PASM 2026 Lecture 1

AI and Symmetry

AI / Machine learning.

- Learning algorithm
- Data and representation

Science:

- Equation & logic
- Physical Law
- Symmetry & Constraints

In the fields of LLM and image processing, two things are driving the fields.

- Universal Approximation Theorem

  a feed-forward neural network with a single hidden layer and a finite number of neurons can approximate any continuous function, provided the activation function is non-constant, bounded, and continuous.

- Scaling Law.

  The performance of a model follows a power-law curve, as the model parameters and data size scale up.

AI for Science (Physics, Chemistry):

Reasoning (2025-)

↙ ↖

Generative Model → surrogate model.

(2024-) (2017-)

In an agentic AI framework,

human - AI agent - skill - tool - task.

For a machine learning model for scientific problems, neural network-based

what's the role of symmetry?

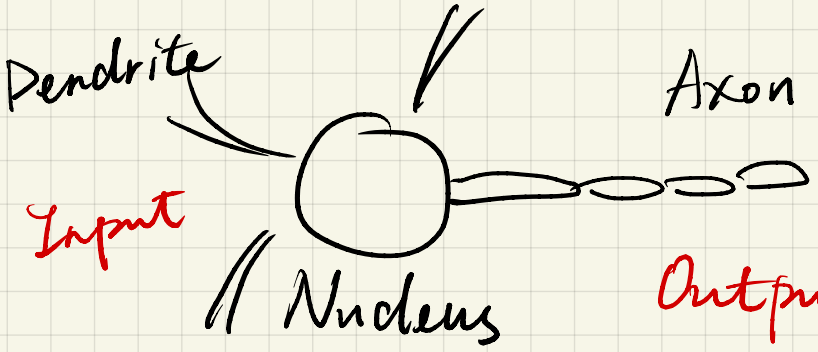
Is it possible for a NN to obey symmetry exactly?

If so, how?

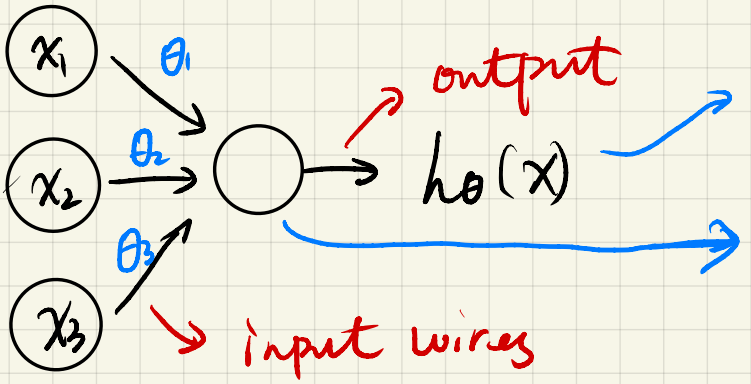
Basic learning algorithm – neural network.

An algorithm that trys to mimic the brain:

Neuron in the brain.



Neuron model: Logistic unit



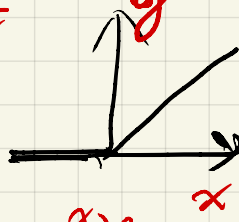
add nonlinearity ↑

e.g.: sigmoid

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

ReLU

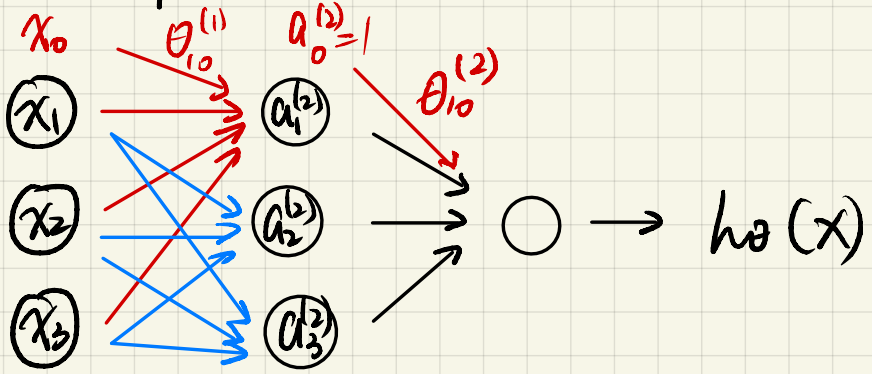
$$g(x) = \begin{cases} x & x > 0 \\ 0 & x \le 0 \end{cases}$$



$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \qquad \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}$$

↓ input feature ↓ weights

A simple neural network



Input layer | Hidden layer | Output layer

$a_i^{(j)}$ = "activation" of unit i in layer j.

