Applications of Machine Learning in Nuclear Physics

Aaron Angerami

Lawrence Livermore National Laboratory
2023 National Nuclear Physics Summer School
University of California, Riverside
July 13, 2023

Lawrence Livermore National Laboratory

Introduction/subject orientation

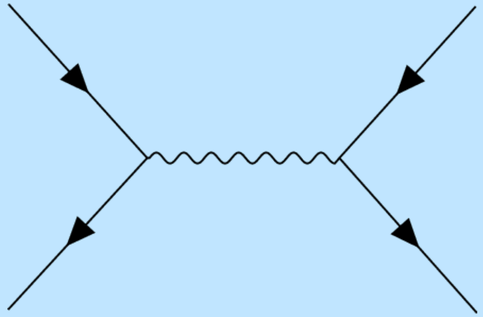
- Machine learning is a vast and rapidly evolving subject
- These lectures provide an informal introduction to the basics of AI/ML and its applications to nuclear physics
- They are not intended to be comprehensive but rather to give you enough of an introduction to the subject that you might use these methods in your own research
- These lectures draw heavily from the Particle Data Group review articles on Statistics and Machine Learning

The ML “revolution”

- Ideas underlying a lot of ML techniques have been available for a long time. Some have been used by physicists in some capacity for decades… so why now?
  - Brute force computing power and availability of data made application of such methods more feasible leading to much more active research in this area
  - Tools have become extremely accessible
    - Frameworks like Tensorflow and Pytorch have high-level Python APIs that allow for the harnessing of this computing power
    - Teaching/collaborative tools like Jupyter notebooks, virtual environments

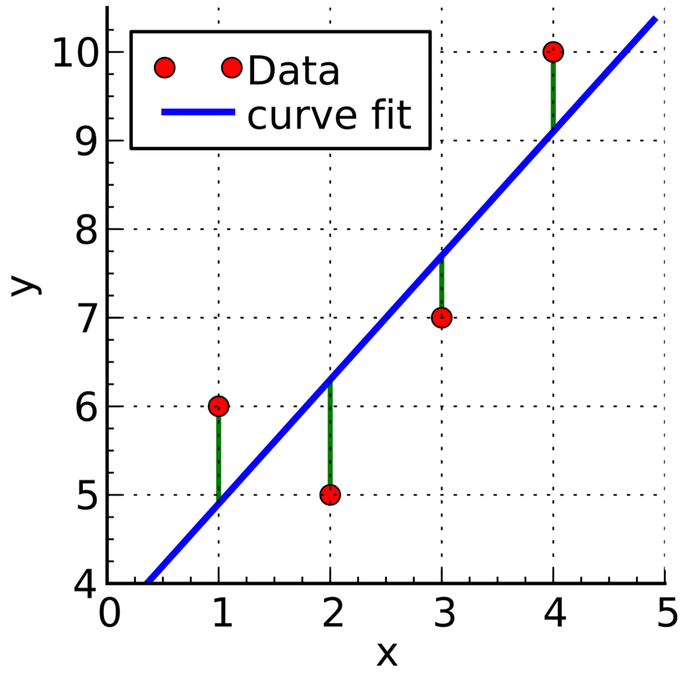
Feynman brought quantum field theory to the masses.

Julian Schwinger on Feynman Diagrams



Paradigm: fitting

- We usually apply a fit function or a *model* to some data
  - The model contains parameters
  - In an ML context this is usually called *regression*



- We frequently encounter two types of “fits”
  - Parameter estimation where parameter means something (inference)
  - Characterization/summary/smoothing of some data, individual parameters meaningless, just care about overall shape

- First fit method researchers enounter is typically a “chi-squared fit” or some form of least squares

Statistical Inference: terminology

- Goal is to use data (x) to test a hypothesis (H) and possibly infer the values of model parameters (θ) using statistics and probability, two interpretations of probability typically used

- *Frequentist* — probability for an outcome of an experiment is empirically interpreted in terms the frequency with which that outcome occurs when the experiment is repeated many times
  - Hypotheses and parameters of a model are not assigned a probability, but confidence intervals can be evaluated

- *Bayesian* — probabilities quantify degree of belief
  - Uses Bayes' Rule to define probability distributions for model parameters, but requires assumptions about the "prior" distribution of the parameters

Statistical Inference: Bayes' Rule

Likelihood (aka $L(\theta)$)
Probability of data given model parameter

Posterior
Probability of model parameter given data

Prior
Probability of model parameter

$$P(\theta \mid x) = \frac{P(x \mid \theta)\ \pi(\theta)}{\int P(x \mid \theta')\ \pi(\theta')\mathrm{d}\theta'}$$

Normalization factor
Numerator integrated over all possible parameter values

Method of Maximum Likelihood

Find values of parameters that maximize the likelihood function

$$\frac{\partial L}{\partial \theta_i} = 0$$

Unfortunately this is often also sometimes called “ML.” If the data consists of a fixed number of measurements (e.g. N “events”) that are independent and follow the same distribution then

$$L(\theta) = \prod_{i=1}^{N} P(x_i \,|\, \theta)$$

For this reason (and a few others) we often consider the log likelihood

$$\ln L(\theta) = \sum_{i=1}^{N} \ln P(x_i \,|\, \theta)$$

Since the logarithm is a monotonic function, the θ values that maximize L will also maximize $\ln L(\theta)$. Note that if N itself is random, then one must use the extended likelihood formalism, which has an additional factor of $\mu^N e^{-\mu}/N!$ to account for Poisson statistics on N.

Method of Maximum Likelihood: Example

Often we assume measurements x_i are normally distributed about our measured value with some uncertainty σ_i, and we fit a model that predicts the mean

$$L(\theta) = \prod_{i=1}^{N} \exp\left[-\frac{1}{2}\left(\frac{x_i - f(x_i;\theta)}{\sigma_i}\right)^2\right]$$

$$-2\ln L(\theta) = \sum_{i=1}^{N}\left(\frac{x_i - f(x_i;\theta)}{\sigma_i}\right)^2 = \chi^2$$

So maximizing the likelihood corresponds to minimizing the χ^2. This is the *Method of Least Squares*, which is usually what people mean when they say a “chi-squared fit”.

