbb{R}^{n \times d}$
- ***Positional encoding*** can be reintroduced but is not hard coded.

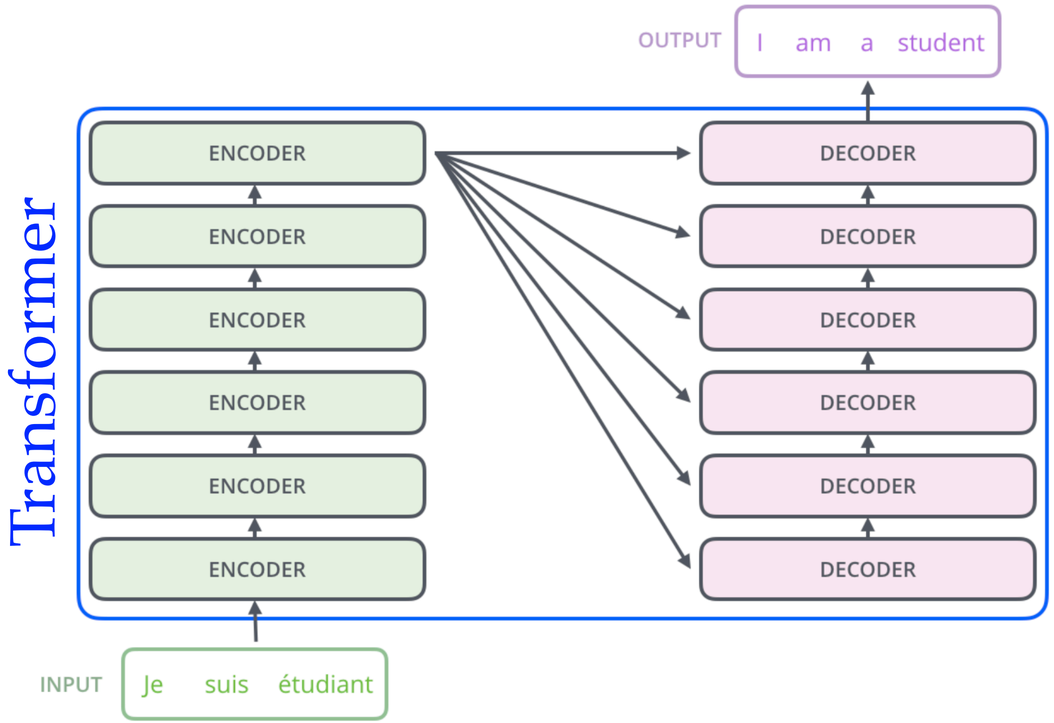
Basics of Transformers module:

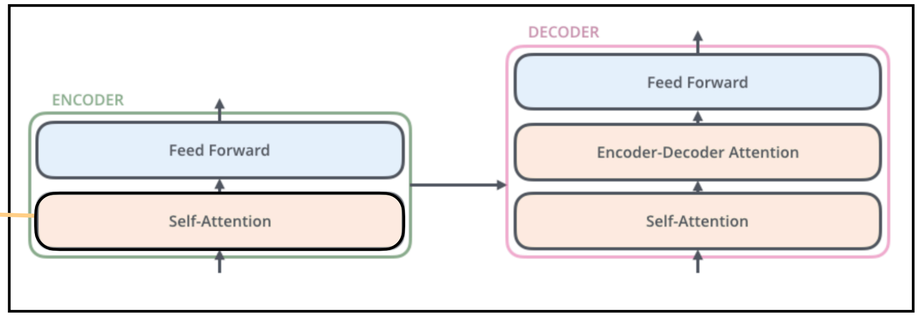
- The key operation in Transformers is the ***self-attention mechanism***:

Query: $Q = x \cdot W_Q \in \mathbb{R}^{n \times d_k}$
Key: $K = x \cdot W_K \in \mathbb{R}^{n \times d_k}$ with $W_Q, W_K, W_V \in \mathbb{R}^{d \times d_k}$
Value: $V = x \cdot W_V \in \mathbb{R}^{n \times d_k}$

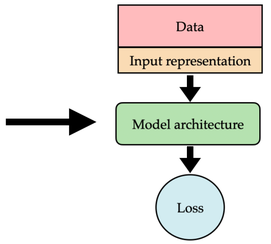
$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V$$

- The self-attention block allows to model arbitrary *pair-wise relations* between any two input elements





https://jalammar.github.io/illustrated-transformer/

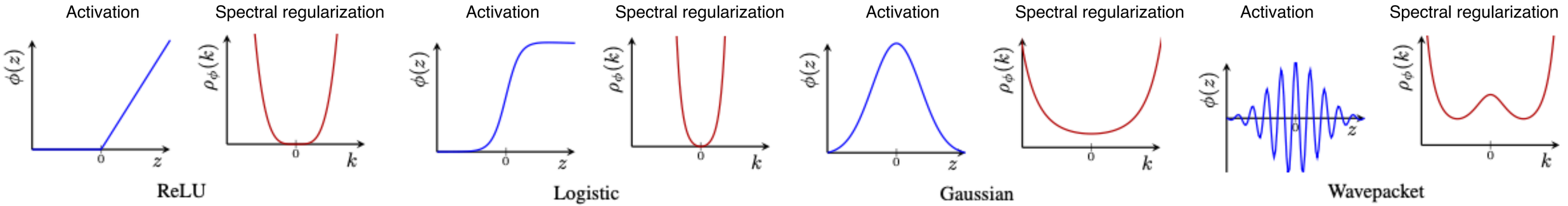


Architecture design

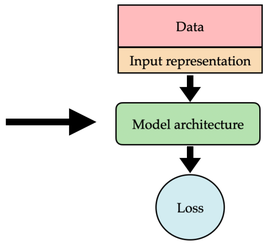
The role of activation

The **activation function** calculates the output of a computing unit based on its individual inputs and weights.

- Activation functions strongly influence what a network "prefers" to learn in terms of *frequency content*.
- The choice of activation is strongly related with *smoothness* properties of the model.
- Thus, choosing an activation function shapes the frequency content of learned functions, analogous to selecting a *regularization prior in spectral domains*[1].



[1] https://arxiv.org/pdf/2503.10587



Architecture design

Embedding symmetries

If your data and task are known to be symmetric under a group G, then:

- Your model should respect those symmetries
- The output should be **invariant** or **equivariant** under group actions

There are multiple ways to embed symmetries in a model:

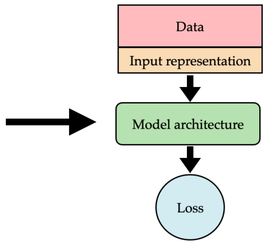
- Learn them from data (Later in today's lecture)
- Suggest them with penalties
- Encode them in the architecture

Data efficient (fewer parameters)

Exact constraint (improves generalization)

Rigidity penalizes accuracy





Architecture design

Embedding symmetries

If your data and task are known to be symmetric under a group G, then:

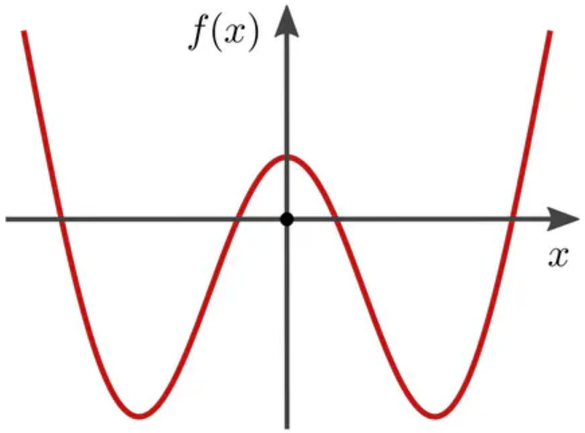
- Your model should respect those symmetries
- The output should be **invariant** or **equivariant** under group actions

Illustrative example: fitting a symmetric function with a polynomial[1]

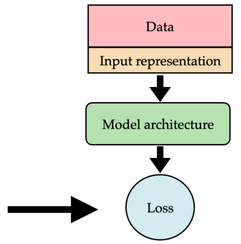
- Non-equivariant naive approach: $y(x) = \sum_{n=0}^{N} w_n x^n$ (n parameters)
- Equivariant approach $y_{\text{symm}}(x) = \sum_{n \text{ even}} w_n x^n$ ($n/2$ parameters)

  More efficient (half number of parameters), more generalizable (symmetric by construction)

symmetric: $f(-x) = f(x)$



Illustrative example from: https://maurice-weiler.gitlab.io/blog_post/cnn-book_1_equivariant_networks/#how-to-construct-equivariant-models



Loss function design

Penalty constraints in the loss function bias the model solution towards a preferred family of models.