$\Theta^{(j)}$ = weight matrix for function mapping from layer j to layer $j+1$

Forward propogation:

$$a_1^{(2)} = \sigma(\Theta_{10}^{(1)} x_0 + \Theta_{11}^{(1)} x_1 + \Theta_{12}^{(1)} x_2 + \Theta_{13}^{(1)} x_3) \rightarrow z_1^{(2)}$$

$$a_2^{(2)} = \sigma(\Theta_{20}^{(1)} x_0 + \Theta_{21}^{(1)} x_1 + \Theta_{22}^{(1)} x_2 + \Theta_{23}^{(1)} x_3) \rightarrow z_2^{(2)}$$

$$a_3^{(2)} = \sigma(\Theta_{30}^{(1)} x_0 + \Theta_{31}^{(1)} x_1 + \Theta_{32}^{(1)} x_2 + \Theta_{33}^{(1)} x_3) \rightarrow z_3^{(2)}$$

$$h_\Theta(x) = \sigma(\Theta_{10}^{(2)} a_0^{(2)} + \Theta_{11}^{(2)} a_1^{(2)} + \Theta_{12}^{(2)} a_2^{(2)} + \Theta_{13}^{(2)} a_3^{(2)}).$$

Vector and matrix form:

$$z^{(2)} = \begin{bmatrix} z_1^{(2)} \\ z_2^{(2)} \\ z_3^{(2)} \end{bmatrix} \qquad x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_n \end{bmatrix}$$

vector ← matrix → vector

$$\Rightarrow z^{(2)} = \Theta^{(1)} x \quad (\text{or } \Theta^{(1)} a^{(1)})$$

$$a^{(2)} = g(z^{(2)})$$

So, a forward propogation is established.

$$x \leftrightarrow a^{(1)} \xrightarrow[\Theta^{(1)}]{} z^{(2)} \xrightarrow[g(z^{(2)})]{} a^{(2)} \xrightarrow[\Theta^{(2)}]{} z^{(3)} \xrightarrow[g(z^{(3)})]{} a^{(3)} \leftrightarrow h_\Theta(x)$$

Now, define a cost function to evaluate the prediction (relative to the ground truth) ↑

$$J(\Theta) = f(h(\Theta), \underset{\hat{}}{y})$$

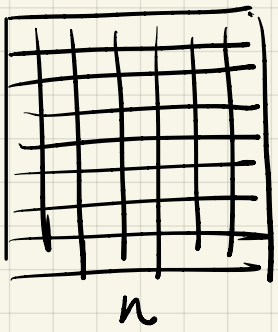
- classification: cross entropy
- regression: mean absolute error squared

Θ will be updated through the so-called backpropogation.

└ a "learning" process.

The requirement is every step is differentiable.

- The representation is learned
- Information flow is affected by the weight matrix, which defines the "channels" connecting two layers of NN.



An image is represented by a $(n \times n)$ pixel grid, with n_c color information in each pixel. (RGB: 3; Grey: 1)

The input data dimension: $n \times n \times n_c$.