Method of Maximum Likelihood: Uncertainty Estimates

Lore: A "good fit" has a χ^2/number of free parameters ~ 1, and for estimated parameter $\hat{\theta}$, define uncertainty $\delta\theta$ by $\chi^2(\hat{\theta} + \delta\theta) - \chi^2(\hat{\theta}) = 1$, i.e. vary the χ^2 by 1.

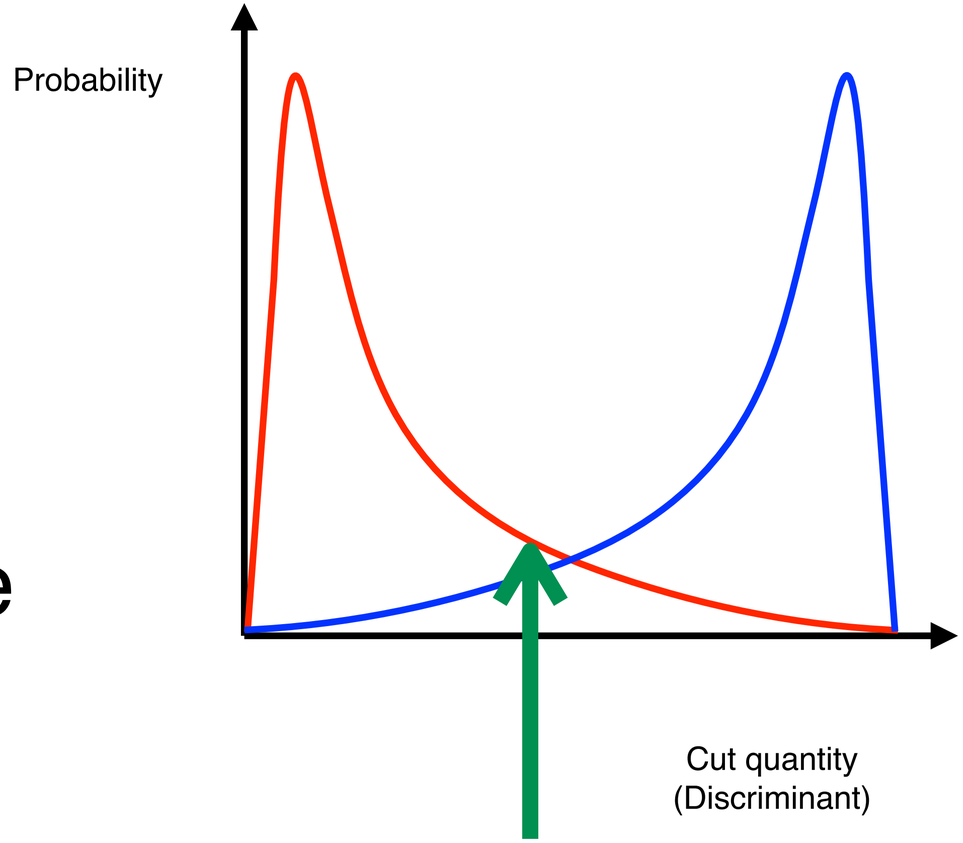
This is true insofar as the assumption of normally, independently distributed data holds. Often this is only approximately true and the χ^2 doesn't have the correct normalization, or has small deviations from the parabolic behavior near the minimum. A more general procedure is to evaluate second derivatives of the log likelihood

$$(\widehat{V}^{-1})_{ij} = -\left.\frac{\partial^2 \ln L}{\partial\theta_i \partial\theta_j}\right|_{\widehat{\boldsymbol{\theta}}}$$

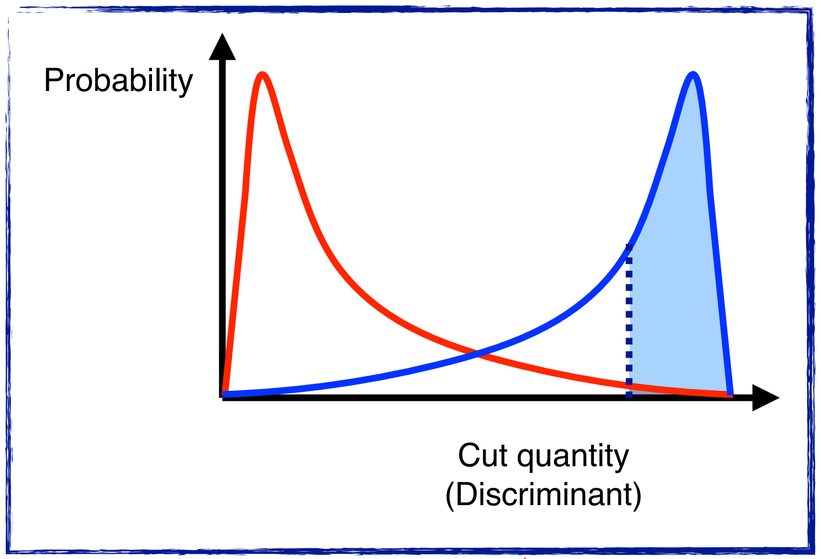
Covariance matrix (inverse)

