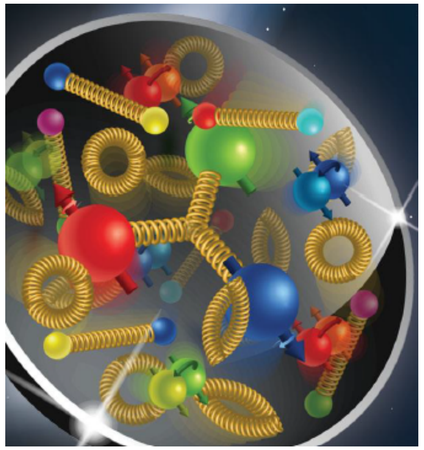
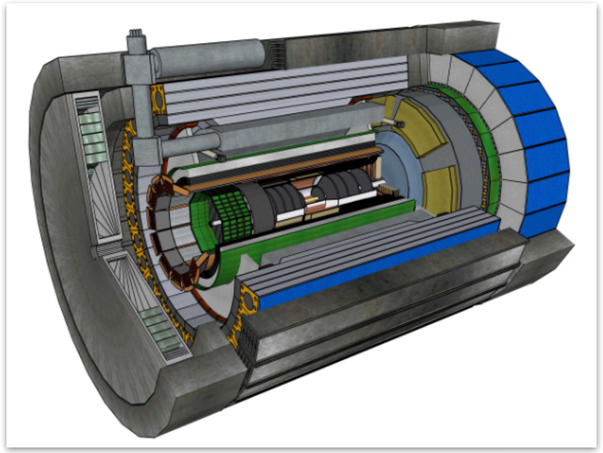
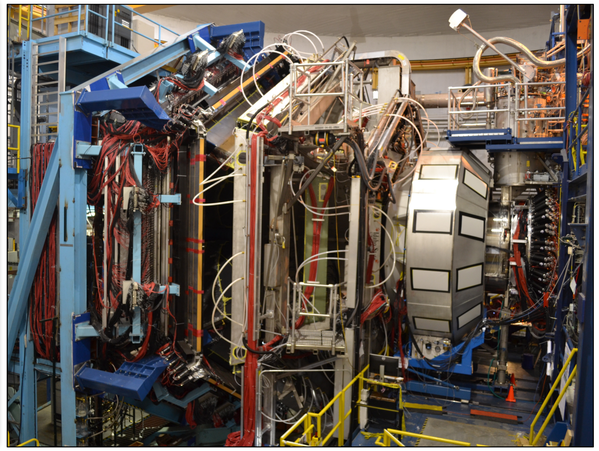
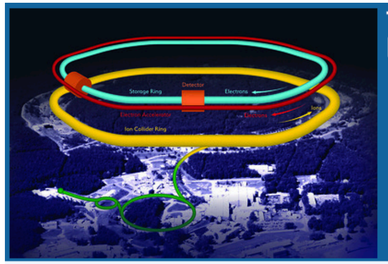




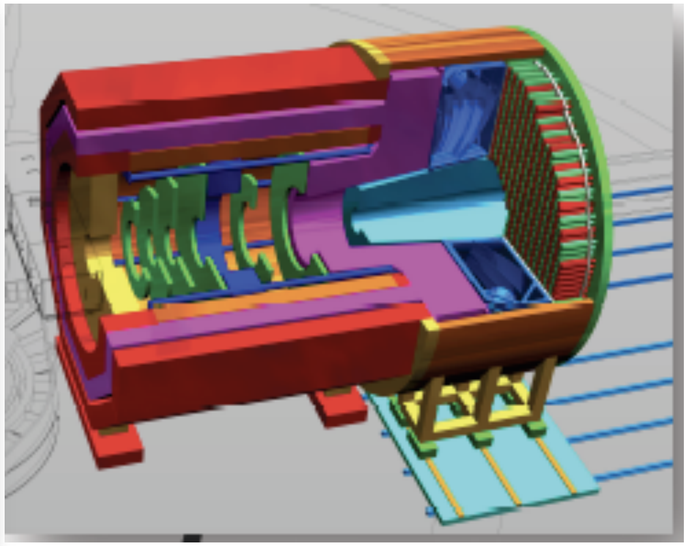




The 2024 CFNS Summer School on the Physics of the Electron-Ion Collider

3–14 Jun 2024
CFNS

Enter your search term



AI-supported methods for Real-time data analysis

Part II - AI/ML fundamentals

M.Battaglieri (INFN)

Jefferson Lab clas e-dLab12 Brookhaven National Laboratory ePIC INFN

Outline

Part II

AI/ML fundamentals

- AI basics
- AI/ML algorithms
- Linear regression
- Minimization
- Extrapolation vs. interpolation
- Neural Nets
- An example: GAN to extract physics
- Autoencoder

Machine/Deep Learning

Typical problem:

- **Data set** with independent (Xi) and dependent (Yj) variables
- **The model,** function of variables Xi, and some parameters Pk
- **The cost (loss) function** to judge how the model performs on observed Z(Xi,Yi) varying parameters Pk

How it works:

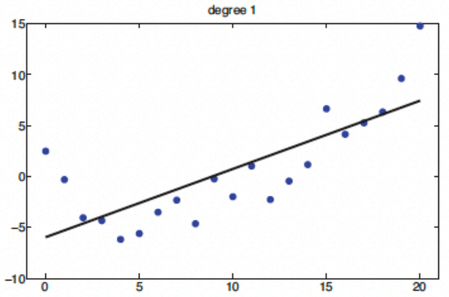
- Split (randomly) the data set into 2 (independent) subsets **training** and **test**
  (… better in 3: a small part of the test data set can be used for **validation** during trainig)
- The model learns from the training data set
- The performance of the model (changing Pk) is evaluated using the cost (loss) function
- When the training is concluded, apply the resulting parameters to the the test data set

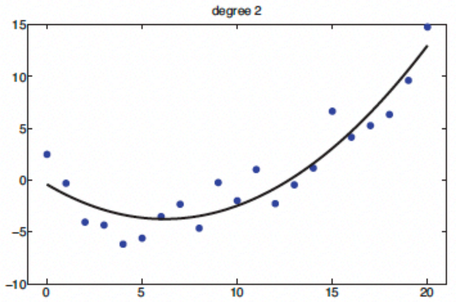
Some caveats

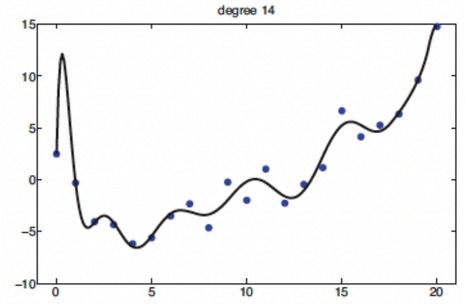
- Fitting is not predicting (this is why we want to use a validation set during training)
- Using a complex model may result in overfitting
- Better to use the simplest model
- Not different from a traditional fit …

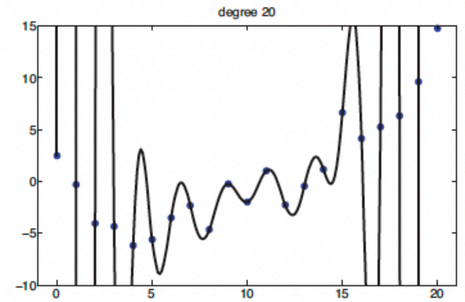
Model selection

- No free lunch theorem: there is no universally best model (All models are wrong, but some models are useful. — George Box)
- Cross-validation to empirically select the best model for the specific problem









Data science is not an 'exact science', data scientist's experience is priceless …

(Machine) learning algorithms

How do we learn from data?

- Machine learning systems process **data sets** made by **examples** (a vector)
- Examples are a collection of **features** (element of the vector)
- The data set is divided into **batches**
- Every run of the AI/ML algorithm over the training data set is called an **epoch**

In a HEP/NP scattering data set:
- example = data entries (events)
- features = 4-momentum components

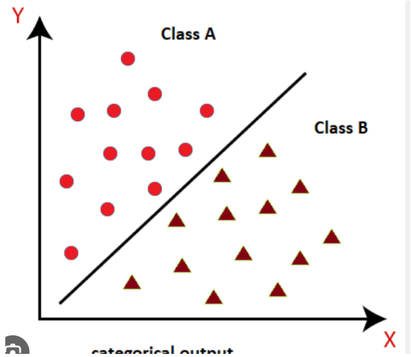
  - **Performance**: the accuracy of the model is evaluated by measuring the error rate. Values can be 0/1 or *log(Probability)*
  - **Unsupervised**: the algorithm finds the underlying law by itself
  - **Supervised**: an instructor teaches the algorithm what to do
  - **Reinforced learning**: the algorithm may change in time interacting with the environment

An ML algorithm is a transformation that connects a set of features with an output variable(s)

Classification

- ML algorithm is a function $f: R^n \rightarrow \{1, \ldots, k\}$

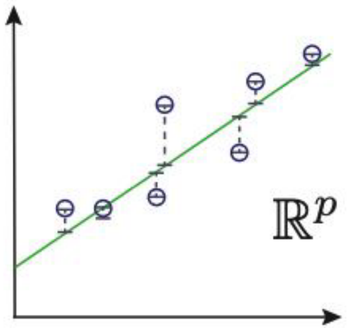
In a HEP/NP scattering data set:
- separate signal from background



Regression

- ML algorithm takes a vector x in R^n as input and produces a scalar y in R as output $f: R^n \rightarrow R$

In a HEP/NP scattering data set:
- fitting a model to data



(Machine) learning algorithms

Linear regression

- The output is a linear function of the input
- w (weights) = parameters
- P (Performance) = Mean Square Errors
- The prediction matches the test value when the (Euclidean) distance between the predicted and the test sample decreases to zero
- The ML algorithm should change w to minimize the MSE
- Minimization can be done analytically or iteratively
- A generalization of linear regression introduces an intercept (b = bias)

$$\hat{y} = \boldsymbol{w}^\top \boldsymbol{x}$$

$$\mathrm{MSE}_{\mathrm{test}} = \frac{1}{m}\sum_i (\hat{\boldsymbol{y}}^{(\mathrm{test})} - \boldsymbol{y}^{(\mathrm{test})})_i^2 .$$