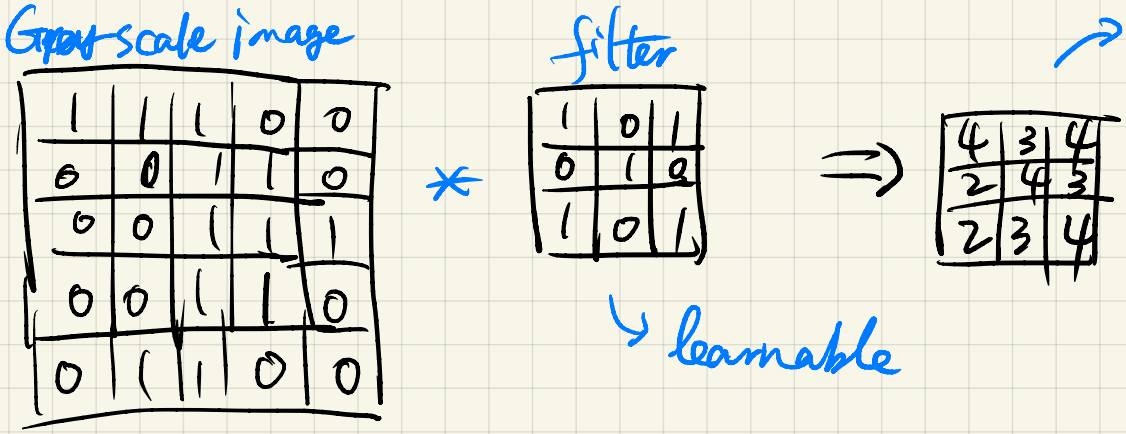
An important concept: convolution.

Convolution in CNN is a mathematical operation that slides a set of small filters (or kernel) over the input image

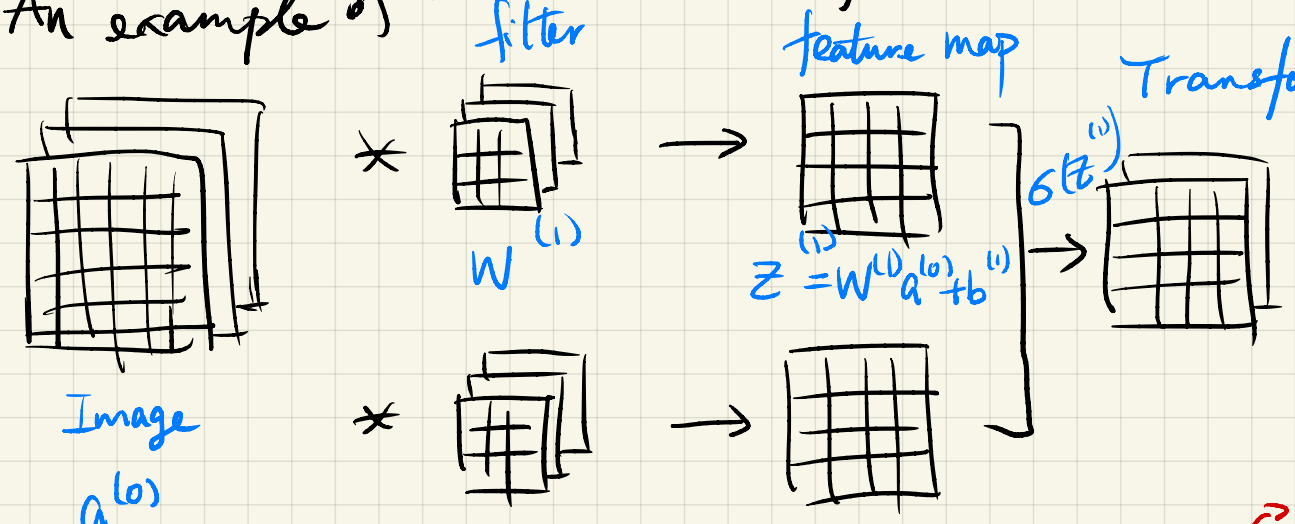
Two steps:

① The filter: a small matrix with learnable weights

② Sliding and calculating the dot product



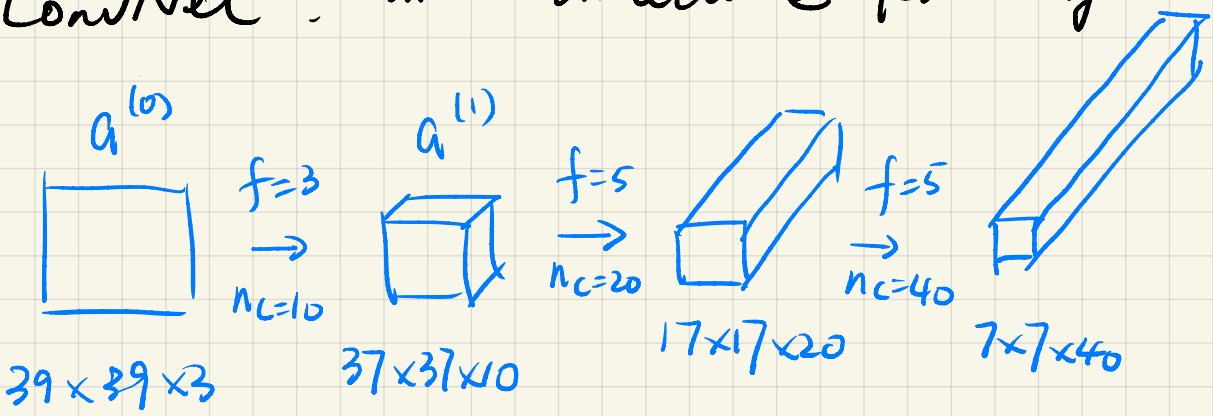
An example of a convolution layer

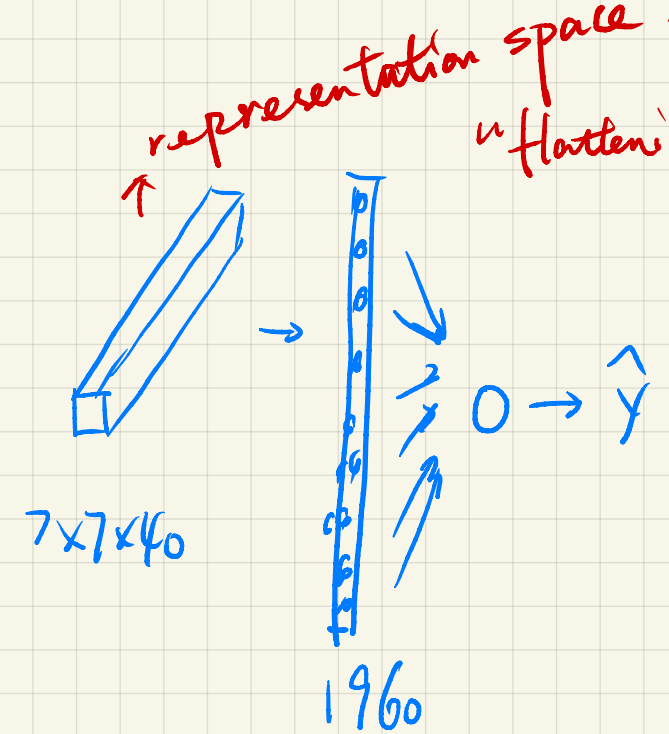


$n \times n \times n_c$ | $f \times f \times n_c$ | $(n-f+1) \times (n-f+1) \times n_c'$ ← # of filters.

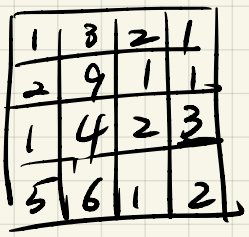
$6 \times 6 \times 3$ | $3 \times 3 \times 3$ | $4 \times 4 \times 2$

ConvNet: an architecture for image and beyond



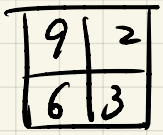


Pooling:

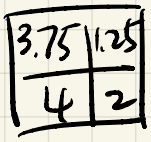


f=2

S=2 stride

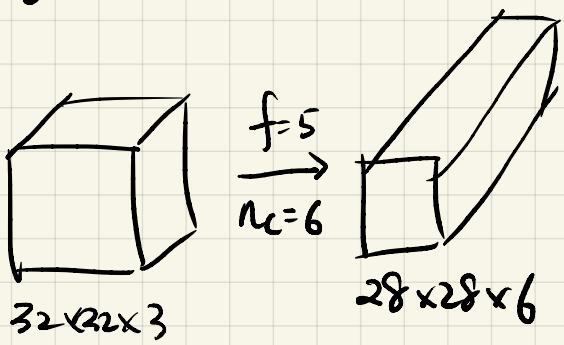


Maxpooling.



Average pooling.

Why Convolution?

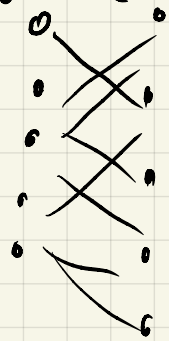


How many learnable parameters in those filters?