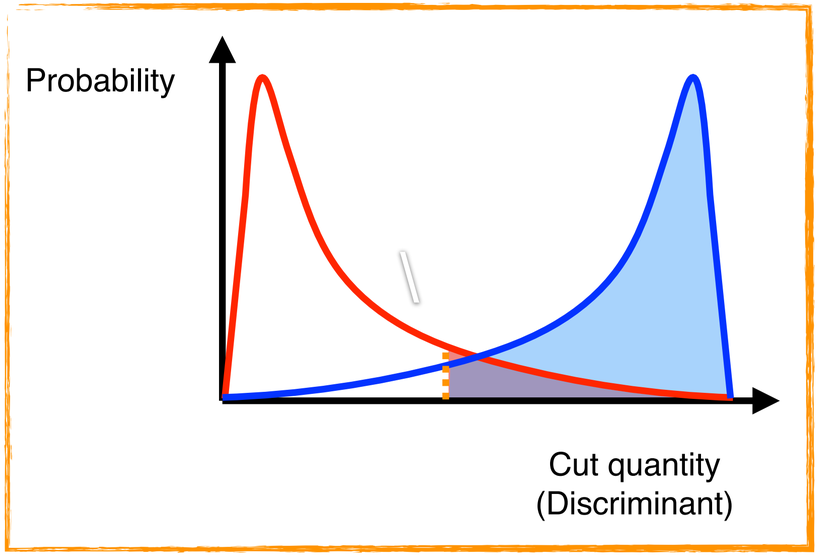
Paradigm: classification

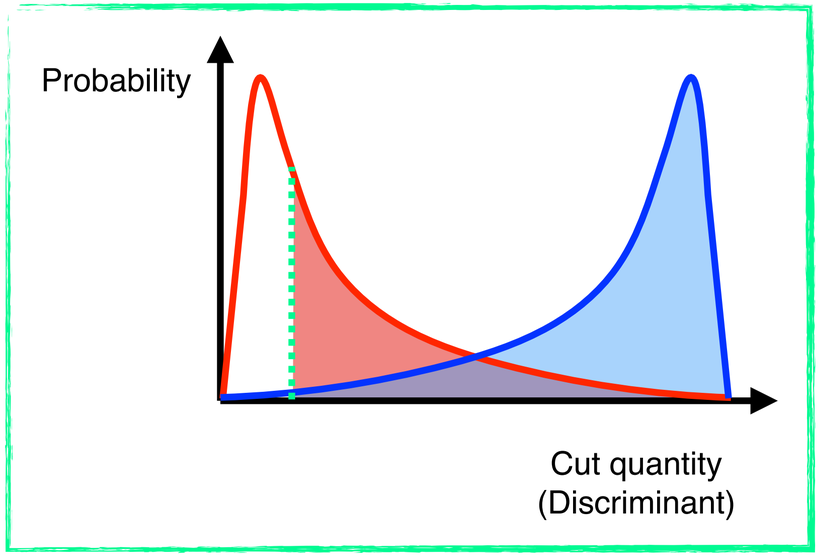
- Observed data may contain multiple contributions, some of which are interesting (signal) and some of which are not (background)
- The simplest thing we might do is use our intuition for the two cases to apply some cut on an observable to separate the events into two classes: *classification*



Paradigm: classification

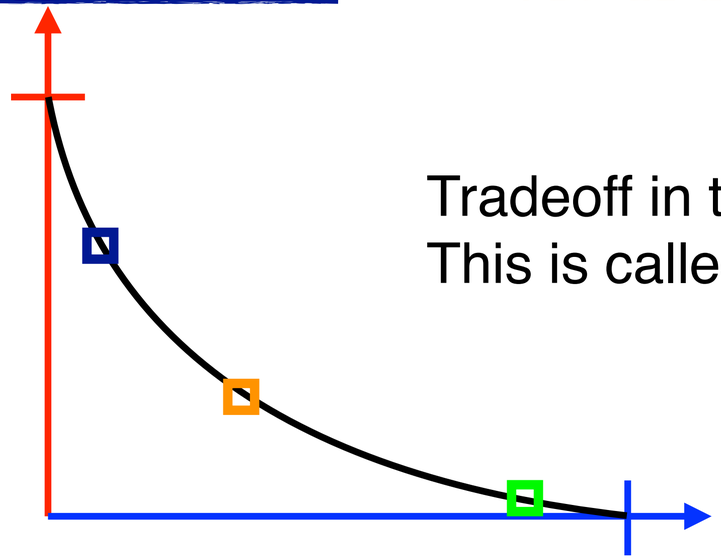






Rejection:

$$1 - \int_{\text{cut}}^{1} B(x)\mathrm{d}x$$



Tradeoff in the “cut space”
This is called a ROC curve (Receiver Operating Characteristic)

Efficiency: $\int_{\text{cut}}^{1} S(x)\mathrm{d}x$

Moving to ML

- These are established methods of analyzing data, but they have limitations, which have become increasingly limiting the more data we have access to
- They often require assumptions about the form of the data leading to biases
- These methods don’t generalize well to a higher number of dimensions
  - Computational cost and curse of dimensionality
  - Complexity in prescribing form of high-dimensional multi-variate correlations
- This makes it very difficult to make full use of what the data is telling us

Machine Learning Paradigms

- *Supervised learning* — Each element of the dataset has a label giving the "ground truth" value of the quantity you are trying to estimate, e.g. dataset has form $\{x_i, y_i\}_{i=1,\ldots,N}$ and we wish to learn $f(x) = y$
  - Examples: regression and classification

- *Unsupervised learning* — No labels! dataset has form $\{x_i\}_{i=1,\ldots,N}$ and we are trying to learn something else, usually about the data itself
  - Examples:
    - Learning a representation of the data (possibly lower dimensional)
    - Clustering and data mining
    - Learning the (probability) density distribution $p(x)$

- *Reinforcement learning* — Formulated to aid in decision making or "optimal control" problems. Labels are not needed, but a "reward function" (provided by user) is used to dynamically contextualize the model output
  - Won't cover this here except to say there is a tradeoff between exploitation of current knowledge and exploration of new possible choices.