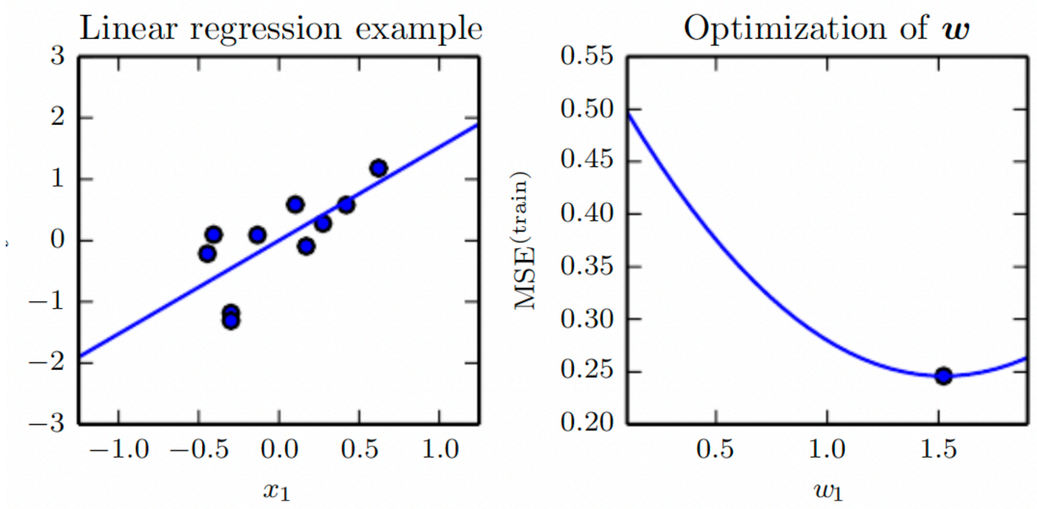
$$\mathrm{MSE}_{\mathrm{test}} = \frac{1}{m}||\hat{\boldsymbol{y}}^{(\mathrm{test})} - \boldsymbol{y}^{(\mathrm{test})}||_2^2 ,$$

$$\hat{y} = \boldsymbol{w}^\top \boldsymbol{x} + b$$

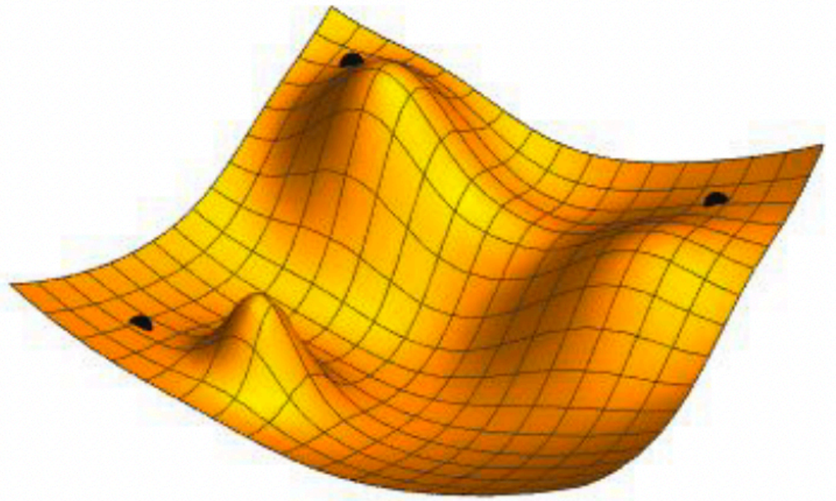
- The cost function can be defined in different ways: e.g. LASSO (Least absolute shrinkage and selection operator)

$$\hat{\boldsymbol{w}}_{\mathrm{LASSO}}(\lambda) = \arg\min_{\boldsymbol{w}\in\mathbb{R}^p} ||\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y}||_2^2 + \lambda||\boldsymbol{w}||_1$$

- The distance is defined including weight decay with a **hyperparameter** λ to select small weights (if w is big, the cost function gets bigger)
- This a form of **regularisation** of the model



Cost function minimization



The model is fit to training data by minimizing the cost function,

… but there are practical issues that complicate life:

- the cost function is not known at the ground truth
- it is defined in a multi-d space
- it may have multiple local minima
- in modern ML algorithm you may have millions of parameters

Different methods to find the minimum:

- Gradient Descent (GD)

$v_t = \eta_t \nabla_\theta E(\theta_t)$

$\theta_{t+1} = \theta_t - v_t$

η_t = learning rate

learning rate decreases in directions where the gradient is large

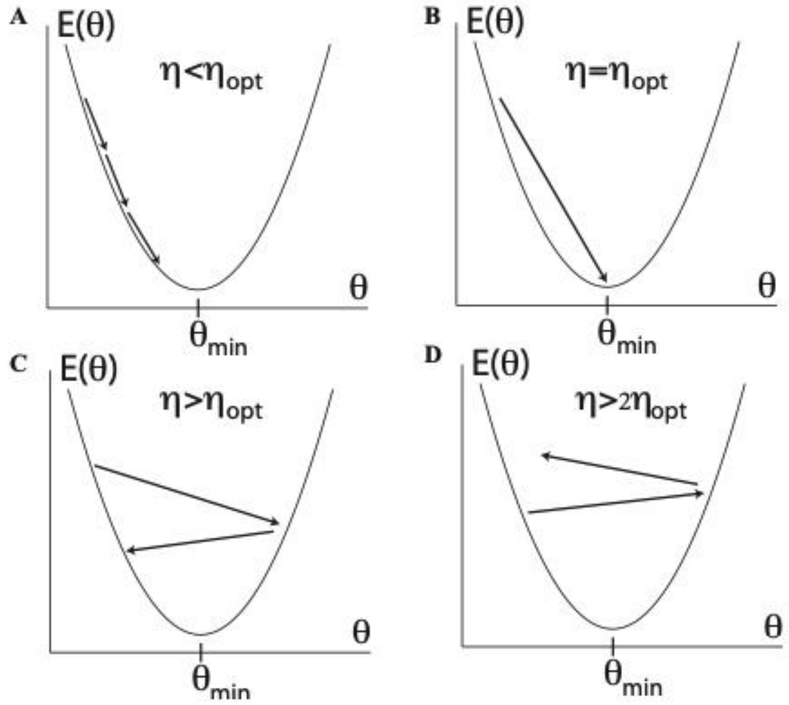
- Stochastic Gradient Descent (SGD)

$v_t = \eta_t \nabla_\theta E^{MiniBatch}(\theta_t)$

$\theta_{t+1} = \theta_t - v_t$

chose randomly where to start to avoid local minima

- Gradient Descent with momentum
- Gradient Descent with second momentum
  - RMSprop
  - ADAM



Fitting vs. predicting (interpolating vs. extrapolating)

We would like the ML algorithm *learns* the underlying law from (training) data and not simply reproduces the (training) data set features (otherwise it will not be able to say anything about the test data set outside the training range)

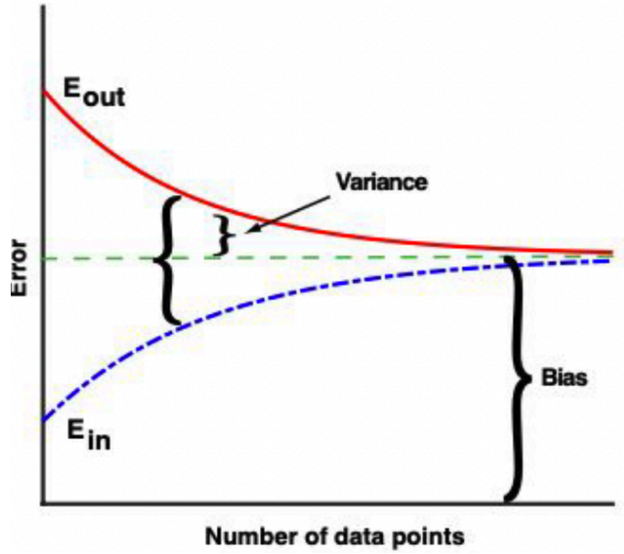
In-sample error vs. out-of-sample error (aka: How could ML possibly work???)

- We minimized the error in the training sample: what about the test?

  Bias = difference between model and true

  Variance = difference between in/out errors

- Assuming that training and test samples are identically distributed
  - A more sophisticated (**expressive**) model can minimize the bias
  - Variance can be reduced by increasing statistics (train/test sample)

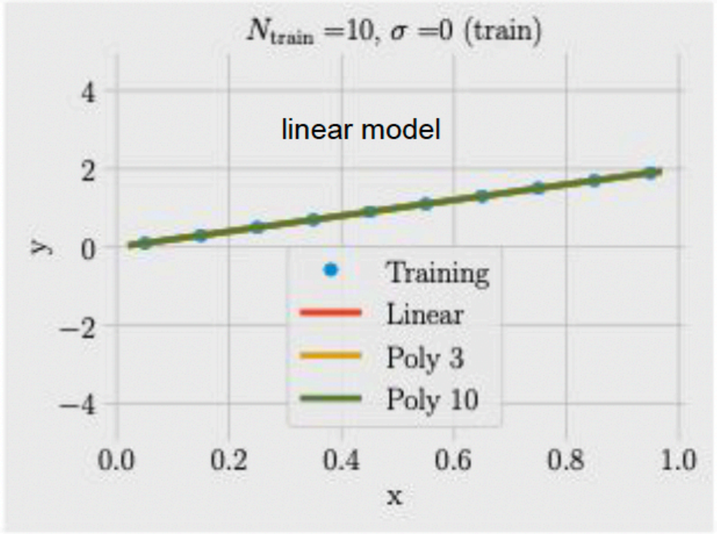


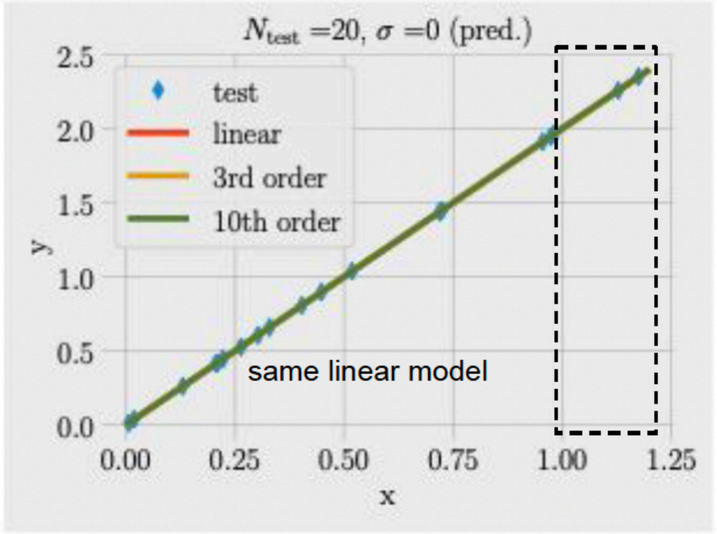
- The model that provides the best explanation for the current dataset will probably not provide the best explanation for future datasets
- The discrepancy between Ein , Eout grows with the complexity of our data and of our model (increased model parameters, high dimensional space, **curse of dimensionality**)
- For these reasons (and for complicated models), predicting and fitting can be different things. Need to pay attention to out-of-sample performance.
- Fitting existing data well is fundamentally different from making predictions about new data.