- Machine learning models, especially over-parameterized ones, can easily *overfit* the data
- Regularization terms in the loss introduce soft constraints that reduce variance at the cost of biasing the solution towards a preferred class of functions. The aim is to prevent overfitting and promote *generalization* by encoding prior knowledge.
- The overall loss function minimized during training:

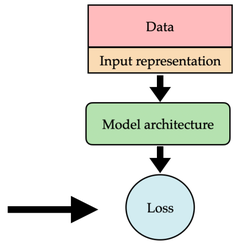
$$L_{\text{total}} = L_{\text{data}} + \lambda R(\theta)$$

Main objective (fitting the data)

Hyper-parameter controlling the strength of the regularization

Regularization term (Function of the trainable parameter of the model θ)





Loss function design

Penalty constraints in the loss function bias the model solution towards a preferred family of models.

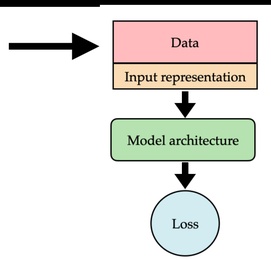
Common penalties:

- **L2 regularization (weight decay)**: $R(\theta) = ||\theta||_2^2$ (promotes *smoothness*)
- **L1 regularization**: $R(\theta) = ||\theta||_1$ (promotes *sparsity*)
- **Entropy regularization**: $R(p) = \sum_i p_i \log(p_i)$ (encourages *uncertainty* and *exploration* in a probabilistic output)
- **Physics-based constraints** encourage to satisfy conservation laws
  - **Physics Informed NN (PINN)**: is a neural network trained not just to fit data, but also to satisfy a known physical law, typically expressed as a partial differential equation (PDE):

$$L[f_\theta] = \sum_{i=1}^{N} L_{\text{data}}\left(y_i, f_\theta(x_i, t_i)\right) + R_{\text{phys}}(\theta)$$

$$R_{\text{phys}}(\theta) = \frac{1}{M}\sum_{j=1}^{M} \left| \mathcal{F}[f_\theta](x_j, t_j) \right|^2$$

Partial differential operator: $\mathcal{F}[f] = \partial_t f + u\partial_x f$
Must satisfy: $\mathcal{F}[f(x,t)] = 0$, for $x \in \Omega$, $t \in [0,T]$



Data Augmentation

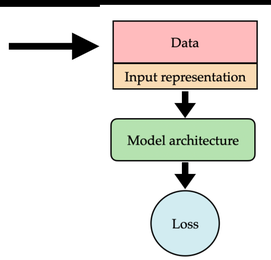
Hand-craft training examples can expose relevant corners of the solution space

Data-Augmentation (DA) is a data-driven and informed regularization strategy that artificially increases the number of training samples.

- DA is an *implicit* regularization, as it is not a function of a model's parameter but a function of the training samples.
- DA can be used to learn *invariance* or *equivariance* to specific transformations, even when those are not groups.
- DA requires more *domain knowledge* to be successful than weight decay (luckily we have plenty of it in scientific domains! :))

https://arxiv.org/pdf/2204.03632



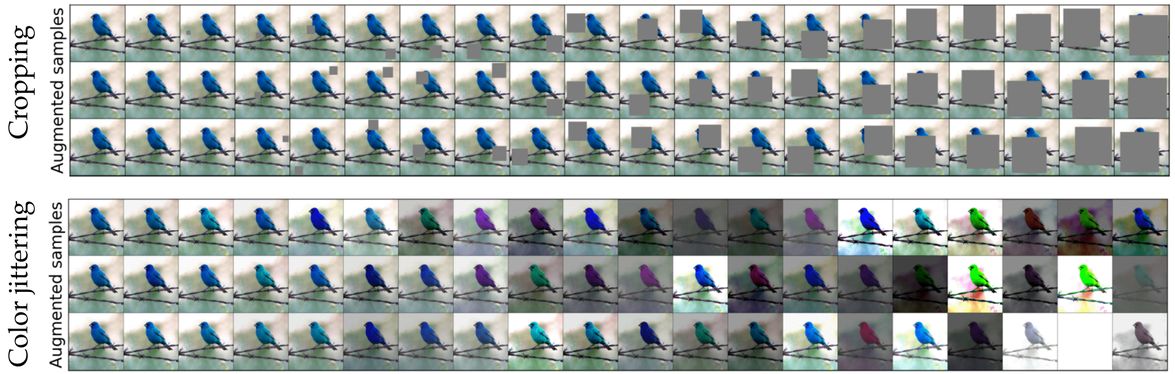


Data Augmentation

Hand-craft training examples can expose relevant corners of the solution space

Data-Augmentation (DA) is a data-driven and informed regularization strategy that artificially increases the number of training samples.

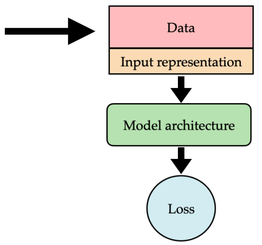
- DA is an *implicit* regularization, as it is not a function of a model's parameter but a function of the training samples.
- DA can be used to learn *invariance* or *equivariance* to specific transformations, even when those are not groups.
- DA requires more *domain knowledge* to be successful than weight decay (luckily we have plenty of it in scientific domains! :))



Data augmentation is largely employed in state-of-the-art **computer vision** models

For vision, the art of augmentations design is largely based on **heuristics**

https://arxiv.org/pdf/2204.03632



Data Augmentation

Hand-craft training examples can expose relevant corners of the solution space

Data-Augmentation (DA) is a data-driven and informed regularization strategy that artificially increases the number of training samples.

- DA is an *implicit* regularization, as it is not a function of a model's parameter but a function of the training samples.
- DA can be used to learn *invariance* or *equivariance* to specific transformations, even when those are not groups.
- DA requires more *domain knowledge* to be successful than weight decay (luckily we have plenty of it in scientific domains! :))

In physics we can often define **controlled transformations** that are critical for successfully solve specific tasks.

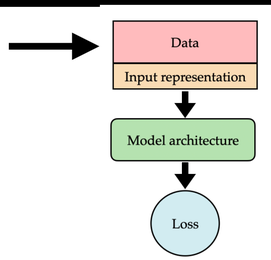
Some examples:

high-level

low-level

- Physics laws: repeated simulations of a known stochastic physics process using different seeds provide different views of the same *physics concept* and help generalize
- Model uncertainties: simulated data varying the model parameters within uncertainties can be used to learn *domain shifts* or make a model robust against them.
- Experimental conditions: changing in the experimental setup that impact on the outcome of the observation of various physical phenomena can also be used to learn or mitigate unwanted *domain shifts*
- Symmetries: transformations that leave the solution invariant or modify it in an equivariant way are often independent of the specific physics phenomenon