The neural network

- In the context of a fit, we usually try to make our model more sophisticated by adding terms
- In a neural network we do this by repeated function composition $a^{(n)}(a^{(n-1)}(\cdots a^{(0)}(x)\cdots)$
- The functions we use are called *activation functions*
  - These are usually taken from a class of non-linear functions with desirably smoothness properties, e.g. Tanh or logistic function
- Repeated composition, having a deep neural network, enables the model capture non-linear behavior
  - Models may have many hidden layers
- Neural networks are exceptionally powerful at capturing (possibly non-linear) correlations between multiple variables

Common activation functions

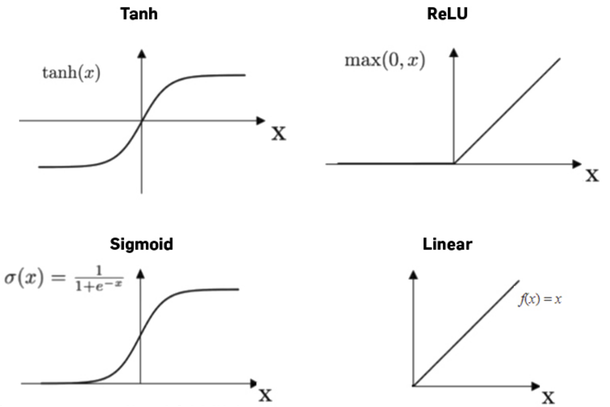


Image credit: AI Wiki

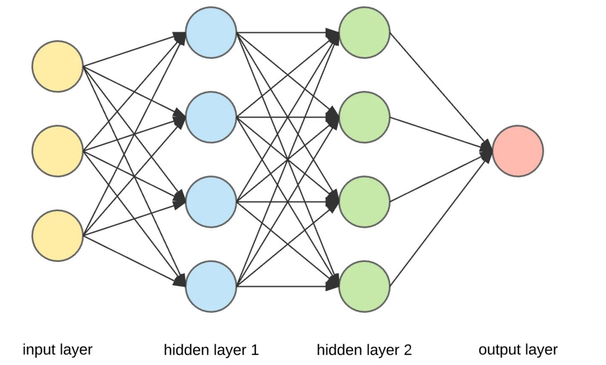
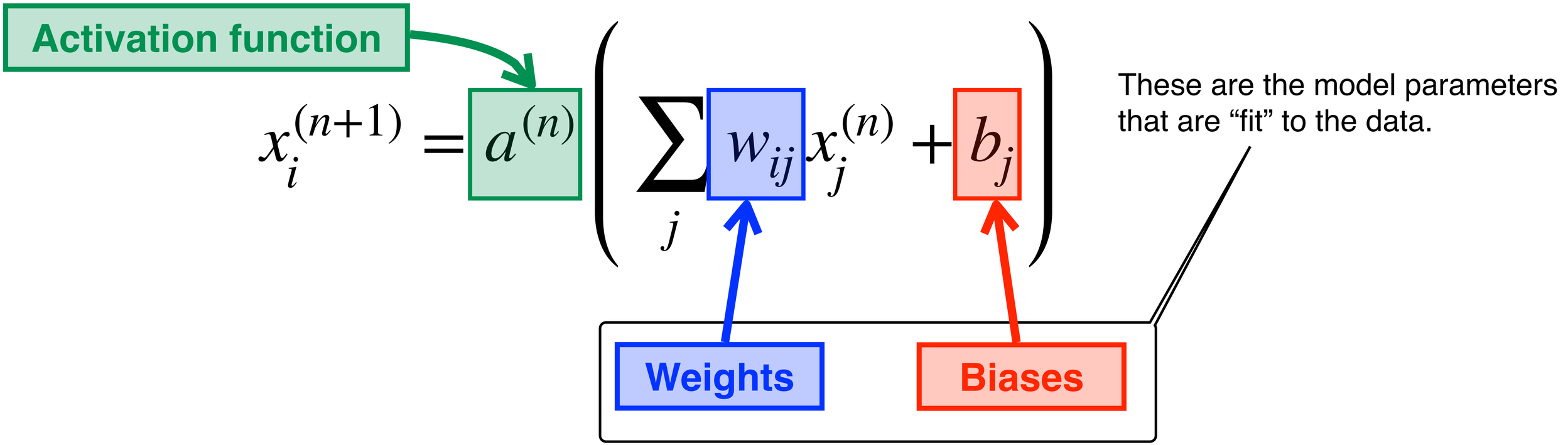


Image credit: Towards Data Science

The neural network



The full sequence of function compositions is directly differentiable since activation functions are known, one can recursively apply the Chain Rule

Practical anatomy of training an ML model

- The model: $f_\theta(x)$
  - Parameters (weights and biases) collectively referred to as "ϕ", together with activation functions
- The loss: $\mathscr{L}(f_\theta(x), y)$
  - This effectively plays the role of the χ^2 or $\ln L$ in our previous examples
  - In some cases the loss may be one of these quantities, $\mathscr{L}(f_\theta(x), y) = (y - f_\theta(x))^2$
  - Another common form is the MAE (mean absolute error)
  - For classification problems one uses the cross entropy $\mathscr{L}(f_\theta(x), y) = \sum_{c \in \text{classes}} \mathbf{1}(y = c) \ln f_\theta(x)$
- *Gradient Descent*: we compute the gradients of the loss and update the parameter values according to $\theta \to \theta - \lambda \nabla_\theta \mathscr{L}(f_\theta(x), y)$
  - λ is called the *learning rate*, as this procedure is repeated iteratively, the value λ can be adjusted according to a schedule to improve convergence
  - This process is called *back propagation*

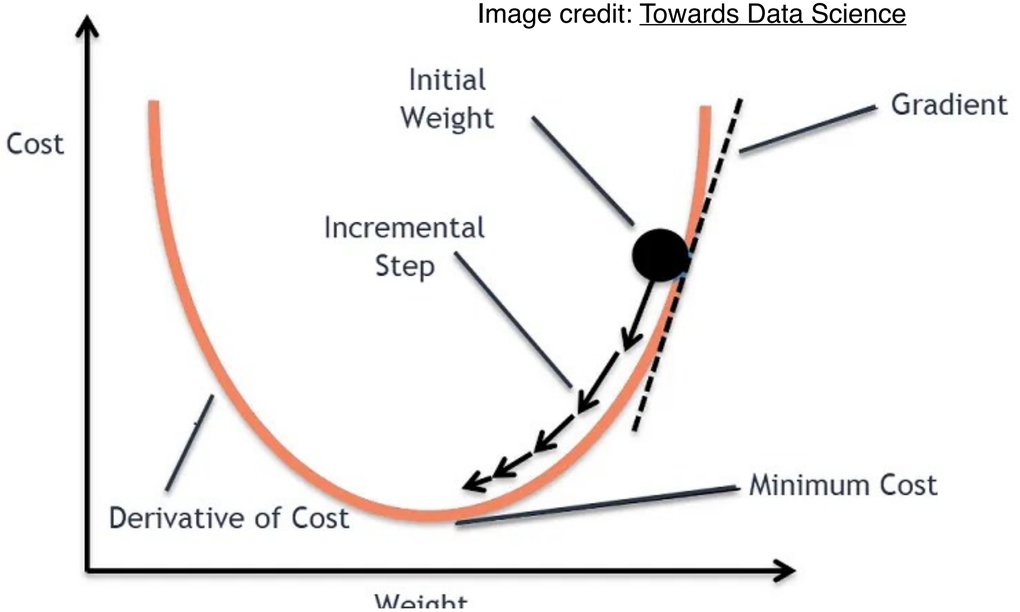
Some common problems: network has too many parameters