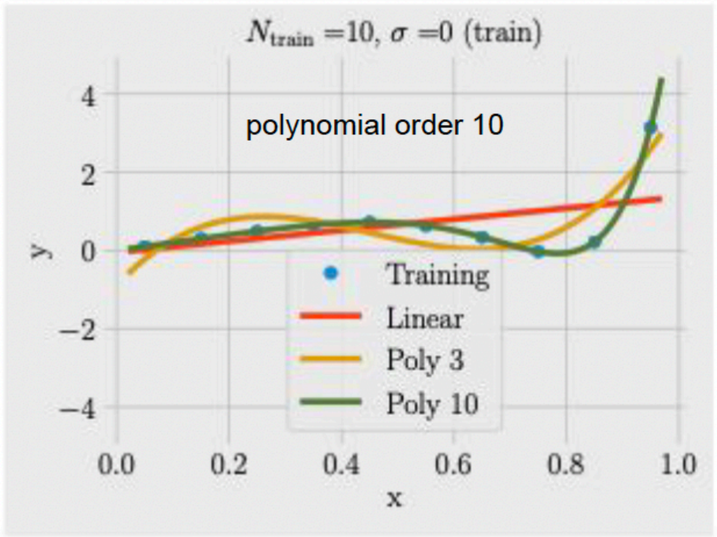
Fitting vs. predicting

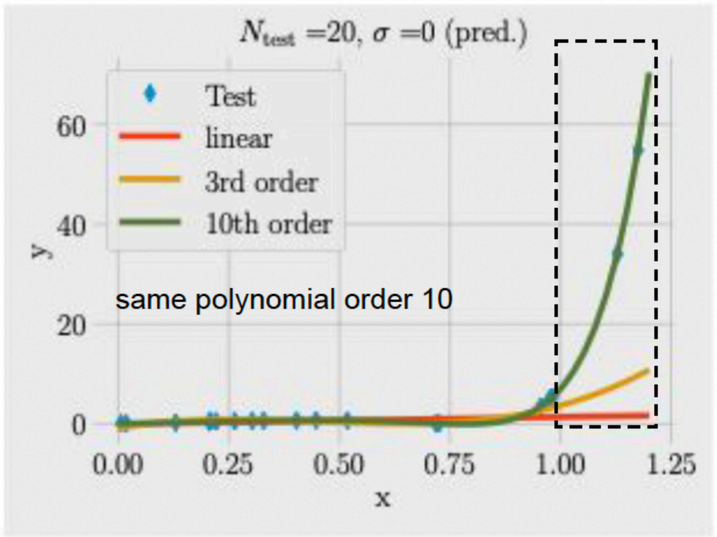
Signal only

- Traing interval [0,1.0]
- Test interval [0,1.2] (larger than the training interval)
- Data sampled from
  - f(x) = 2 x
  - f(x) = 2 x -10 x^5 + 15 x^{10}
- If no noise, even with a small training set (N_{train} = 10 < N_{test} = 20) the model class that generated the data provides the best fit and also the best out-of-the sample prediction.





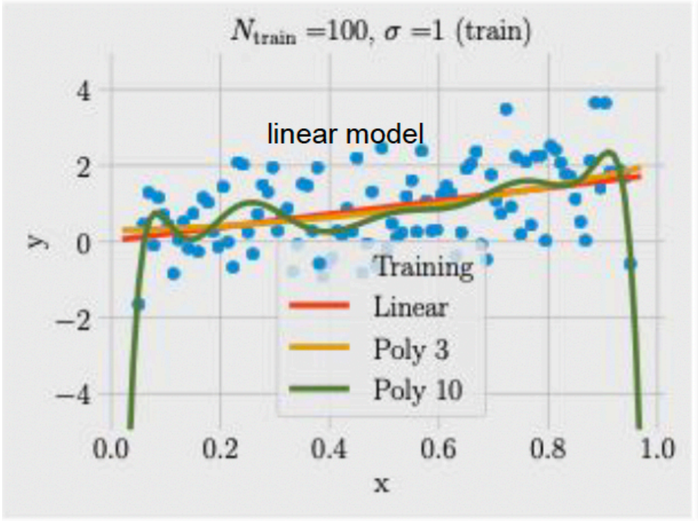


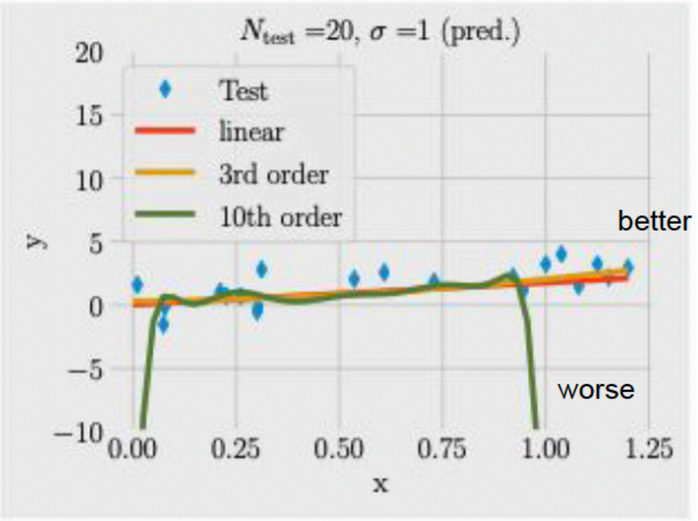


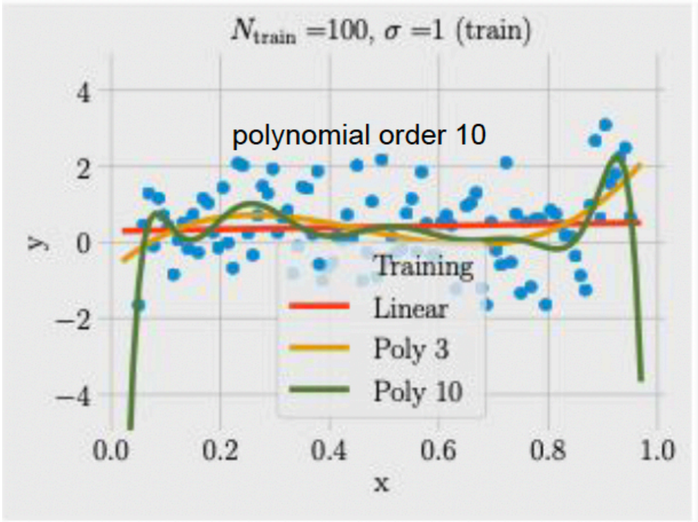
Fitting vs. predicting

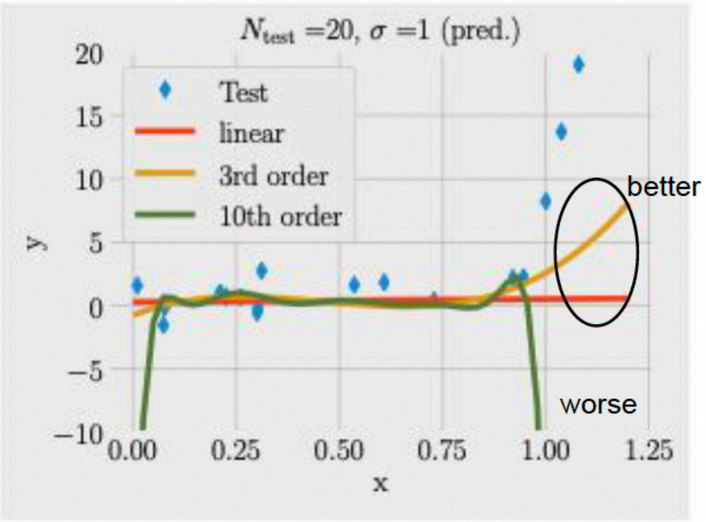
Signal + noise

- Traing interval [0,1.0]
- Test interval [0,1.2] (larger than the training interval)
- Data sampled from
  - f(x) = 2 x
  - f(x) = 2 x -10 x^5 + 15 x^{10}
- Noise=1, larger training set (N_{train} = 100, N_{test} = 20)
- LINEAR provides better out-of-sample prediction for POLY10 too





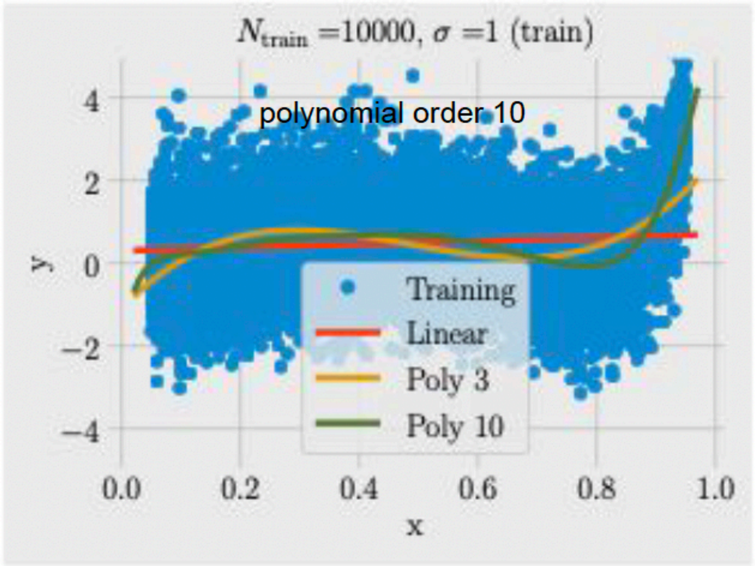


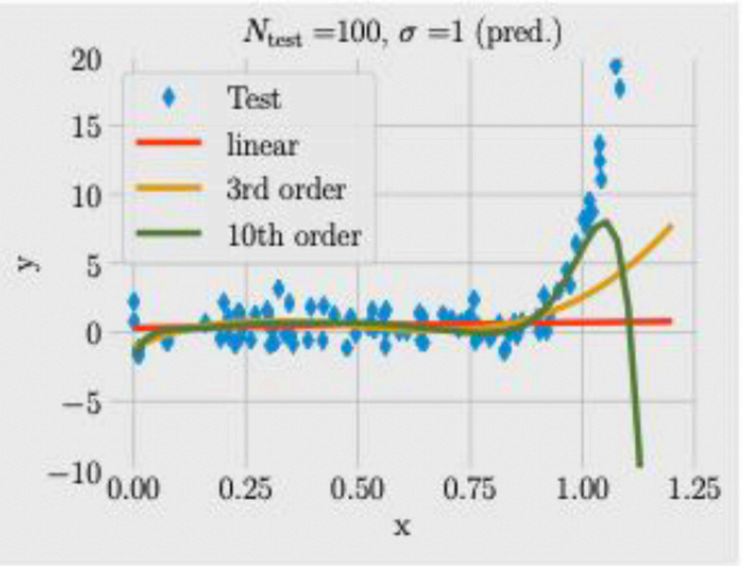


Fitting vs. predicting

Signal + noise with a larger training sample

- Traing interval [0,1.0]
- Test interval [0,1.2] (larger than the training interval)
- Data sampled from
  - f(x) = 2 x
  - f(x) = 2 x -10 x^5 + 15 x^{10}
- Noise=1, larger training set (N_{train} = 10000, N_{test} = 100)
- POLY10 provides the best fit and extrapolation close to the training interval but degrades quickly outside





Bias-variance tradeoff: with limited statistics, it is better to use less expressive (simpler) models (larger bias smaller variance)

A side note …

Can AI help in recovering detectors effects?