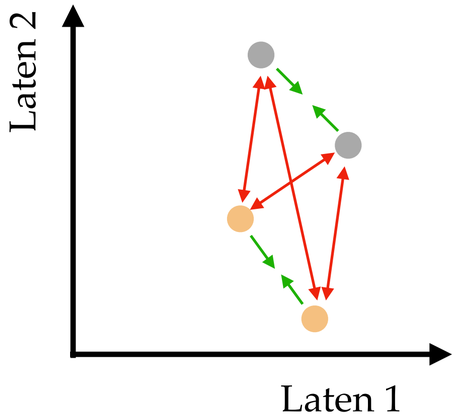
Domain knowledge

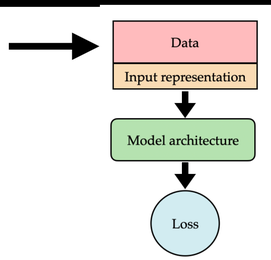


Data Augmentation

Contrastive Learning

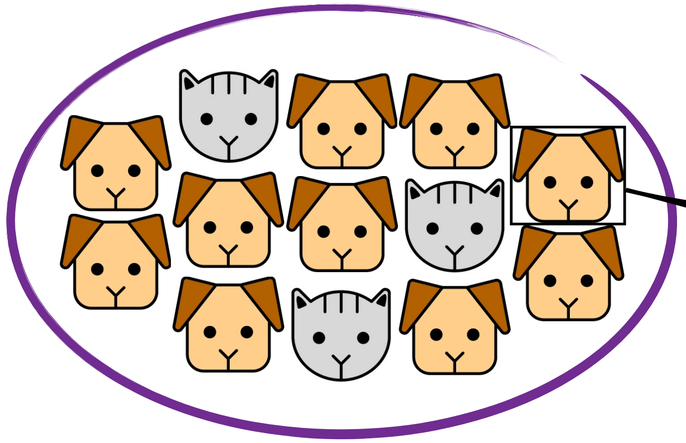
- One typical learning strategy designed to exploit augmentations is contrastive learning.
- The key idea is to organize the data in the learnt embedded space such that similar examples are clustered together while different examples are pushed apart.
- To this end we define:
  - *positive pairs* the tuples of views representing the *same* training example
  - *negative pairs* the tuples of views representing *different* training examples



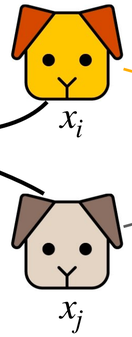


Data Augmentation

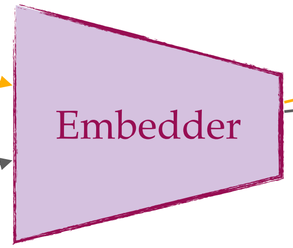
Contrastive Learning: self-supervised



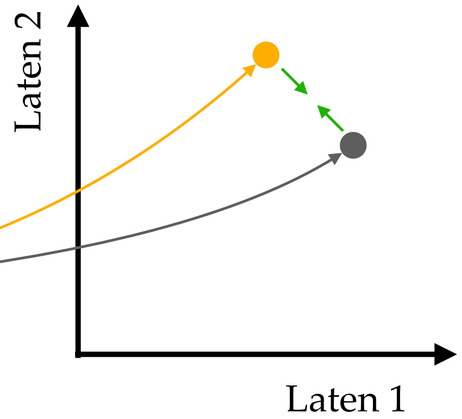
unlabelled data points



Generate two augmented *views* of the same example



$\phi_\theta : (\mathcal{X}, d_\mathcal{X}) \xrightarrow{\phi} (\mathcal{Y}, d_\mathcal{Y})$



Constraint the organization of the views in the embedding space:

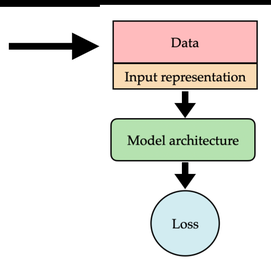
- Views of the same object are pushed together
- Views of different objects are pulled apart

SimCLR loss [2002.05709]:

$$l_{\text{SimCLR}}(x_i, x_j) = -\log \frac{e^{\text{sim}(\phi_\theta(x_i),\phi_\theta(x_j))/\tau}}{\sum_{k=1}^{N} e^{\text{sim}(\phi_\theta(x_i),\phi_\theta(x_k))/\tau}}$$

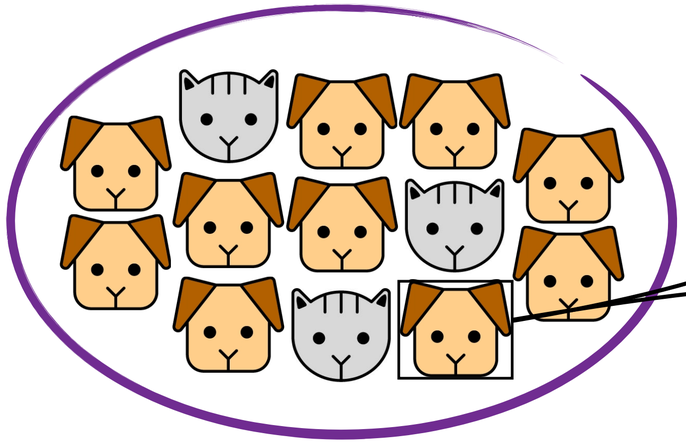
Cosine similarity:

$\text{sim}(\boldsymbol{u}, \boldsymbol{v}) = \boldsymbol{u}^\top \boldsymbol{v} / \|\boldsymbol{u}\| \|\boldsymbol{v}\|$

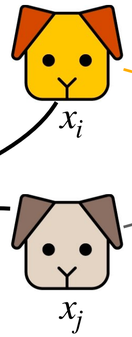


Data Augmentation

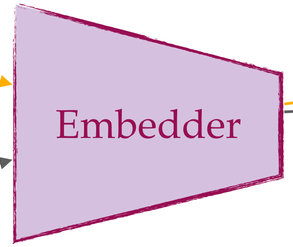
Contrastive Learning: self-supervised



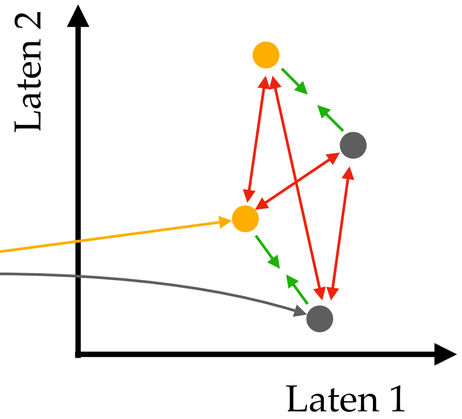
unlabelled data points



Generate two augmented *views* of the same example



$\phi_\theta : (\mathcal{X}, d_\mathcal{X}) \xrightarrow{\phi} (\mathcal{Y}, d_\mathcal{Y})$



Constraint the organization of the views in the embedding space:

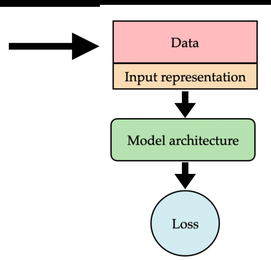
- Views of the same object are pushed together
- Views of different objects are pulled apart

SimCLR loss [2002.05709]:

$$l_{\text{SimCLR}}(x_i, x_j) = -\log \frac{e^{\text{sim}(\phi_\theta(x_i),\phi_\theta(x_j))/\tau}}{\sum_{k=1}^{N} e^{\text{sim}(\phi_\theta(x_i),\phi_\theta(x_k))/\tau}}$$

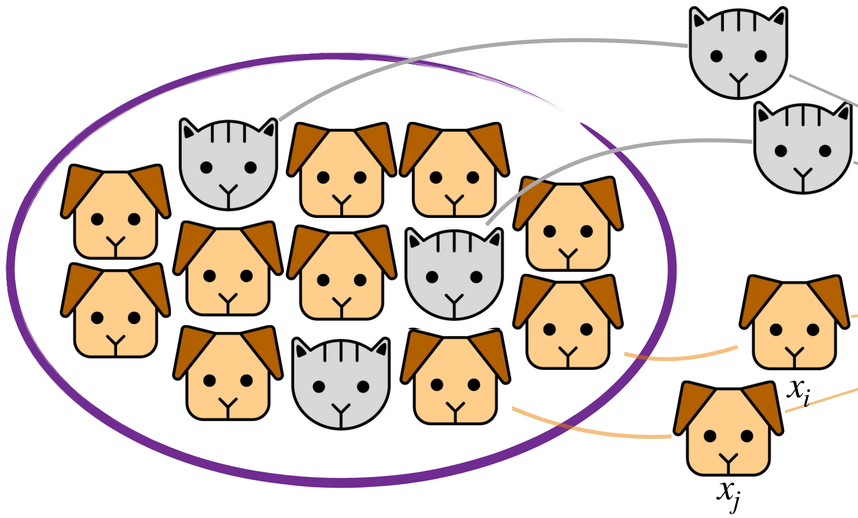
Cosine similarity:

$\text{sim}(\boldsymbol{u}, \boldsymbol{v}) = \boldsymbol{u}^\top \boldsymbol{v} / \|\boldsymbol{u}\| \|\boldsymbol{v}\|$



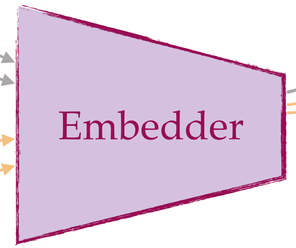
Data Augmentation

Contrastive Learning: supervised

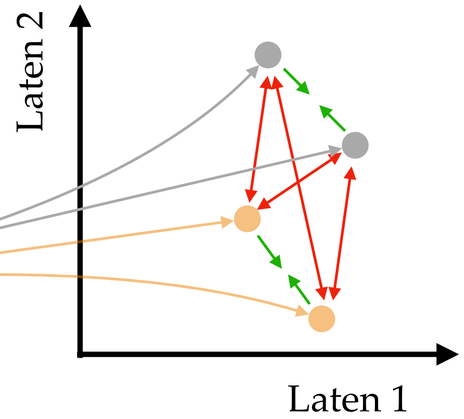


labelled data points

Generate two *views* of the same class



$\phi_\theta : (\mathcal{X}, d_\mathcal{X}) \xrightarrow{\phi} (\mathcal{Y}, d_\mathcal{Y})$



Constraint the organization of the views in the embedding space:

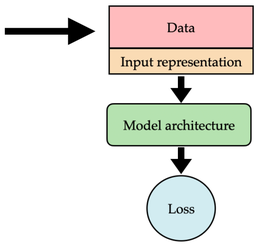
- Views of the same object are pushed together
- Views of different objects are pulled apart

SimCLR loss [2002.05709]:

$$l_{\text{SimCLR}}(x_i, x_j) = -\log \frac{e^{\text{sim}(\phi_\theta(x_i),\phi_\theta(x_j))/\tau}}{\sum_{k=1}^{N} e^{\text{sim}(\phi_\theta(x_i),\phi_\theta(x_k))/\tau}}$$

Cosine similarity:

$\text{sim}(\boldsymbol{u}, \boldsymbol{v}) = \boldsymbol{u}^\top \boldsymbol{v} / \|\boldsymbol{u}\| \|\boldsymbol{v}\|$



Data Augmentation

Hand-craft training examples can expose relevant corners of the solution space

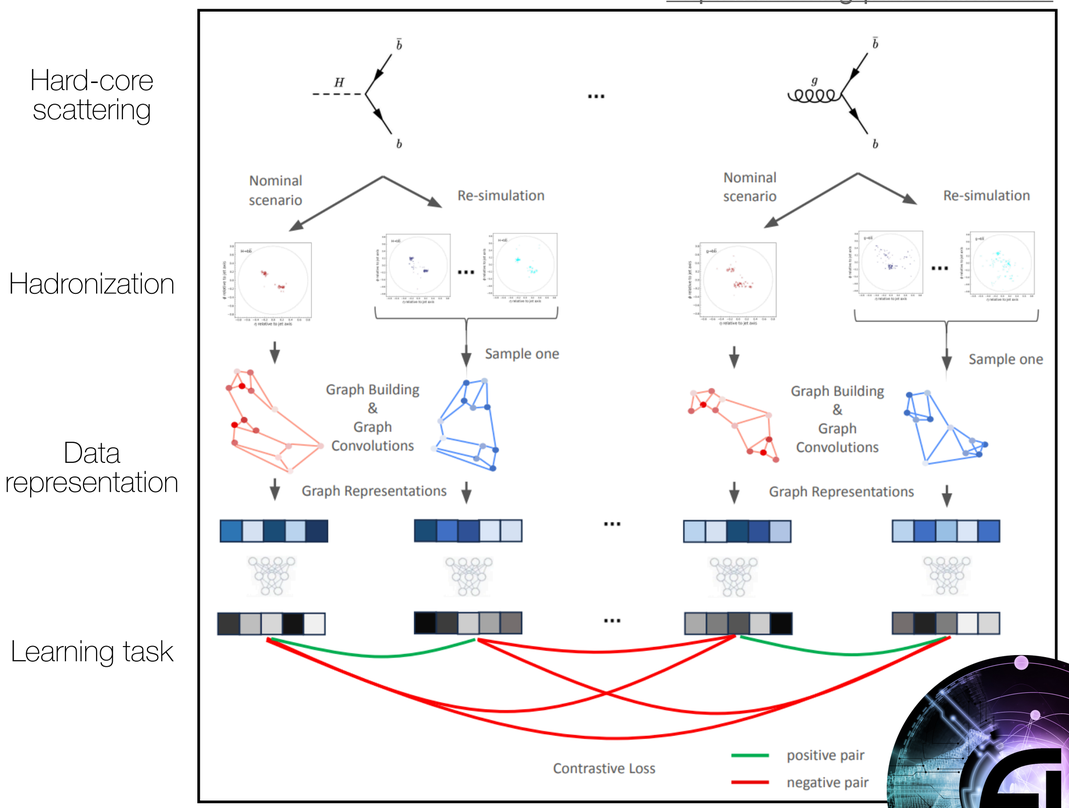
- A key distinction between DA and explicit regularizations is that DA requires more *domain knowledge* to be successful than weight decay (luckily we have plenty of it in scientific domains! :)

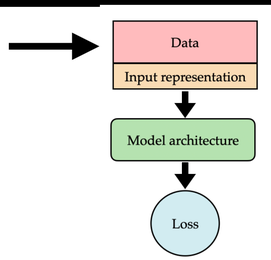
https://arxiv.org/pdf/2403.07066

Example: Jet hadronization at the LHC

Jet hadronization is a stochastic process affecting proton-proton collisions at the LHC. Jet hadronization doesn't change the nature of the hard-core scattering, therefore objective tasks aiming at inferring the physics of hard-core scattering from LHC data should be independent of the hadronization.

The same hard-core interaction can be re-simulated with different hadronization seeds to create augmentations

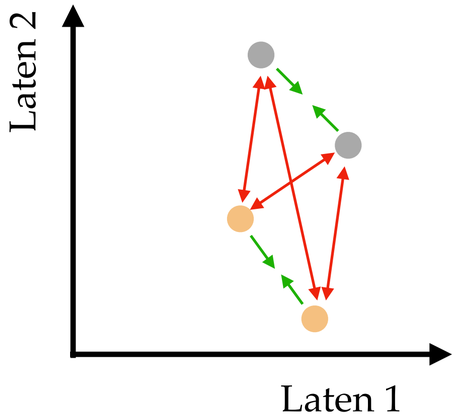




Data Augmentation

Challenges and limitations

- **Scalability**: need large samples to effectively learn inductive bias, especially if the model is large too.
- **Implicit nature**: since the bias is only implicitly induced, it is hard to control and/or guarantee a specific outcome. Induced bias might fail to generalize or cause additional unwanted features. (Loss penalties are preferable when designing them is possible.)