- The model is too general/complicated
  - Deep networks are very expressive, but can have millions of parameters
  - Can lead to instability and difficulty finding minimum
  - Can also be highly computationally expensive
- Solution: introduce restrictions on which nodes are connected, effectively fixing weights equal to zero
  - Can dramatically reduce number of parameters, e.g. convolutional NN
  - Potential limitations introduced by this and also by the choice of activation function are referred to as *inductive biases*

Some common problems: gradient descent is inefficient

- To improve the efficiency of the calculation we use *Stochastic Gradient Descent* (SGD) which only computes the gradients averaged over a subsample of the data, usually these samples are called *minibatches*
  - To stabilize the procedure one may further extend the gradient-update step using an *optimizer*.

Image credit: Towards Data Science



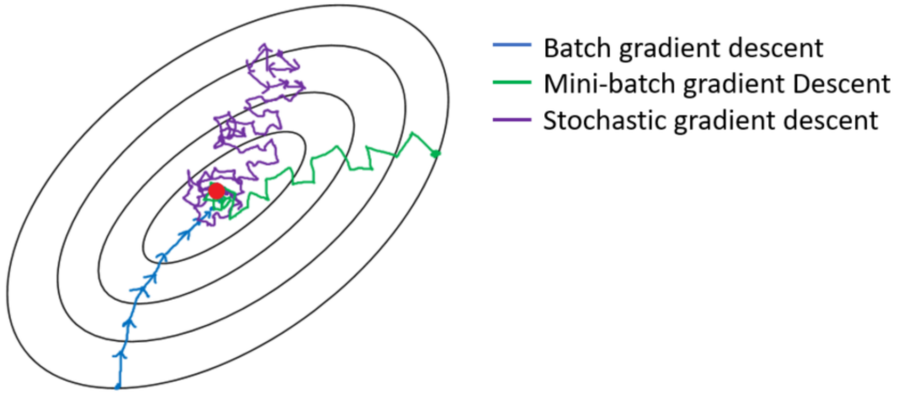
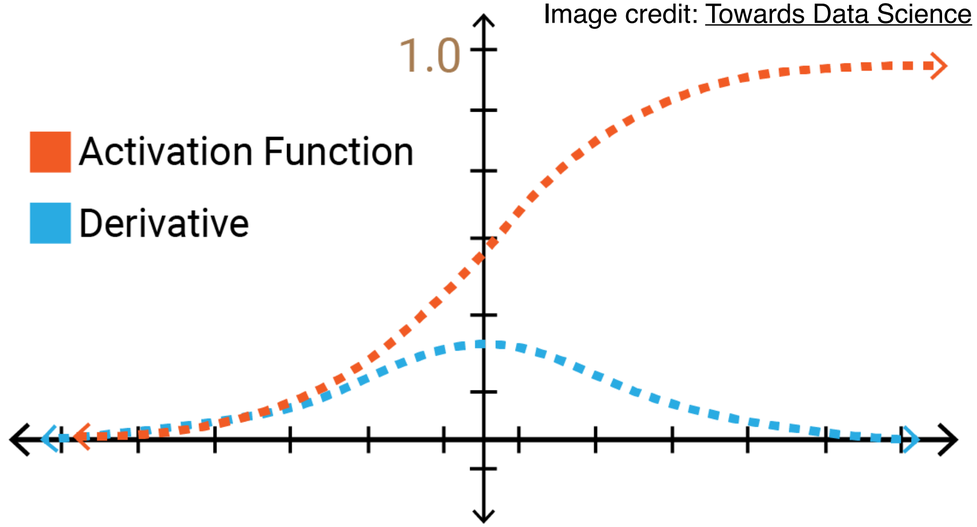


Image credit https://sweta-nit.medium.com/batch-mini-batch-and-stochastic-gradient-descent-e9bc4cacd461

Some common problems: vanishing gradients

- Many activation functions map a large range of inputs to a small range of outputs, i.e. they are flat over a large part of their domain. This results in small gradients.
- In a shallow network this isn't a huge deal, but in deeper networks it is a problem as the problem compounds itself each step you go back in the backpropgation
- This leads to very small gradients for parameters in the early layers, making it difficult to update them effectively
- Solutions include:
  - Changing the activation function for example ReLU
  - Introducing residual connections where output from earlier layers is concatenated with layer ones, bypassing the cumulative effect of the activation
  - Batch normalization: standardizing outputs so tails of activation are de-emphasized