All detectors introduce (at least) two effects in a measured data set:

- **smearing:** the detector has a finite resolution making DETECTED ≠ TRUE
- **acceptance**: every detector only covers a fraction (small/large) of the available phase space

Smearing (unfolding)

- Fixable!
- Train the ML algorithm with data passed through a proxy of the detector (GEANT-like) to learn the distortions
- Use this knowledge to unfold detector distortion

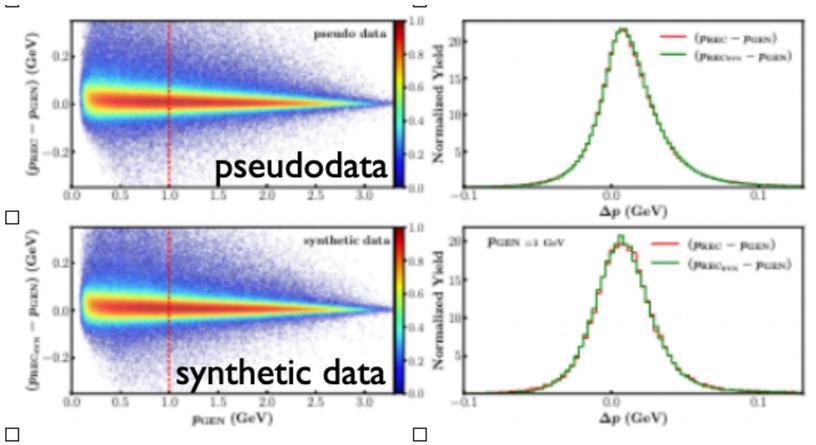
Acceptance

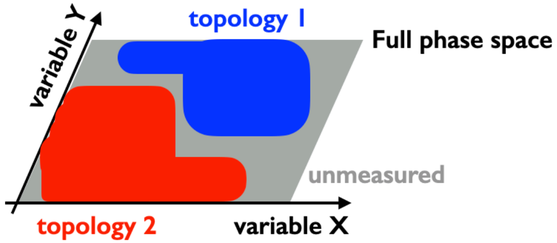
- More difficult to recover: the cross section (Probability Density Function) can not be constrained by general rules (other than being positive) since it reflects the underlying (a-priori unknown) physics
- No PDF extrapolation outside detector acceptance is possible (based on measured phase space)
- A possible approach is to impose conditions to the PDF via constraints on scattering aptitudes Ai (parity conservation, analiticity, unitarity, …)
- Considering that:

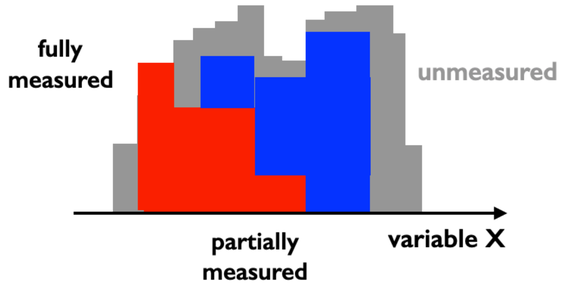
$$\text{XSec} = \sum |A_i|^2 \qquad A_i = \sum (\text{Scattering amplitude for each possible process})$$

Ai are difficult to constrain supervising on XSec

Resolution of the CLAS detector learned by a GAN







Neural networks

- NNs are powerful supervised learning techniques
- Around the ’80s, it became widely used today thanks to the progress in hardware (CPU, GPU, and now FPGA)
- Several libraries are available to simplify custom implementation (PyTorch, Kera, TensorFlow, …)

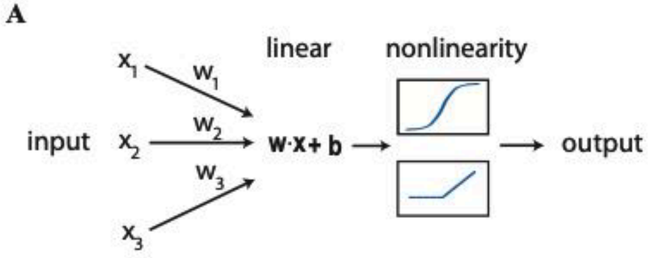


FIG. 35 **Basic architecture of neural networks.** (A) The basic components of a neural network are stylized neurons consisting of a linear transformation that weights the importance of various inputs, followed by a non-linear activation function. (b) Neurons are arranged into layers with the output of one layer serving as the input to the next layer.

- The basic unit of a neural network is a stylized **neuron** *i* that takes *d* input features *x = (x1, x2, …, xd)* and produces a scalar output $a_i(x)$
- The first **layer** is called the **input layer**, the middle layers are called the **hidden layers**, the final layer is the **output layer**
- The exact function a_i depends on the type of non-linearity used in the NN
- It can be decomposed into a linear operation that weights the relative importance of the various inputs and a non-linear transformation $\sigma_i(z)$ **activation function**

weights bias

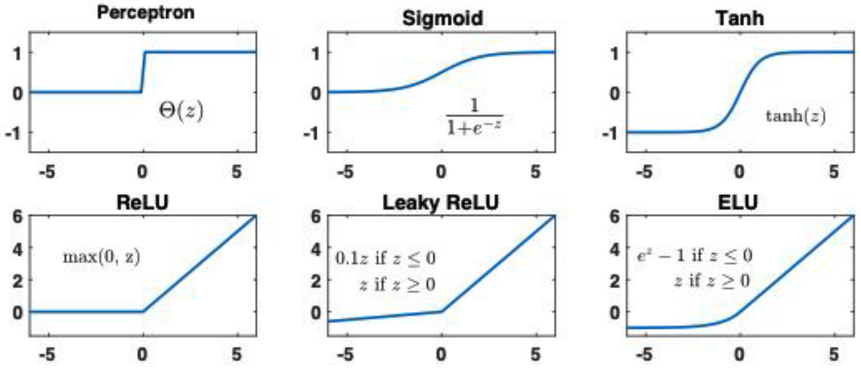
$$z^{(i)} = \boldsymbol{w}^{(i)} \cdot \boldsymbol{x} + b^{(i)} = \mathbf{x}^T \cdot \mathbf{w}^{(i)}$$

inputs

activation

$$a_i(\mathbf{x}) = \sigma_i(z^{(i)})$$

(non-linear functions)

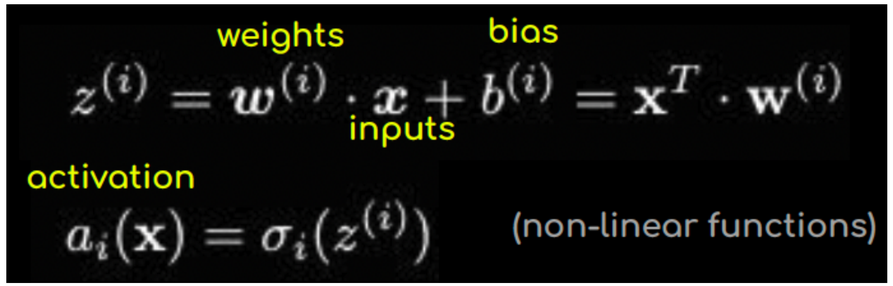


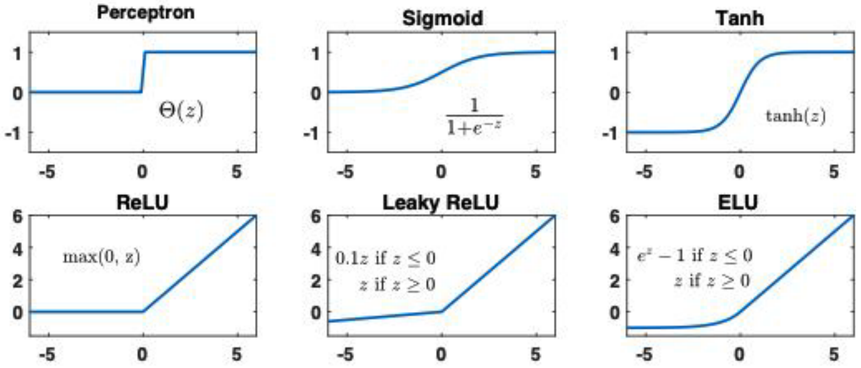
Neural networks

- NNs are powerful supervised learning techniques
- Around since '80s, it became widely used today thanks to the progress in hardware (CPU, GPU, and now FPGA)
- Several libraries are available to simplify custom implementation (PyTorch, Kerr, TensorFlow, …)

- The choice of the activation function is critical
- NN is trained using a gradient descent method (a derivative of neural function as a function of weights *w* and bias *b*)
- Derivative of step-function can be an issue (tanh and sigmoid were widely used in the past)
- When w's become large, some activation functions saturate and the derivative tends to zero (vanishing gradient).
- ReLU and ELU are not vanishing even for large w's (widely used today)

- The basic unit of a neural network is a stylized **neuron** *i* that takes *d* input features *x = (x1, x2, …, xd)* and produces a scalar output $a_i(x)$
- The first **layer** is called the **input layer**, the middle layers are called the **hidden layers**, the final layer is the **output layer**
- The exact function a_i depends on the type of non-linearity used in the NN
- It can be decomposed into a linear operation that weights the relative importance of the various inputs and a non-linear transformation $\sigma_i(z)$ **activation function**



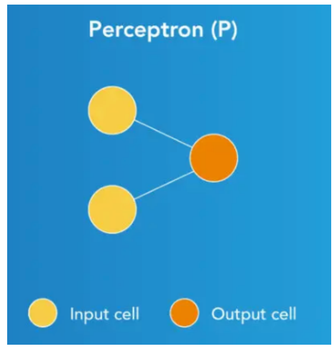


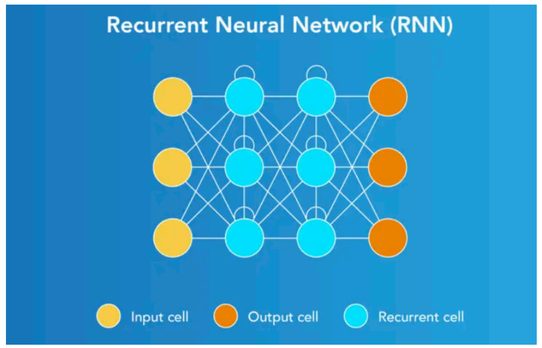
Neural networks in practice

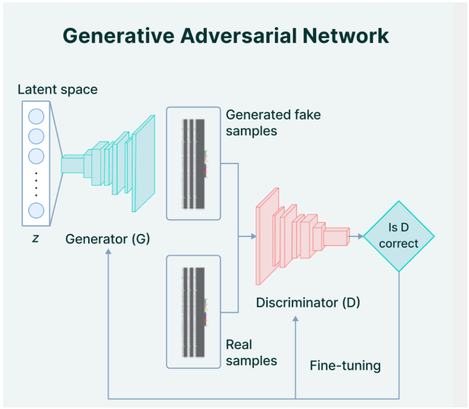
- Chose the NN architecture
- Define your loss function
- Regularize it
- Prepare the data set
- Train the network
- Check results

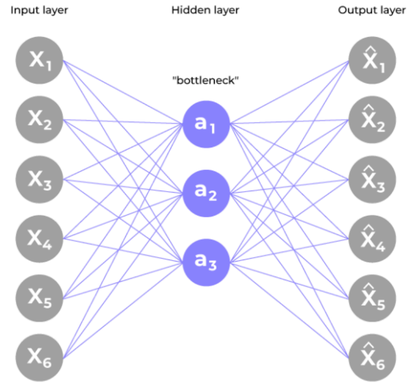
have fun!