Physics-informed ML

Summary

- Informing ML = the art of inducing bias
- First step: recognize the properties and structures of your data and your task
- Many different ways to guide ML:
  - *Data engineering* (representation, selection, augmentations)
  - *Architecture design* (shape connections and activation, embed symmetries)
  - *Shaping the loss landscape* (penalties, regularizations)

- Assumptions = Efficiency, interpretability, robustness
- Assumptions = Constraints



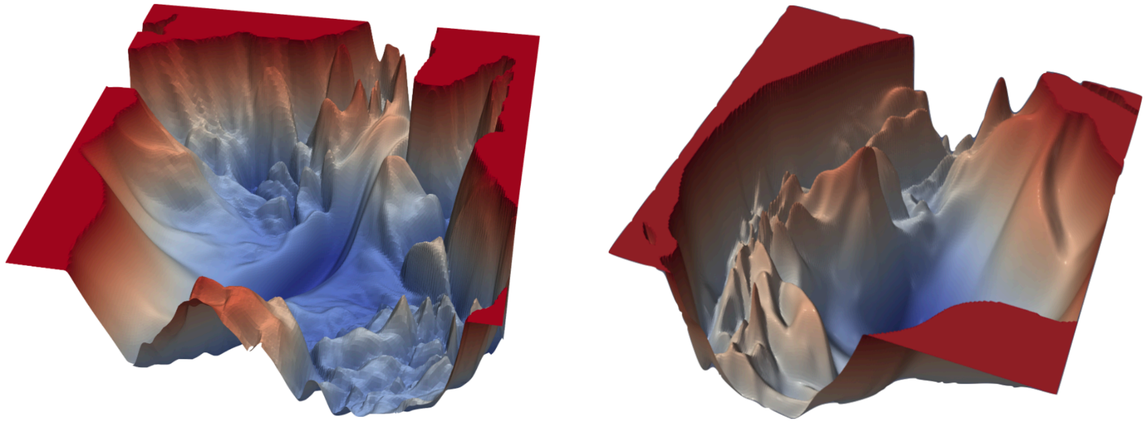
PART B: Physics-inspired AI



Why physics *inspires* successful AI models?

Imitating nature to build efficient artificial systems

Physical systems evolves to their stationary condition so they *naturally* solve an optimization problem (minimum of the energy).



Energy landscapes

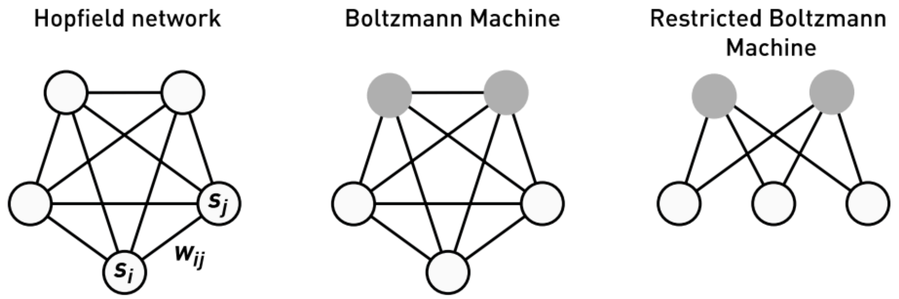
Physics-inspired AI

Imitating nature to build efficient artificial systems

Based on *statistical physics* and *thermodynamics*, the seminal work of Hopfield and Hinton in the 80s has been instrumental in shaping the field of Artificial Neural Networks (ANNs).

This work was awarded the Nobel prize in Physics in 2024.

We will dive into the history of ANNs, starting from the general framework of **Energy-based models** to learn about the first principles introduced by Hopfield and Hinton that made modern machine learning possible.



https://www.nobelprize.org/uploads/2024/11/advanced-physicsprize2024-3.pdf

Physics-inspired AI

Energy-Based Models (EBMs)

EBMs capture dependencies between observed and unobserved variables by associating a scalar *energy* as a measure of *compatibility* to each configuration of the variables (state of the system).

System:

X observed variables

Y unobserved variables

$\rightarrow E(Y, X)$ energy of the system (scalar)

A Tutorial on Energy-Based Learning. Le Cun et al



Physics-inspired AI

Energy-Based Models (EBMs)

Learning dependencies consists in finding an energy function that associates *low energies to correct configurations* of the remaining variables, and higher energies to incorrect ones.

The search for the best energy function is performed within a family of energy functions $\mathscr{E}$ indexed by a parameter θ:

$$\mathscr{E}_\theta = \{E(\theta, Y, X) \,|\, \theta \in \Theta\}$$

Prediction on Y is obtained by initializing Y as random noise and update Y via back-propagation until the minimum of $\mathscr{E}_\theta(Y, X)$ is found.

Training exploits two main tricks:

- **Contrastive learning:** the energy of *positive examples* (true pairs) is pushed down, while the energy of *negative examples* (wrong pairs) is pulled up (hard to scale in high dimensions).
- **Regularization:** add regularization terms to the energy definition to force convexity. This guarantees that the energy grows in the surrounding of the minimum.

A Tutorial on Energy-Based Learning. Le Cun et al



Physics-inspired AI

Energy-Based Models (EBMs)

Use cases:

- *Inference* (e.g. decision-making): consists in setting the value of observed variables and finding values of the remaining variables that minimize the energy

$$Y^* = \arg\min_{Y \in \mathcal{Y}} E(Y, X)$$

"Given X, which value of Y is the most compatible?"

- *Ranking:* the model is used to select multiple data points that best satisfy a given criterion

"Is Y_1 or Y_2 more compatible with this X?"

- *Detection*: the energy of the system for a given data point X under the hypothesis $\bar{Y}$ is compared to a threshold

$$s_{\bar{Y}}(X) = \begin{cases} 1 & \text{if} \quad E(\bar{Y}, X) \leq \text{thr} \\ 0 & \text{if} \quad E(\bar{Y}, X) > \text{thr} \end{cases}$$

"Is this value of Y compatible with X?"

- *Conditional density estimation*: the output of the system is not used directly to produce actions, but is given to a human decision maker or is fed to the input of another, separately built system

"What is the conditional probability distribution over $\mathcal{Y}$ given X?"

A Tutorial on Energy-Based Learning. Le Cun et al



Physics-inspired AI

Energy-Based Models (EBMs)

From energies to probabilities:

- When EBMs are used in combination with other systems, they must be calibrated!
- The only consistent way to *calibrate energies* is by turning them in probabilities.
- The simplest and most common way to obtained a normalized probability distribution from an EBM is by constructing the *Gibbs distribution*

$$P(Y|X) = \frac{e^{-\beta E(Y,X)}}{\int_{y \in \mathcal{Y}} e^{-\beta E(y,X)} dy}$$

Partition function (Z):
- Must converge to a non infinite value!
- Can be **intractable** (hard to estimate)!

probabilistic modeling comes with a **high price,** and should be avoided when the application does not require it!

A Tutorial on Energy-Based Learning. Le Cun et al



Physics-inspired AI

Energy-Based Models (EBMs)

A general framework for AI

Many existing machine learning models can be expressed simply in the framework of energy-based learning.

Some examples:

- Support Vector Machines
- Simple classification and regression models
- Recurrent NNs
- Hopfield Networks
- Boltzmann machines
- Diffusion models
- …



Physics-inspired AI

Energy-Based Models (EBMs)

Support Vector Machines (SVMs):

- SVMs are supervised learning models used for **binary classification** (and extended to multiclass or regression), which aim to find the *optimal separating hyperplane* between two classes with the *maximum margin*.
- Two hyperplanes in $\mathbb{R}^D$ are defined such that:
  $\mathrm{w}^T x - b = +1$ most offending incorrect answer for class $y = +1$
  $\mathrm{w}^T x - b = -1$ most offending incorrect answer for class $y = -1$

- The distance between the two hyperplanes *(margin)* is $\frac{2}{\|\mathrm{w}\|}$.
- To solve the classification problem while maximizing the margin the **Hinge loss** is used:

$$L = \underbrace{\frac{1}{2}\|w\|^2}_{\text{Regularizer}} + C \underbrace{\sum_{i=1}^{N} \max\left(0, 1 - y_i(w^\top x_i + b)\right)}_{L_{\text{hinge}}(x_i, y_i)}$$