Image credit: Towards Data Science



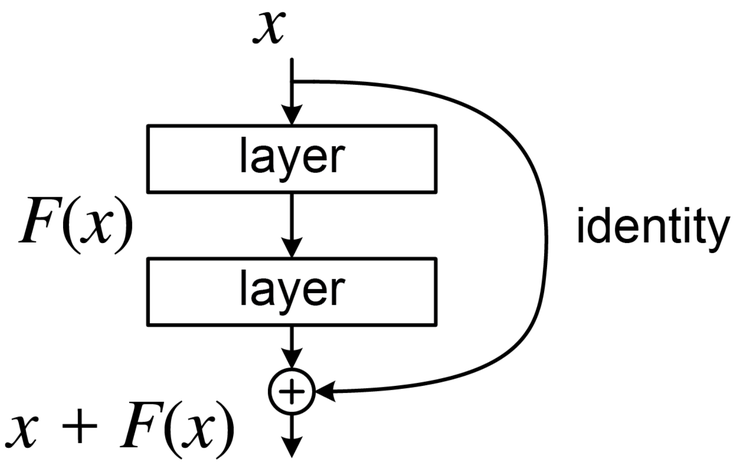
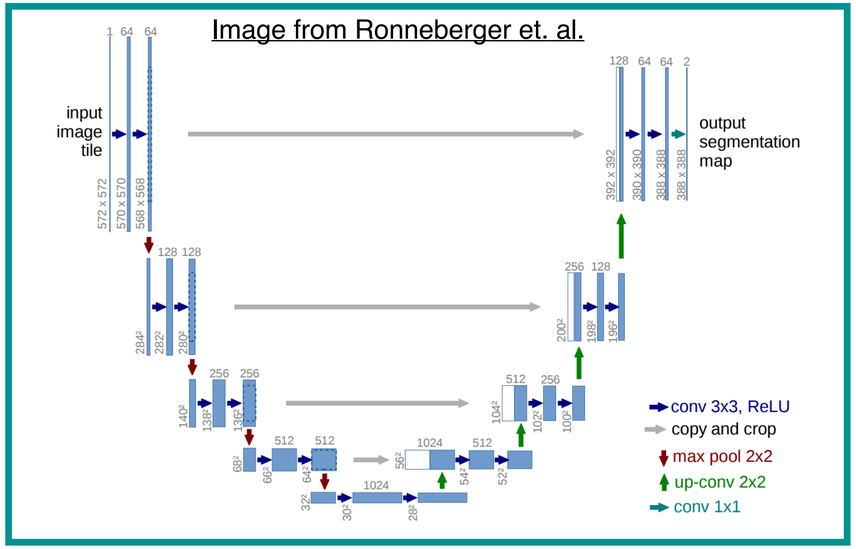


Image credit: Wikipedia— Residual Neural Network

Some common network architectures

- Dense or fully connected
  - Most general form which can be good or bad
  - Potentially unbiased, but lots of parameters
- Convolutional neural networks (CNNs)
  - Only “nearby” neurons are combined
  - Neighborhood and weighting determined by a filter
  - Filter weights are learned parameters and help identify features
  - Common in image processing, computer vision
  - Tasks: image recognition, object identification, semantic segmentation
- U-Net
  - Specific convolutional architecture exploiting residual connections



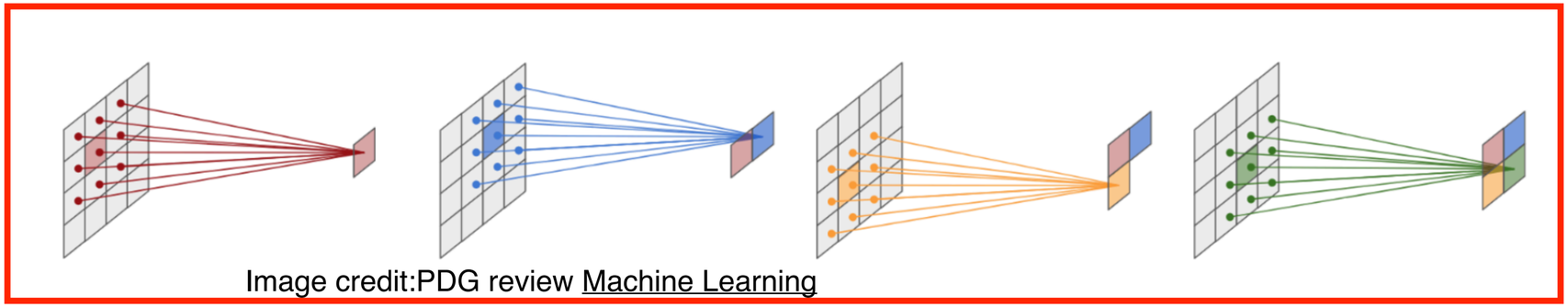


Image credit:PDG review Machine Learning

Generative Models

- Super hot topic right now (chat GPT)
- *Generative*: model output looks like real data
- Properties:
  - Generate samples, i.e. produce $x_i \sim p(x)$
  - Evaluate the density, provide values of $p(x)$ for given x whether that x value was used in the training sample or not
- Examples:
  - Variational Auto-Encoders (VAEs)
  - Generative Adversarial Networks (GANs)
  - Diffusion Models and Normalizing Flows

Highlighted application: surrogates

- You have a problem involving an operation that is computationally expensive and is used repeatedly
  - Example: a function evaluation in a minimization procedure or a fit, often due to high dimensionality
- Train a fast surrogate model to approximate it
  - Evaluating a likelihood $P(x \mid \theta)$ where x and/or θ are high dimensional or the evaluation is simply slow
- Since generative surrogate is a DNN it will be smooth, differentiable and invertible
  - Monte Carlo simulations are stochastic and generally not differentiable, but their surrogates are so you can take derivatives and/or invert the surrogate model

Generative Adversarial Networks (GANs)

- Model has two components
  - Generator: takes random inputs and outputs something with the same format/structure as the input data, e.g. an image
  - Discriminator: takes images and classifies them as real or fake
- Train the discriminator to distinguish between real and fake, but train the generator to fool the discriminator
  - Example of a zero-sum game
- GANs are:
  - Very powerful
  - Generate samples very fast
  - Notoriously hard to train
  - Tons of GAN variants out there