- NN is more powerful than a simpler linear regression thanks to the inter-connections of weights and non-linearity introduced by the activation function
- Universal approximation theorem: a neural network with a single hidden layer can approximate any continuous, multi-input/multi-output function with arbitrary accuracy
- The more complicated a function, the more hidden units (and free parameters) are needed to approximate it
- Adding hidden layers also allows neural nets to learn more complex features from the data.
- Choosing the exact network architecture for a neural network remains an art
- Define the NN type (Perceptron, Feed Forward NN, Multilayer perceptron, Convolutional NN, Encoders, …) and details of the network (how many nodes, how many layers, the activation function, …)





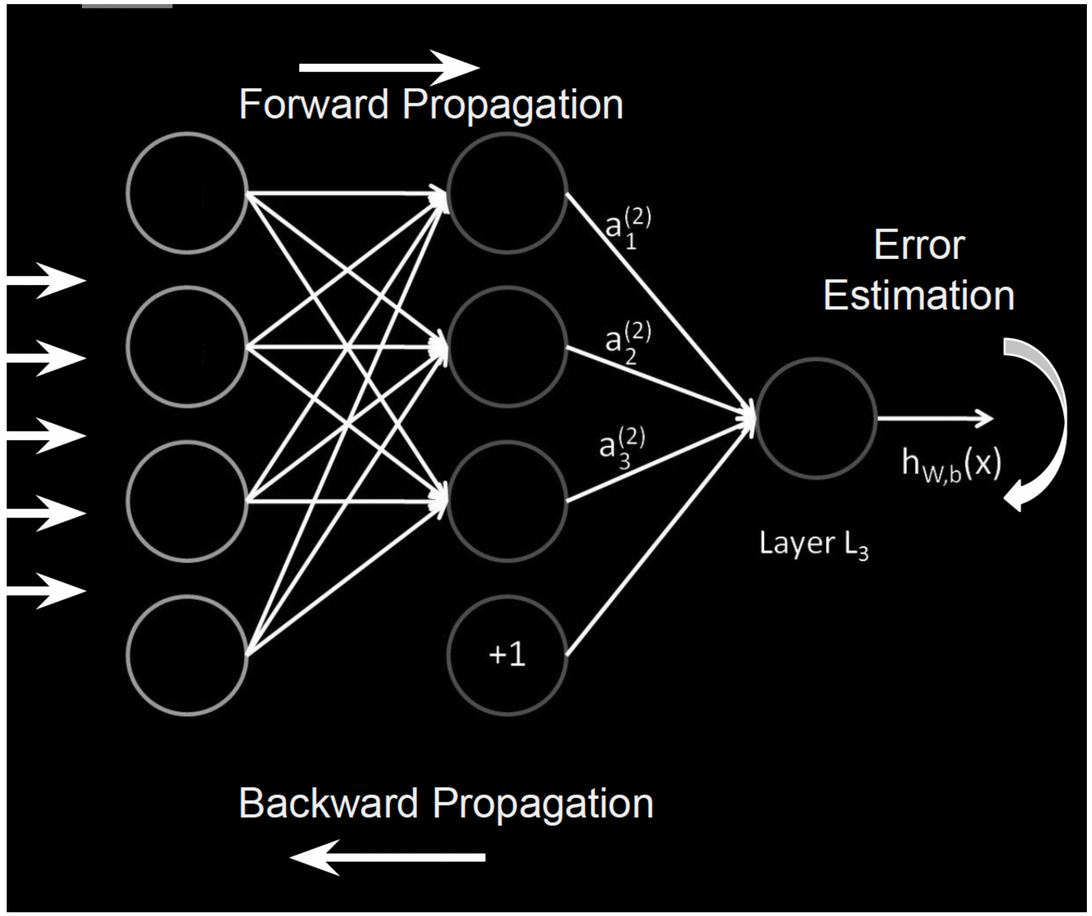




Neural networks in practice

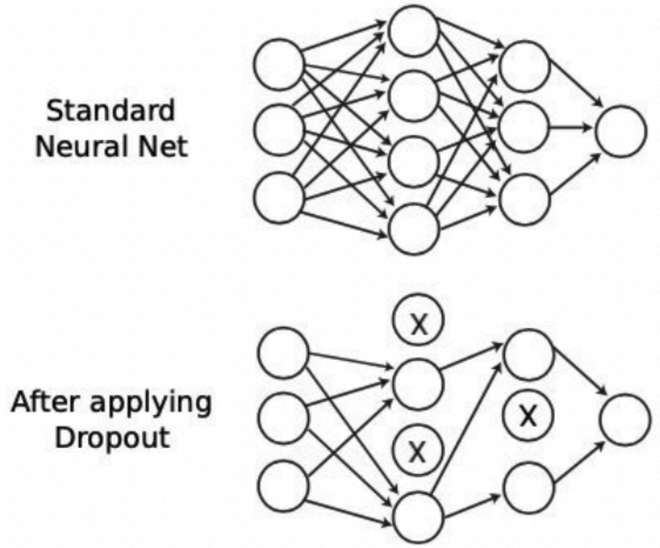
- Chose the NN architecture
- Define your loss function
- Regularize it
- Prepare the data set
- Train the network
- Check results

have fun!



- Deep Learning (adding more hidden layers) nets are good for learning non-linear functions (heavy processing tasks)
- Backpropagation is the key in the training procedure (how the NN improves over time)
- Regularisation (e.g. dropout) contributes to outperforming on new data (besides the training set)

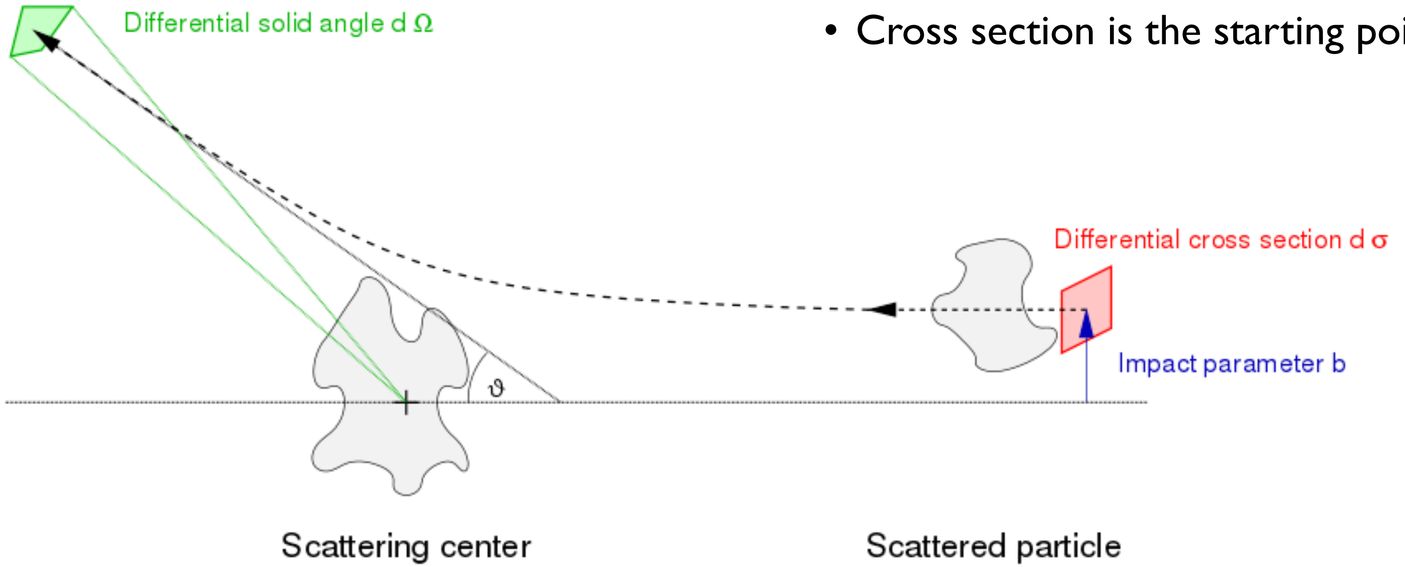
- Implicit regularization using Stochastic Gradient Descent: initialization, hyperparameter tuning, early stopping, .,...,



The cross section in particle physics

$$\frac{\mathrm{d}\sigma}{\mathrm{d}\Omega} = (2\pi)^4 m_i m_f \frac{p_f}{p_i} |T_{fi}|^2$$

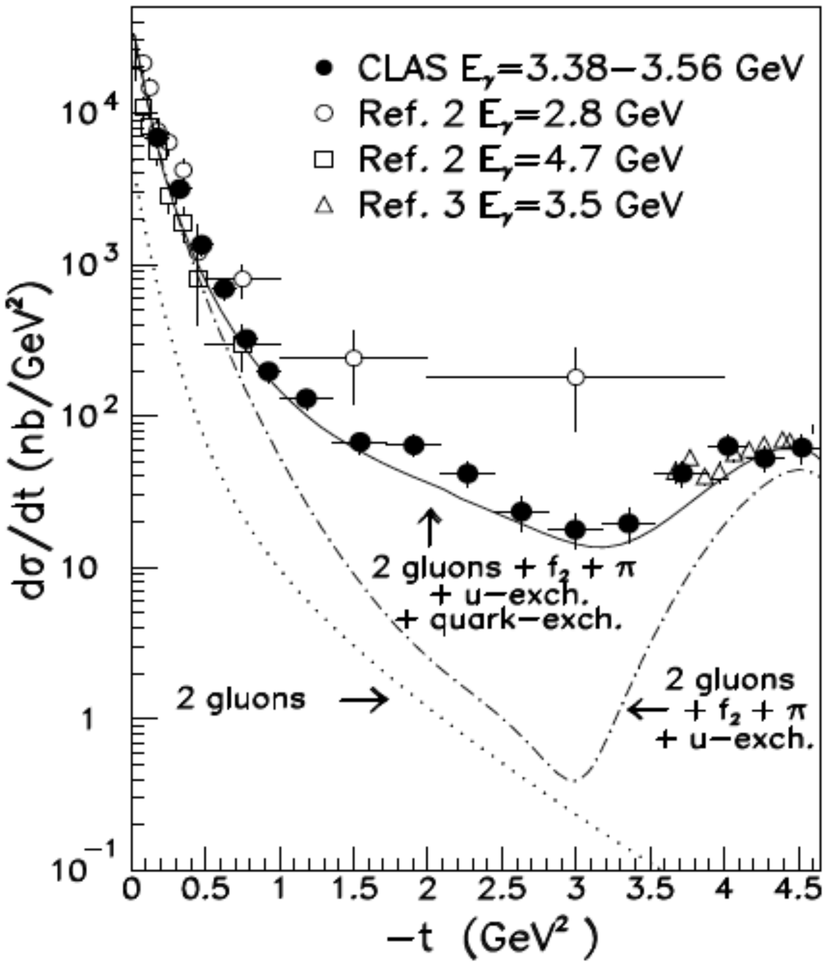
- The *cross section* is related to the transition probability between an initial to a final state
- In case of scattering, cross sections provides information about the elementary interaction
- Cross section is expressed as squared sum of scattering amplitudes (complex functions) depending on the kinematic Lorentz-invariant of the problem and embedding the interaction properties
- It is derived by measuring the momentum distributions of reaction particle (at different CM energy)
- Correlations between particles in the final state reflects the underlying dynamics
- Cross sections fully replaces the 4-mom data sample in a compact and efficient way
- Cross section is the starting point for any higher level physics analysis



- Traditional approach: particles (4-momenta) measured into the detector, extract the relevant observables, extract physics mechanisms
- Cross section **preserves** this information as replacement for the original particle-by-particle scattering information