- This is a EBM with $E(y, x) = -y_i(\mathrm{w}^T x - b)$

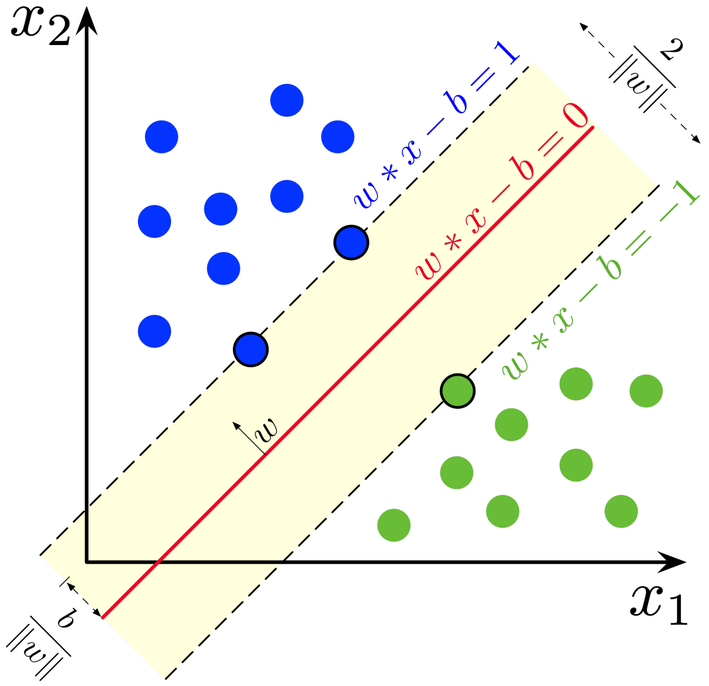
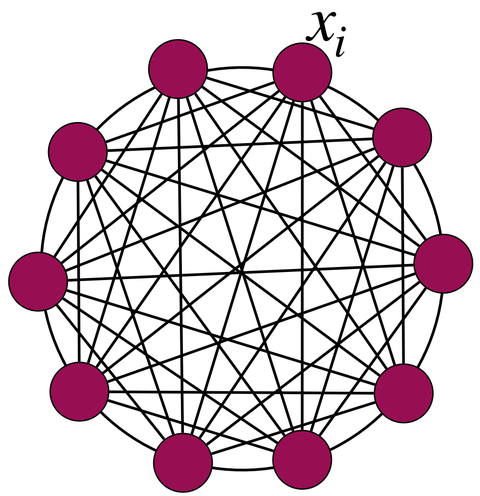


Image from Wikipedia: Support Vector Machines

Physics-inspired AI

Energy-Based Models (EBMs)



Hopfield Networks (HNs):

- HNs are recurrent neural networks with *binary* neurons and *symmetric bidirectional* connections.
- The energy function for a system $X = (x_1, \ldots, x_n)$ of n neurons is defined as:

$$E(X) = -\frac{1}{2}\sum_{i,j} w_{ij} x_i x_j + \sum_i b_i x_i$$

Where:
$x_i \in \{-1, +1\}$
$w_{ij} = w_{ji}$ $(w_{ii} = 0)$
b_i bias term

Hopfield, J. J. Neural networks and physical systems with emergent collective computational abilities. *PNAS* **79**, 2554–2558 (1982).

Physics-inspired AI

Energy-Based Models (EBMs)

Hopfield Networks (HNs):

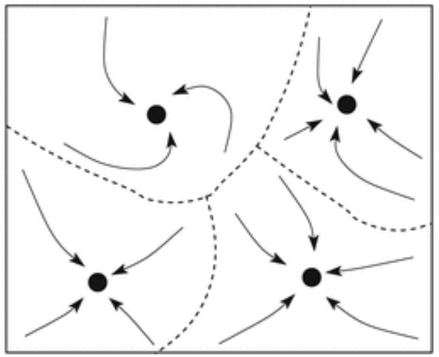
- HNs are used as a model for *associative memory*:
  - The model **stores patterns** $\{\xi^{(1)}, \xi^{(2)}, \ldots, \xi^{(M)}\}$, with $\xi^{(m)} \in \{-1, +1\}^n$ such that they become **stable attractors** (local minima) of the network's energy function. The patterns are stored *in the weights* values following the **Hebbian learning rule** (one shot learning):

$$w_{ij} = \frac{1}{M} \sum_{\mu=1}^{M} \xi_i^{(\mu)} \xi_j^{(\mu)}, \quad \text{with } w_{ii} = 0$$

"Neurons that fire together, wire together."

  - **Memory retrieval:** The stored patterns can then be retrieved starting from noisy or partial inputs following the gradient of the energy:

$$x_i \leftarrow \text{sign}\left(\sum_j w_{ij} x_j - b_i \right)$$



Physics-inspired AI

Energy-Based Models (EBMs)

Hopfield Networks (HNs):

- HNs are used as a model for *associative memory*:
  - The model **stores patterns** $\{\xi^{(1)}, \xi^{(2)}, \ldots, \xi^{(M)}\}$, with $\xi^{(m)} \in \{-1, +1\}^n$ such that they become **stable attractors** (local minima) of the network's energy function. The patterns are stored in the weights values following the **Hebbian learning rule** (one shot learning):

$$w_{ij} = \frac{1}{M} \sum_{\mu=1}^{M} \xi_i^{(\mu)} \xi_j^{(\mu)}, \quad \text{with } w_{ii} = 0$$

"Neurons that fire together, wire together."

  - **Memory retrieval:** The stored patterns can then be retrieved starting from noisy or partial inputs following the gradient of the energy:

$$x_i \leftarrow \text{sign}\left(\sum_j w_{ij} x_j - b_i \right)$$

  - **Storage capacity**: the maximal number of memory patterns that can be successfully retrieved from a HN scales linearly with the number of feature neurons

$$M \propto n$$

Very limited for realistic applications!



Physics-inspired AI

Energy-Based Models (EBMs)

Modern Hopfield Networks (Dense Associative Memory[1]):

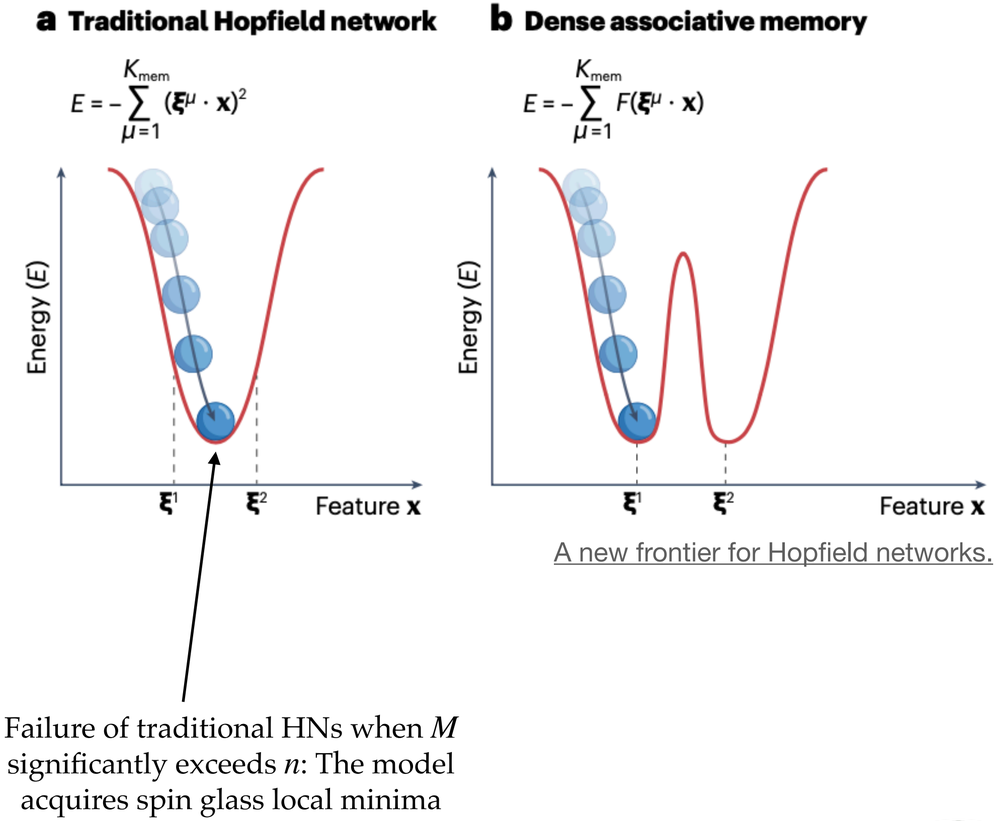
- Using a *rapidly growing activation function* $F(\cdot)$ in the energy definition allow to retrieve *superlinear* scaling of capacity

$$E(X) = -\sum_{\mu=1}^{M} F(\xi^{(\mu)} \cdot x_i)$$

- Connection with Transformer models[2]:

  If $F(\cdot) = \text{SoftMax}(\cdot)$ modern HN reduce to the *Attention Mechanism*.