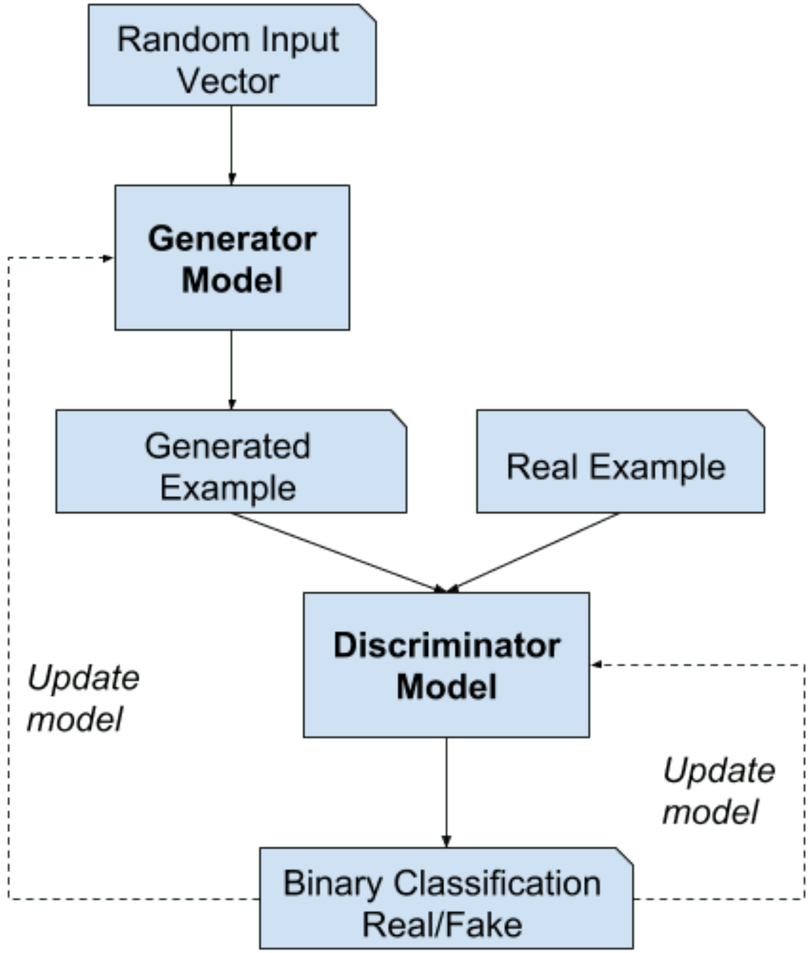


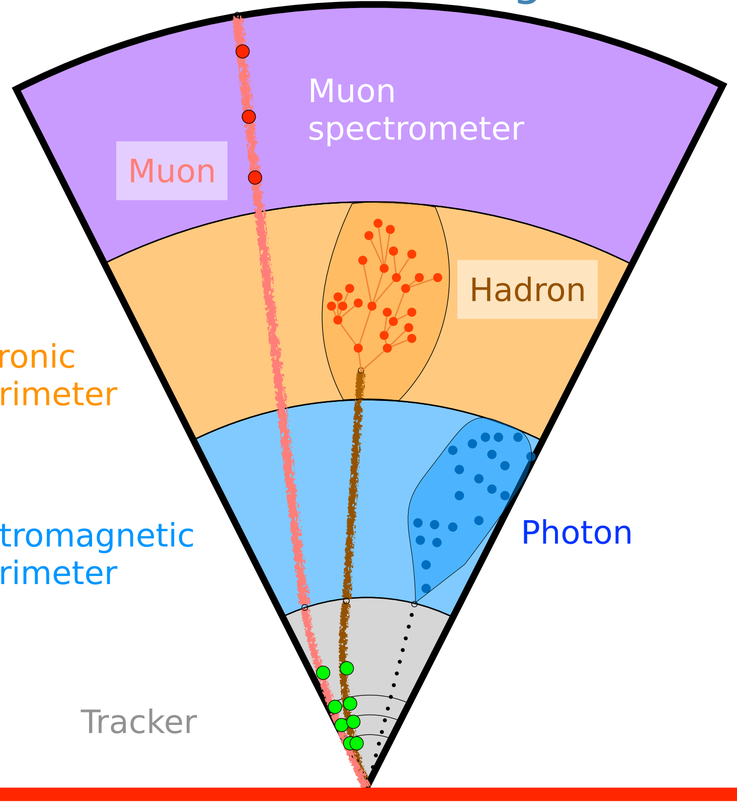
Image from Machine Learning Mastery

Applications

- The following examples are taken from my own research, but there are tons of examples and reviews from high energy physics here

The ATLAS Experiment

Raw detector signals



Reconstruction

Primitives
Calorimeter cells
Track hits

→

Low-level objects
Tracks
Clusters

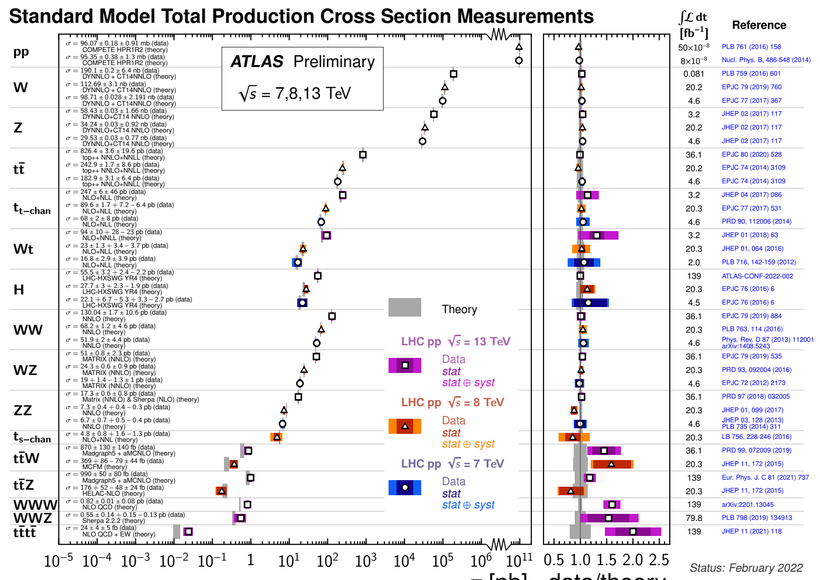
→

Physics Objects
Muons
Electrons
Photons
Hadrons
Jets

Physics Analysis

Experimental corrections
Background removal
Evaluation of systematics
Statistical modeling