5x5=25 25+1=26 26x6=144!

What if we use a fully connected network?

3072 x 4704 = 14M!

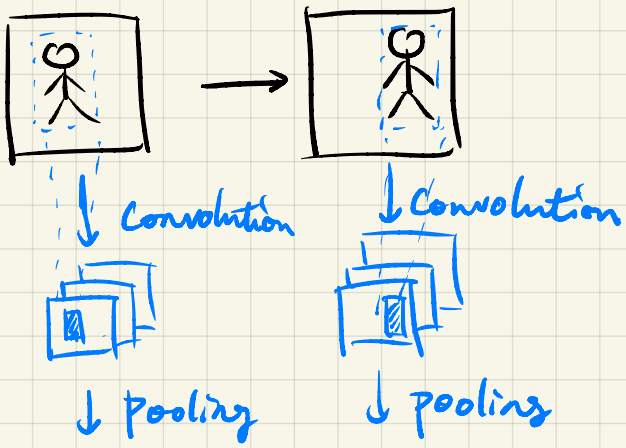


other advantages:

- Parameter sharing
- Sparcity of connections.

Translation invariance / equivariance:

Apply a translation to the image:



same output $\hat{y}$!

This is the first example how a NN is designed to obey invariance

The convolution itself is translation equivariant.

After pooling,

the final output is translation invariant.

If we translate the input image h^{in} by an vector $\vec{t}$,

then the output of each downstream layer

will transform the same way!

A invariant network are equivariant on the inside!

Let's revisit the whole picture briefly:

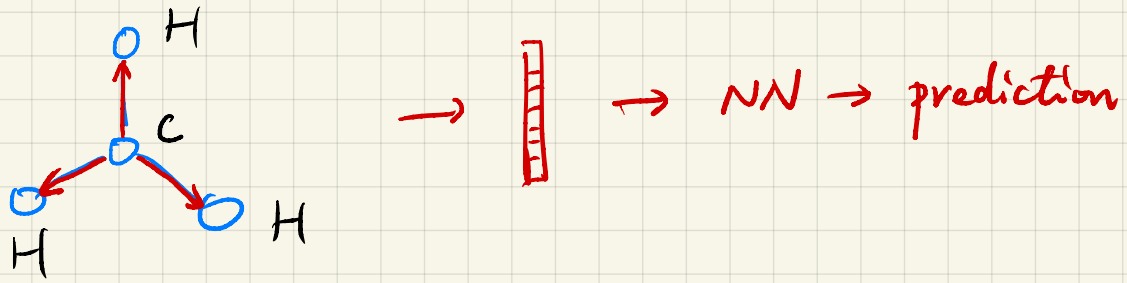
NN: linear layer + nonlinear activation.

Deep NN: representation learned in hidden layers.

CNN: convolution &

translation invariance. (symmetry!)

Let's take a look at representations for atomic systems (molecules and solids).



↓ rotation (R)

$\{i\} \to R\{i\}$ for any $R \in \mathbb{R}^{3\times3}$

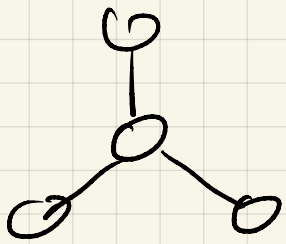
We expect the representation/descriptor to be the same!

⇒ Invariant descriptors { Behler-Parrinello
Coulomb matrix
SOAP
bispectral coefficients.

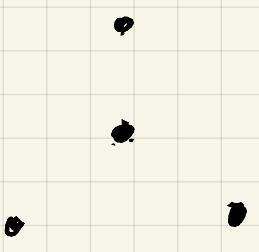
⇒ NN

The potential drawback is that they factor out any symmetry group operation. This is contrary to the NN philosophy of learning increasingly complex (or abstract) representations of the data layer by layer.

Now let's come back to the atomic system:



To describe a atomic system,
it's "point cloud + amendment".



+ bonding
elemental
orbital
spin ...

To handle point cloud in a ML framework,
+ bonding
we use graphs. Graph-based ML is part of geometric deep learning, which extends ML to non-Euclidean domains like graphs, point clouds, 3D meshes. by building models that respect the intrinsic symmetries and geometric structure of the data.
⇒ efficiency & accuracy.