  This finding enables investigation of pre-trained transformer models as a physical system, exploiting theoretical concepts commonly used in statistical physics.

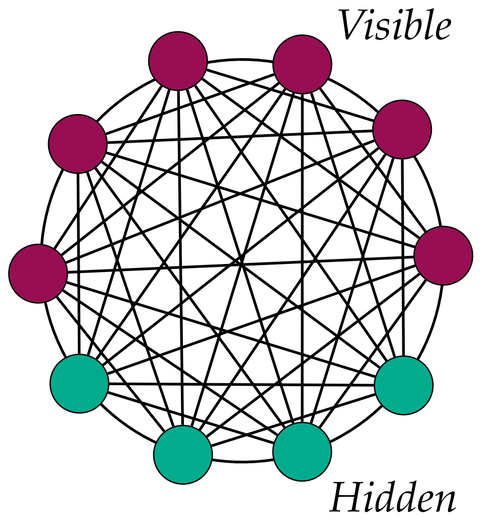


Failure of traditional HNs when M significantly exceeds n: The model acquires spin glass local minima

[1]Krotov, D. & Hopfield, J. J. Dense associative memory for pattern recognition. In *Advances in Neural Information Processing Systems 29* (NIPS, 2016).
[2]Ramsauer, H. et al. Hopfield networks is all you need. In *International Conference on Learning Representations* (ICLR, 2021).

Physics-inspired AI

Energy-Based Models (EBMs)



Boltzmann Machines (BMs):

- BMs are energy-based *generative models* of the joint distribution over visible and hidden variables

  n binary *visible* units: $V \in \{0,1\}^n$ $\quad \rightarrow X = (V, H) \in \{0,1\}^{n+m}$
  m binary *hidden* units: $H \in \{0,1\}^m$

- The energy of the system is equivalent to that of HNs:

$$E_w(X) = -\frac{1}{2}\sum_{i,j} w_{ij} x_i x_j + \sum_i b_i x_i$$

  $w_{ij} = w_{ji}$ $(w_{ii} = 0)$
  b_i bias term

- The *visible* units are the interface between the network and the *environment* (e.g. the data).
- The *hidden* units model the *underlying structure* of the environment that cannot be captured by pairwise connections between visible units.
- As a result, the whole network develops an internal model which captures the regularities of the environment.

D. Ackley, G. Hinton A learning algorithm for Boltzmann machines, *Cognitive Science*, Volume 9, Issue 1 (1985)

Physics-inspired AI

Energy-Based Models (EBMs)

Boltzmann Machines (BMs):

The system is treated as a *system of particles in a thermal bath*.

- At temperature T, the k^{th} particle is given the state $+1$ with probability:

$$P(x_k = 1) = \frac{1}{1 + e^{-\Delta E_k / T}}$$

ΔE_k is the energy gap between the two states of the system in which the k^{th} particle is flipped

$$P(x_k = 1; T)$$

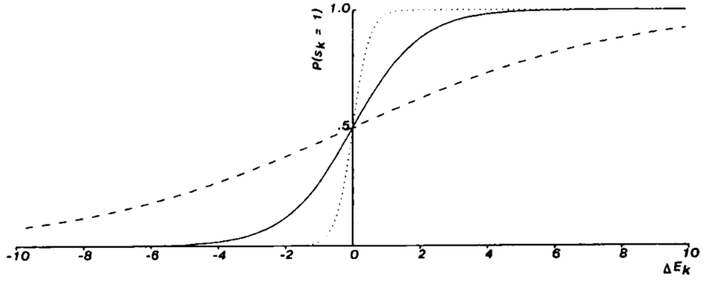


Figure 1. Eq. (5) at $T=1.0$ (*solid*), $T=4.0$ (*dashed*), and $T=0.25$ (*dotted*).

A LearningAlgorithm for Boltzmann Machines

- When thermal equilibrium is reached the probability to find the system in any given state follows the *Boltzmann distribution*.

$$P_w(X_\alpha) = \frac{e^{-E_w(X_\alpha)/T}}{Z}$$

Physics-inspired AI

Energy-Based Models (EBMs)

Boltzmann Machines (BMs):

The model builds on principles of statistical mechanics and thermodynamics to extract the joint *distribution* over the hidden and visible units:

- **Positive phase:** the network is brought to thermal equilibrium with the visible units constrained by the environment

$$P_w^+(X_\alpha) = \frac{e^{-E_w(X_\alpha)/T}}{Z}$$

- **Negative phase:** the network is brought to thermal equilibrium while free

$$P_w^-(X_\alpha) = \frac{e^{-E_w(X_\alpha)/T}}{Z}$$

- The weights of the networks are updated following gradient descent on the **KL divergence**:

$$L = \sum_\alpha P^+(V_\alpha) \log \frac{P^+(V_\alpha)}{P^-(V_\alpha)} \quad \rightarrow \quad \Delta w_{ij} = \epsilon(p_{ij}^+ - p_{ij}^-)$$

p_{ij}^+: average probability of two units both being in the **on** state when the environment is clamping the states of the visible units

p_{ij}^-: average probability of two units both being in the **on** state when the network is running freely



Physics-inspired AI

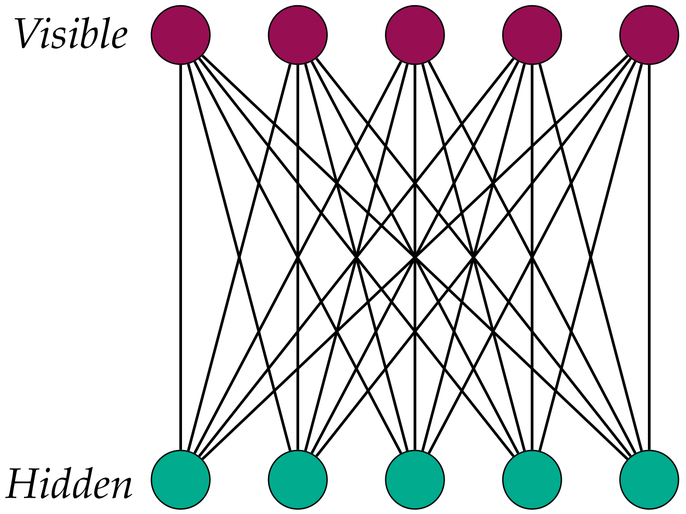
Energy-Based Models (EBMs)

Restricted Boltzmann Machines (RBMs):

The architecture is a **bipartite graph**:

- No connections within visible or within hidden units
- Only **between layers** (restricted!)