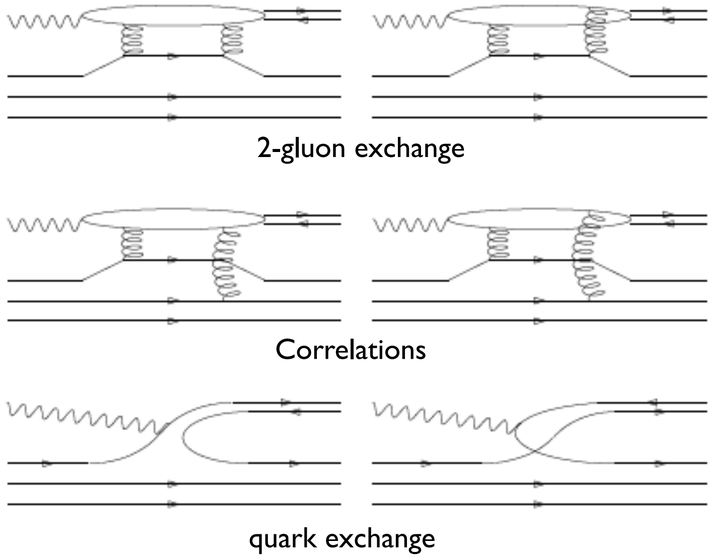
Exclusive reactions: 2 → 2

M. Battaglieri *et al.* (CLAS Collaboration) Phys. Rev. Lett. 90, 022002



JLab-CLAS g6 ω photo production at large momentum transfer

$\gamma p \rightarrow p \omega$



2 → 2 scattering (no polarisation)

- Initial state: known
- Final state: 2 x 3
- Parameters: (2 x 3) - 4 = 2
- Possible choice: *-t* and ϕ
- *t*he physics depends only on one variable (-t)

- It worked (and still works!) well if limited to channels with a single variable
- Xsec, Polarization observables, angular distribution, decay matrix, …

Exclusive reactions: 2 → 3

M. Battaglieri *et al.* (CLAS Collaboration) Phys. Rev. Lett. 90, 022002

2 → 3 scattering (no polarization)
- Initial state: known
- Final state: 3 x 3
- Parameters: (3 x 3) - 4 = 5 (E_γ fixed)
- Possible choice: $M^2_{\pi\pi}$, $M^2_{p\pi}$ θ_π, α, ϕ

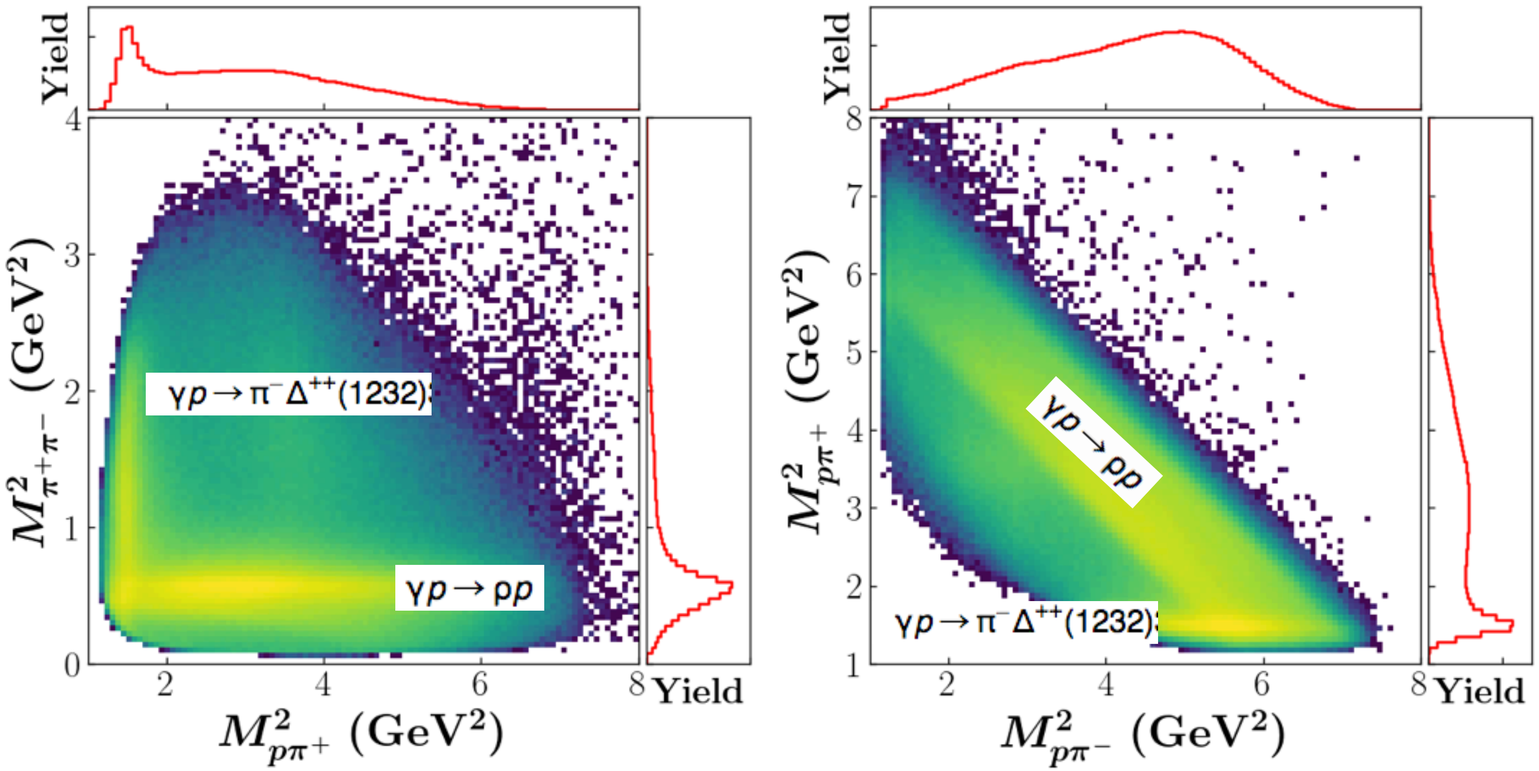
- It does not work (in practice) when you have several independent variables: multi-particle final states (spectroscopy) or multi-variable correlations (SIDIS)
- In the integration to reduce to 1-dim all correlations are lost

AI may provide a new way to look at data and extract observables and physics interpretation (on event by event base)

CLAS g11 2π photo production

- E_γ = (3.0 - 3.8) GeV
  $\gamma\, p \rightarrow p\, \pi^+ \pi^-$ exclusive reaction
  - data set analyses so far $\gamma\, p \rightarrow p\, \pi^+$ (π^-) + small contamination of $\gamma\, p \rightarrow p\, \pi^+$ (more than a missing π^-)
  - complicated dynamic for the overlap of ($p\pi$) to form Δ baryon resonances and ($\pi\pi$) to form meson resonances

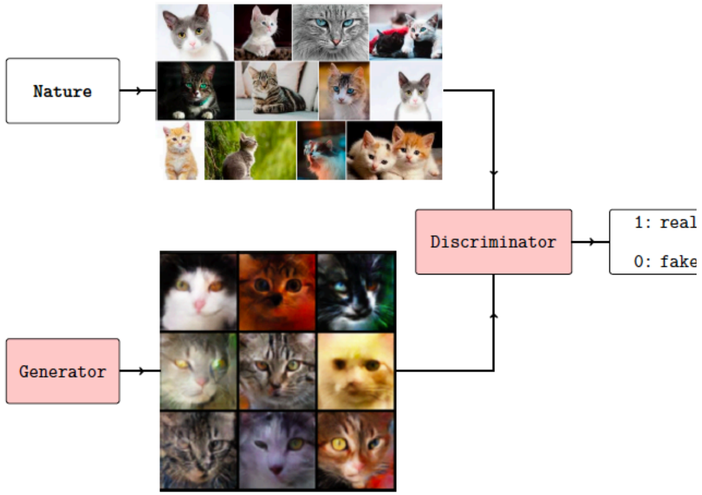
$$\frac{d\sigma\,(\gamma\, p \rightarrow p\,\pi^+\pi^-)}{dM_{\pi\pi}\, dM_{p\pi}\, d\cos(\theta_\pi)\, d\alpha\, d\phi}$$

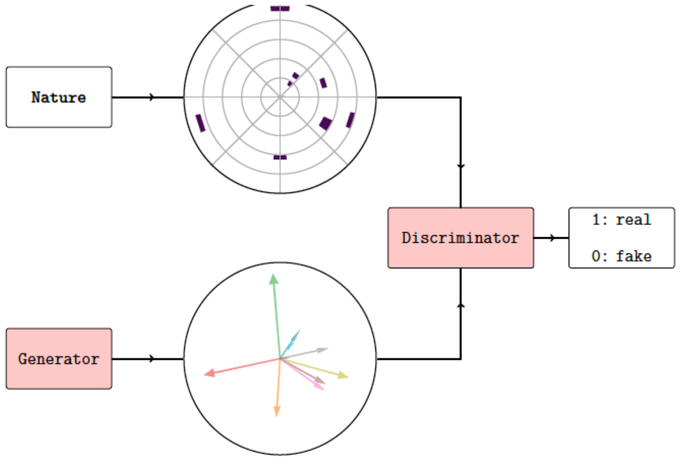


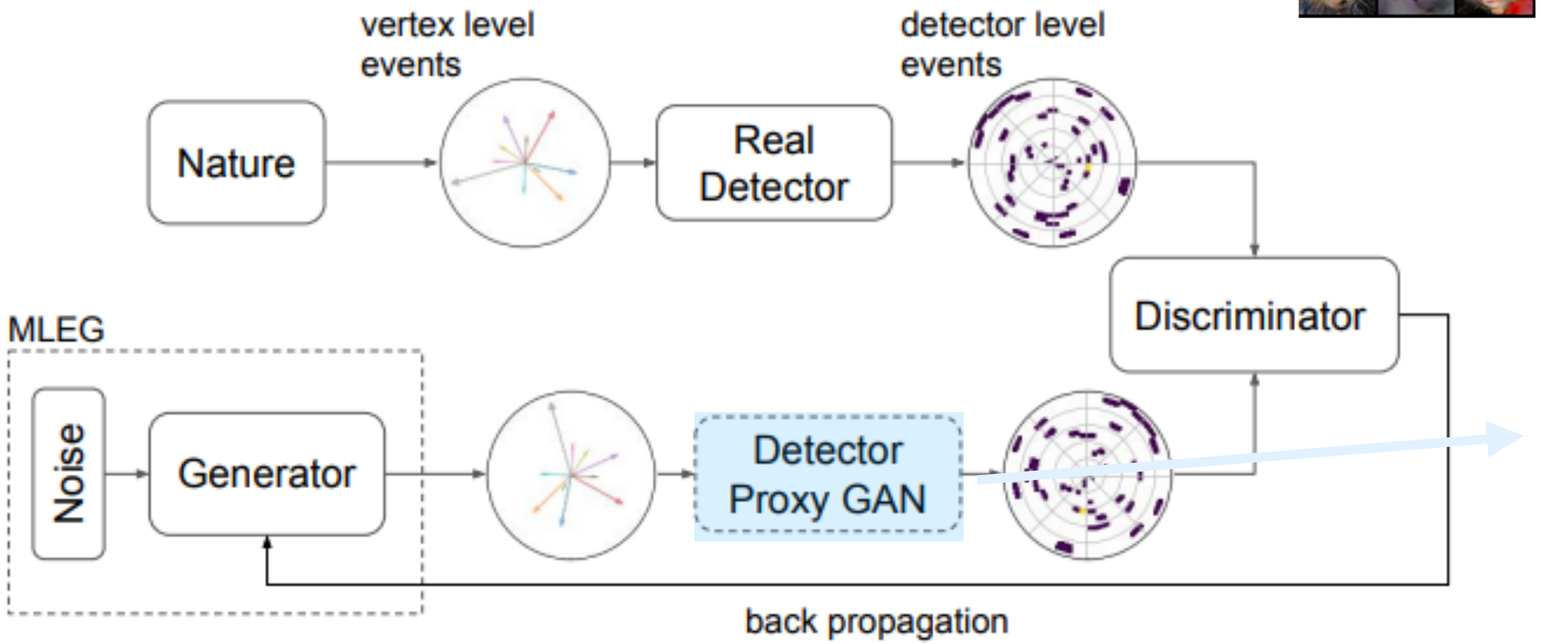
Credit: Y.Alanazi Awadh, , P..Ambrozewicz, G. Costantini A.Hiller Blin, E. Isupov, T. Jeske, Y.Li, L.Marsicano W. Menlnitchouk, V.Mokeev, N.Sato, A.Szczepaniak, T.Viducic