- Feed forward rather than recurrent structure!
- Faster to train (each visible unit and hidden unit can be updated simultaneously)



Physics-inspired AI

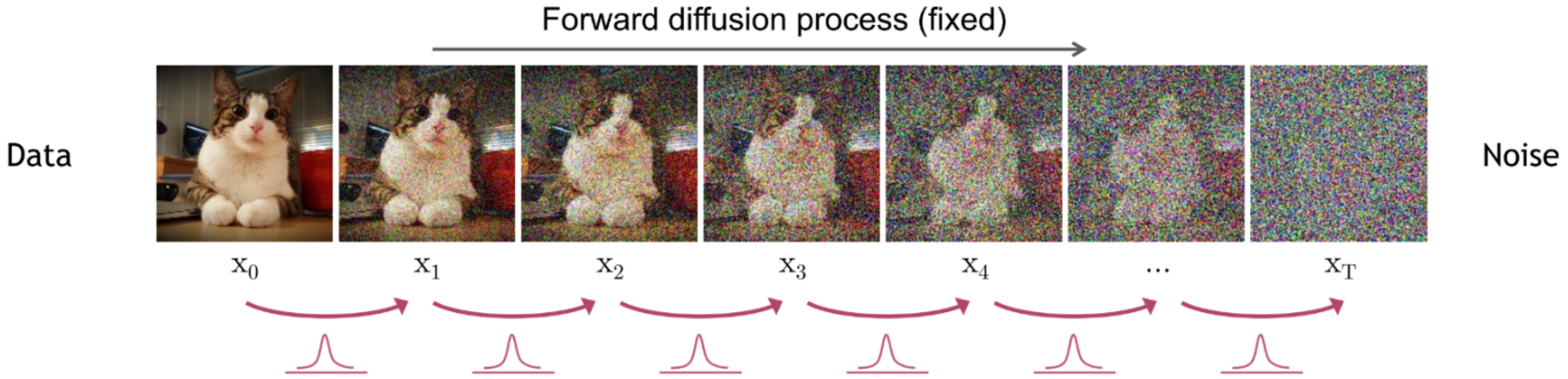
Energy-Based Models (EBMs)

Diffusion Models:

- Diffusion models are inspired by *non-equilibrium thermodynamics*. They define a Markov chain of diffusion steps to add noise to data and then learn to reverse the diffusion process to generate samples from noise.
- *Forward process (diffusion)*: At $t = 0$, the data points follow the distribution q, which is not in equilibrium. Equilibrium is reached by evolving the system via diffusion, e.g. adding gaussian noise to the initial state:

$$q(x_t \mid x_{t-1}) = \mathcal{N}(x_t; \sqrt{1-\beta_t} x_{t-1}, \beta_t \mathbb{I}) \qquad \beta_t \in [0,1]$$

At $T \gg 0$ the distribution approximates an isotropic normal distribution.



More in last year lecture: https://github.com/glouppe/iaifi-summer-school-2024

Physics-inspired AI

Energy-Based Models (EBMs)

Diffusion Models:

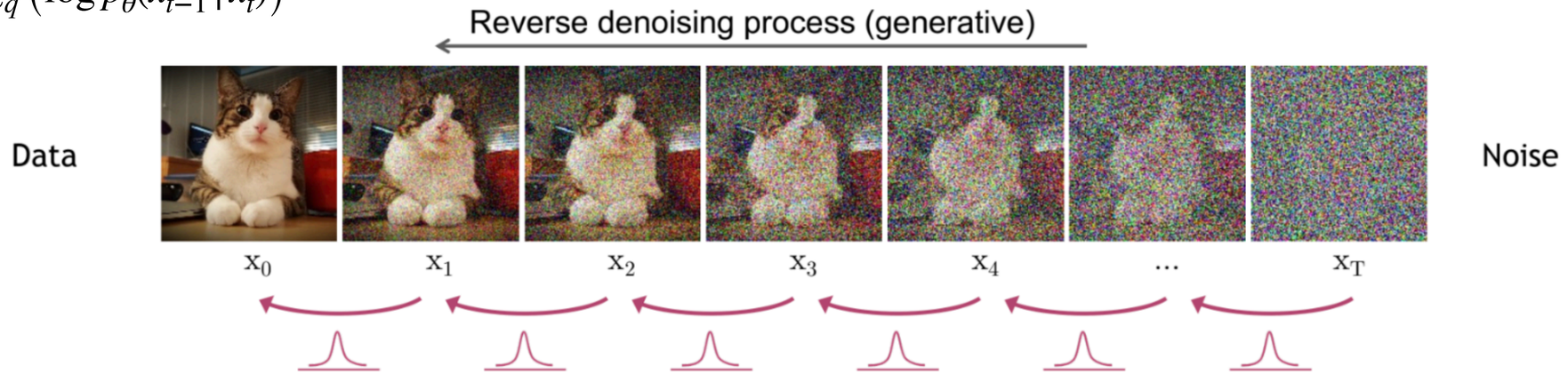
- Diffusion models are inspired by *non-equilibrium thermodynamics*. They define a Markov chain of diffusion steps to add noise to data and then learn to reverse the diffusion process to generate samples from noise.

- *Reverse process (denoising)*: learn to remove noise at each step by modeling the denoising process as a multivariate gaussian with **learnable mean (μ_θ) and covariance matrix (Σ_θ)**:

$$p_\theta(x_{t-1} \mid x_t) = \mathcal{N}\left(x_{t-1}; \mu_\theta(x_t, t), \Sigma_\theta(x_t, t)\right)$$

The learning process involves maximizing the likelihood of the data by minimizing the loss:

$$L(\theta) = -\sum_{t=1}^{T} \mathbb{E}_{x_{t-1}, x_t \sim q}\left(\log p_\theta(x_{t-1} \mid x_t)\right)$$



More in last year lecture: https://github.com/glouppe/iaifi-summer-school-2024

Physics-inspired AI

Optimizers as artificial dynamical systems

- Generalizing the energy-based concept, training any neural network can be viewed as *evolving a system* in **parameter space** to minimize a loss function $L(\theta)$.
- This is analogous to a physical system evolving toward a state of minimum energy.

Some examples of common optimizers:

- *Gradient descent* updates can be interpreted as a the dynamic of a particle in a potential energy landscape:

$$\gamma\dot{\theta} + \nabla_\theta L(\theta) = 0 \quad \rightarrow \quad \theta_{t+1} = \theta_t - \eta\,\nabla_\theta L(\theta)$$

- *Momentum* updates can be interpretedas a the dynamic of a particle in a potential energy landscape with *inertia*:

$$m\ddot{\theta} + \gamma\dot{\theta} + \nabla_\theta L(\theta) = 0 \quad \rightarrow \quad \theta_{t+1} = \theta_t + v_{t+1}, \quad v_{t+1} = \mu v_t - \eta\,\nabla_\theta L(\theta)$$

- *RMSProp* adapts the learning rate individually for each parameter by *normalizing* with the recent magnitude of its own gradients. It behaves like *anisotropic friction*, each parameter dimension has its own effective viscosity.

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E[\nabla_\theta L(\theta_t)] + \varepsilon}} \cdot \nabla_\theta L(\theta_t)$$

$$E[\nabla_\theta L(\theta_t)^2]_t = \gamma E[\nabla_\theta L(\theta_t)^2]_{t-1} + (1-\gamma)\,\nabla_\theta L(\theta_t)^2$$



Physics-inspired AI

Improve learning stability with annealing

The algorithm mimics *annealing in metallurgy*, where a material is slowly cooled to minimize defects and reach a low-energy crystalline state.

Stochasticity is introduced in the learning dynamics to help the system **escape shallow local minima** (common in non-convex loss functions) and eventually settle in deeper, more optimal configurations.

- The probability P of accepting a parameter's update θ' is controlled by a temperature T and the energy difference between the two states: $\Delta L(\theta) = L(\theta') - L(\theta)$.
- The following properties should be satisfied:
  - If $\Delta L(\theta) < 0$: $\quad P(\Delta L(\theta), T) \overset{T \to 0}{\to} 1$
  - $P(\Delta L(\theta), T)$ decreases as $\Delta L(\theta)$ increases

Intuition: At *high temperatures*, the network perform a search of the coarse overall structure of the space of global states, and will find a good minimum at that *coarse level*. At *lower temperatures*, it will begin to respond to the *finer structure* in the loss landscape and will find one of the better minima within the coarse-scale minimum it discovered at high temperature.