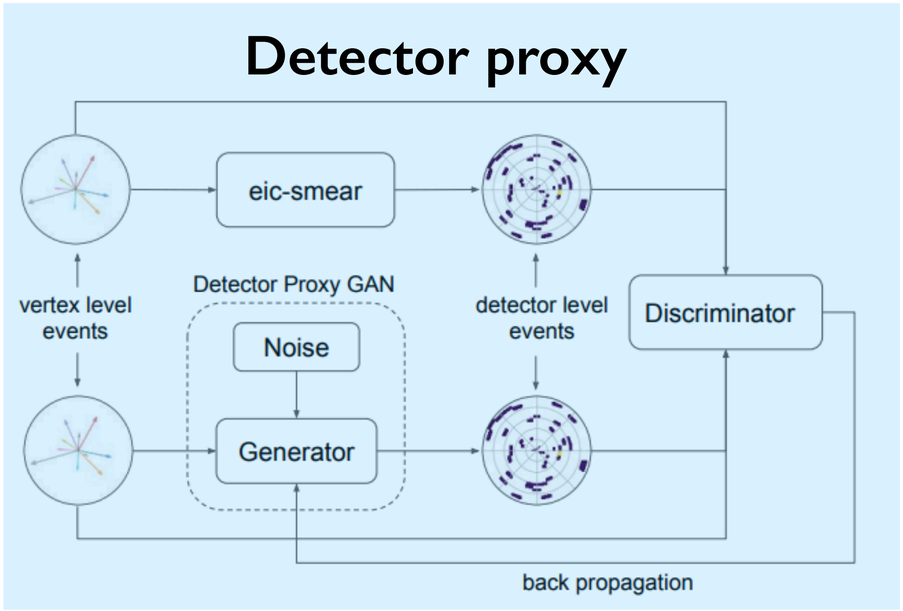
ML Event Generator GAN scheme

- The colored boxes are built using NNs
- Discriminator is trained to output "real" for Nature samples
- Generator is trained to fool the discriminator
- The Generator can be used as data compression tool
- Typical size for the Generator: O(MB) - to be compared to NP/HEP experiments data set O(GB/TB)
- Simple to distribute instead of events stored on tapes







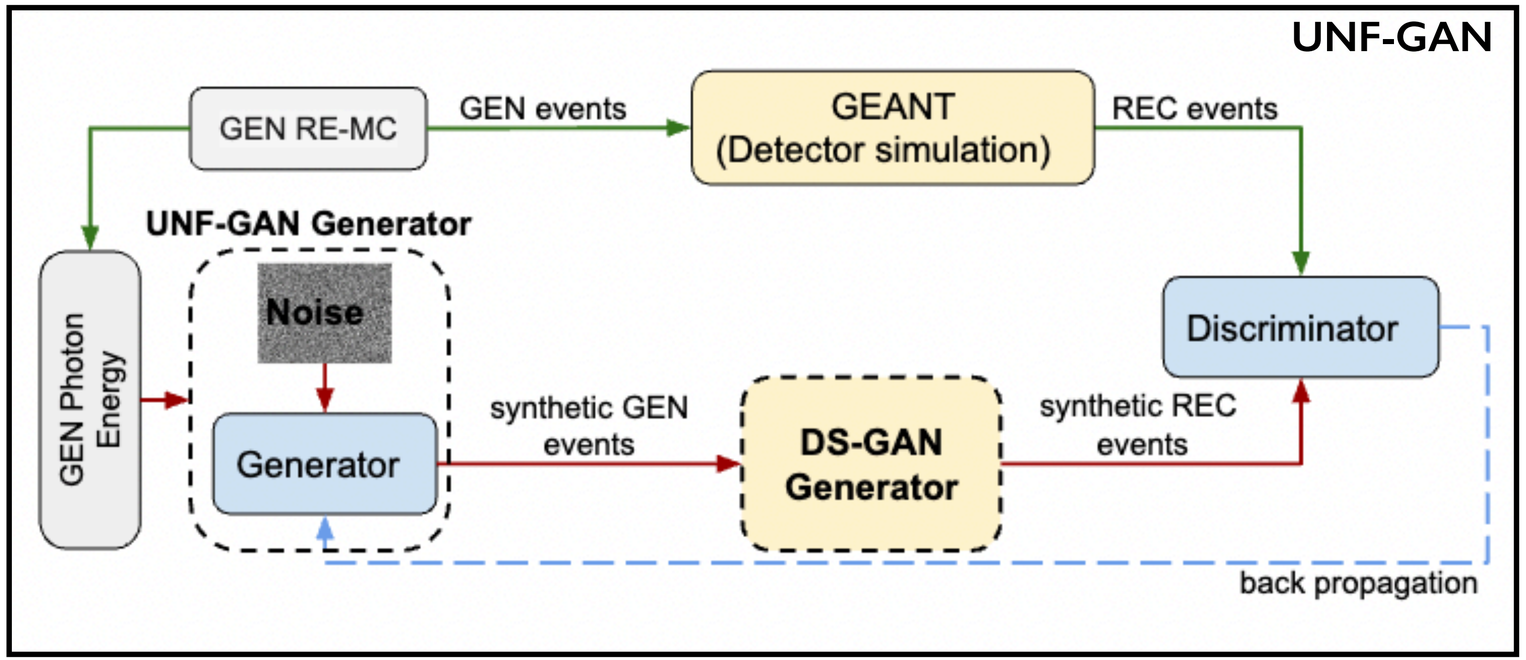


2π photo production closure test

T.Alghamdi, M.Battaglieri, A.Golda, A. Hiller Blin, L.Marsicano, W.Melnitchouk, G.Montaña, E.Isupov, Y.Li, V.Mokeev, A.Pilloni, N.Sato, A.Szczepaniak, T.Vittorini, Y.Alanazi *to appear in ArXiv*

Unfolding GAN (UNF-GAN) that includes the DS-GAN, and training it with RE-MC REC pseudodata

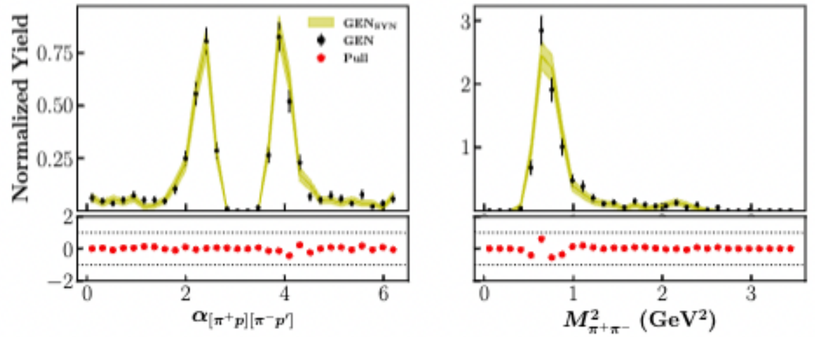
- UNF-GAN trained with RE-MC REC pseudodata (exp data proxy)
- DS-GAN used to unfold CLAS detector effects (within acceptance)



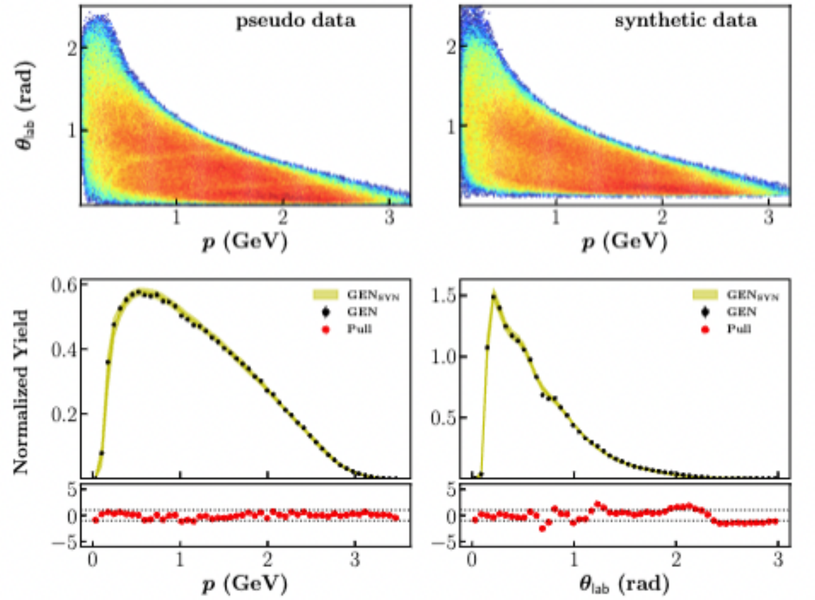
5. Compare UNF-GAN GEN SYNT data to RE-MC GEN pseudodata

Good agreement (± 1σ) for lab variables and in 4D bins

Distribution in 4D bins



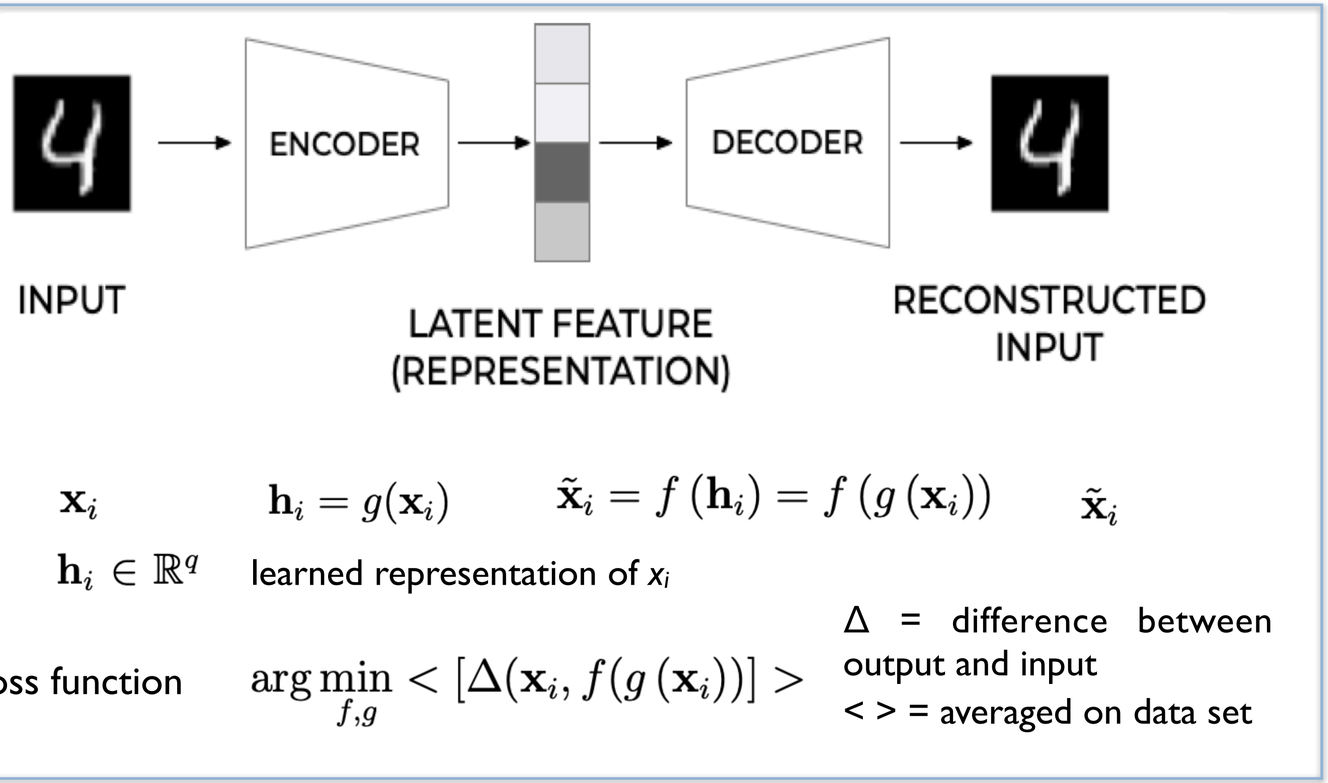
RE-MC GEN pseudodata vs. UNF-GAN SYN data



Autoencoder

An autoencoder is a type of algorithm with the primary purpose of learning an "informative" representation of the data that can be used for different applications by learning to reconstruct a set of input observations well enough

- Components: encoder, decoder (neural nets), and a latent feature representation
- Latent feature representations (learned) should be meaningful (often part of the autoencoder results)



$\mathbf{x}_i$ $\quad \mathbf{h}_i = g(\mathbf{x}_i) \quad \tilde{\mathbf{x}}_i = f(\mathbf{h}_i) = f(g(\mathbf{x}_i)) \quad \tilde{\mathbf{x}}_i$

$\mathbf{h}_i \in \mathbb{R}^q$ learned representation of x_i

Loss function $\arg\min_{f,g} < [\Delta(\mathbf{x}_i, f(g(\mathbf{x}_i))] >$

Δ = difference between output and input

< > = averaged on data set

- E.g. a hand-written digit. The input is the image of the digit, the output is the reconstructed image, and the latent features are the number of lines representing the digit

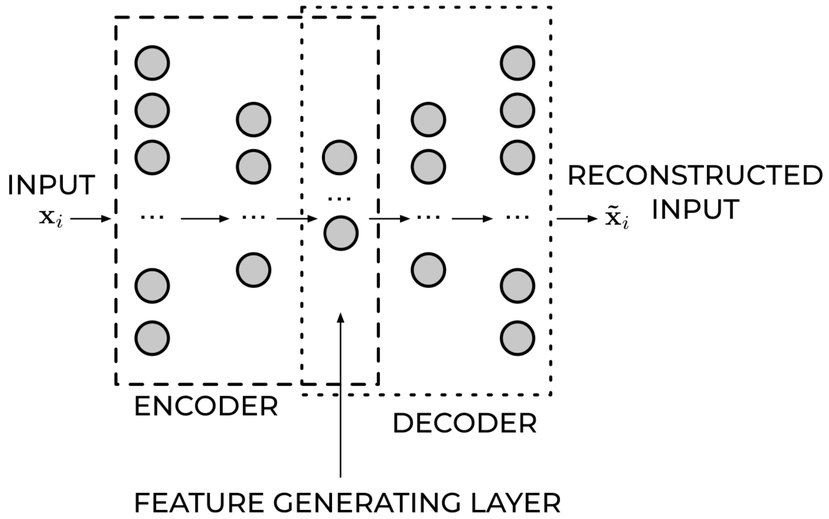
- To reduce latent features dimensionality use regularization to enforce sparsity

$$\arg\min_{f,g} \left(\mathbb{E}\left[\Delta(\mathbf{x}_i, g(f(\mathbf{x}_i))\right] + \lambda \sum_i \theta_i^2\right)$$

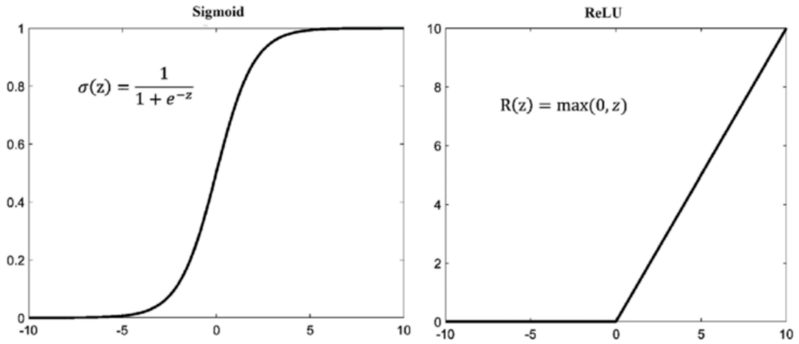
θ_i = parameters (NN weights)

- Another trick is to tie the weights of the encoder and decoder
- Training the AI means finding functions *f* and *g* that minimize the Loss function
- Reducing the dim from n = dim(xi) to q = dim (*hi*) with q<n
- The learned representation is enough for the decoder to reconstruct x_i (part of the information is embedded in the decoder weights)

Feed-Forward autoencoder



- Odd number of layers, symmetrical wrt the middle layer
- Middle layer has a reduced number of neurons q<n (bottleneck)
- Activation functions: ReLU (if $xi>0$) or sigmoids (if $0<xi<1$)



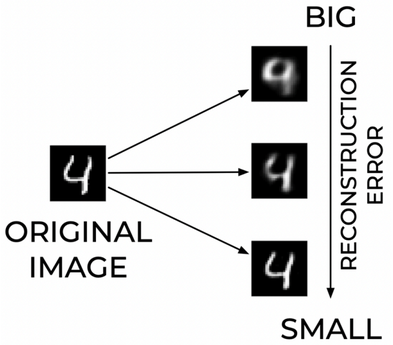
- Data are normalised to stay within [0,1]
- Loss function:

$$L_{\text{MSE}} = \text{MSE} = \frac{1}{M}\sum_{i=1}^{M} |\mathbf{x}_i - \tilde{\mathbf{x}}_i|^2$$

$$L_{\text{CE}} = -\frac{1}{M}\sum_{i=1}\sum_{j=1}[x_{j,i}\log\tilde{x}_{j,i} + (1-x_{j,i})\log(1-\tilde{x}_{j,i})]$$

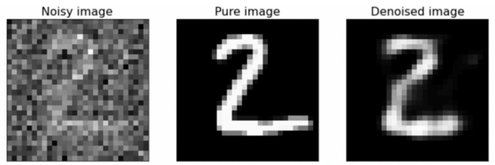
- Reconstruction error (RE)

$$\text{RE} \equiv \text{MSE} = \frac{1}{M}\sum_{i=1}^{M} |\mathbf{x}_i - \tilde{\mathbf{x}}_i|^2$$



Autoencoder applications

- Dimensionality reduction: efficient, training with large amounts of data
- Classification: run a classifier on latent space, very fast, with limited accuracy loss
- Anomaly detection: outlier ID (when trained without or a negligible fraction)
- Denoising: input noisy image and output true image



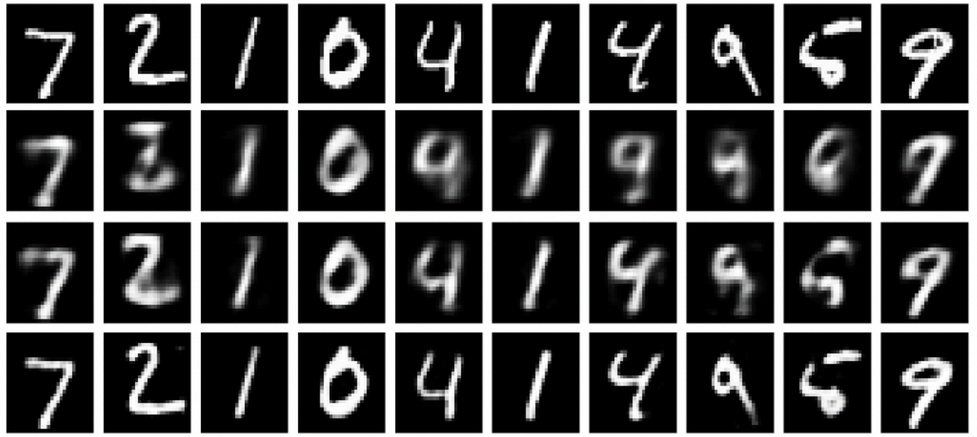
Feed-Forward autoencoder

Example: hand-written figures

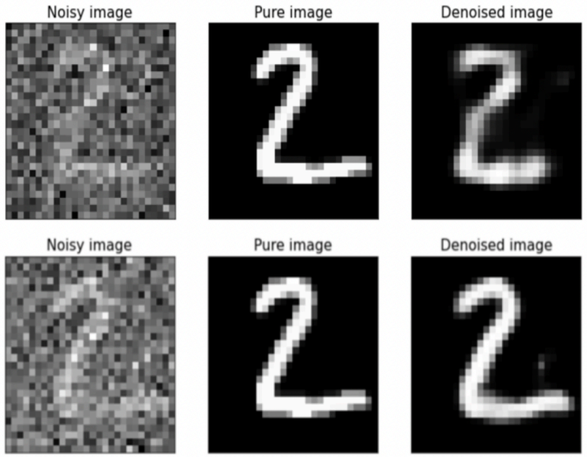
- Goal: compress the image to the latent space and decompress it
- Each image is 28x28 pixels = 784 features (q)

Autoencoder

- 3 layers with different latent dimension (8,16,64)
- ADAM optimizer
- Loss function: L_{CE} and L_{MSE}
- 30 epochs (batch size 256)



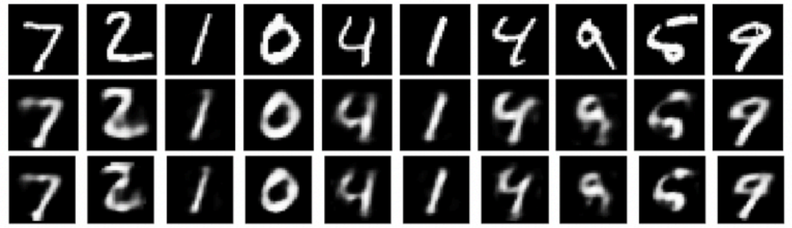
Dimensionality reduction works!



FFA with Architecture (784,32,784)

CA with Architecture ((28x28), (26x26x64), (24x24,32), (26x26x64), (28x28))

Denoising works!



Robust wrt loss function choice

Autoencoder with convolutional layers