Results

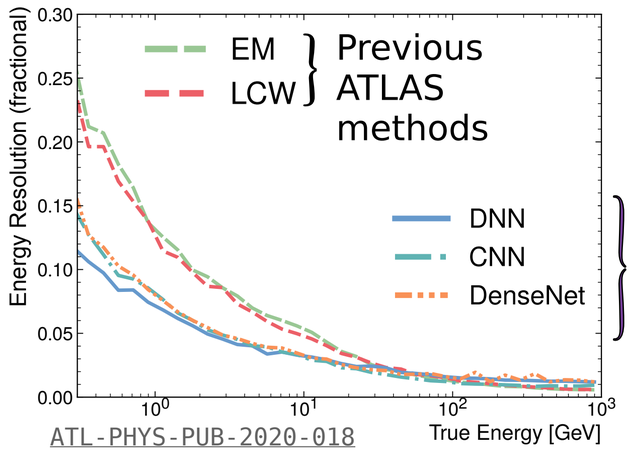


Calorimetric Particle Reconstruction: Problem Statement

- We measure the energy of particles using calorimeters
  - Material initiates electromagnetic or hadronic shower
  - Single high-energy particle → many low energy particles
- Problem: given a group of measured energy deposits from the shower, assign an energy (regression) and type (classification) to the shower that can be used in physics analyses
- Challenges:
  - Not all energy is measured
  - Deposited subject to stochastic fluctuations
  - Details of these effects depend on incident particle energy and type (electromagnetic vs hadronic)
- Features:
  - Physics of shower is ~well known and implemented in highly detailed and validated detector simulation
  - Very fine, three-dimensional segmentation of calorimeters (O(100) “voxels”) per shower, but in default reconstruction only a few reduced quantities are used to reconstruct particle energies

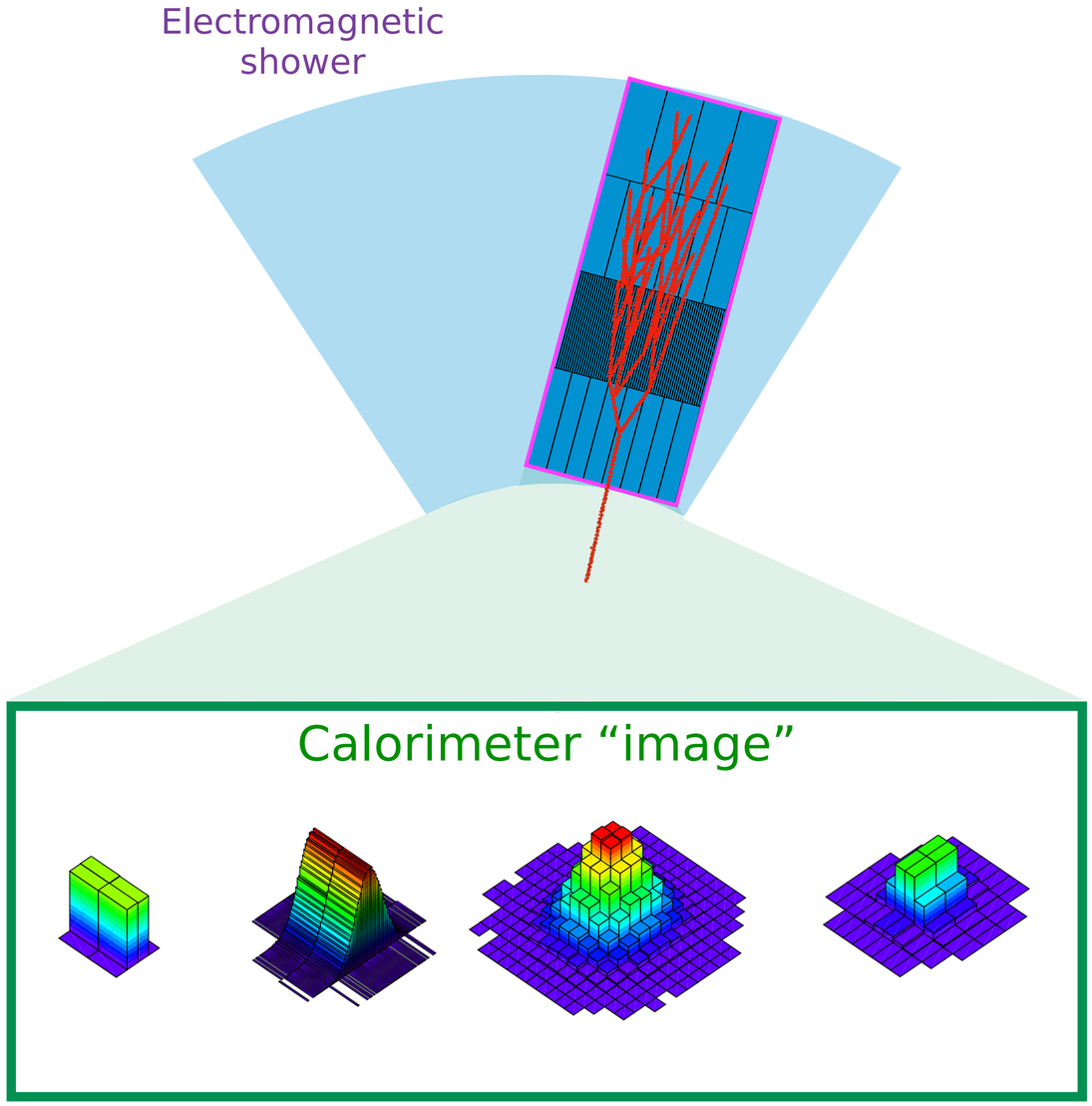
Calorimetric Particle Reconstruction: ML Approach

- Started with convolutional networks
  - Calorimeter system has multiple “layers” kind of like a “stacked” RGB image
  - Major improvement to energy scale/ resolution
- Requires all pixels (cells) in a given layer are the same “size”
  - Condition only satisfied for central region of detector (~25% of acceptance)



ATL-PHYS-PUB-2020-018

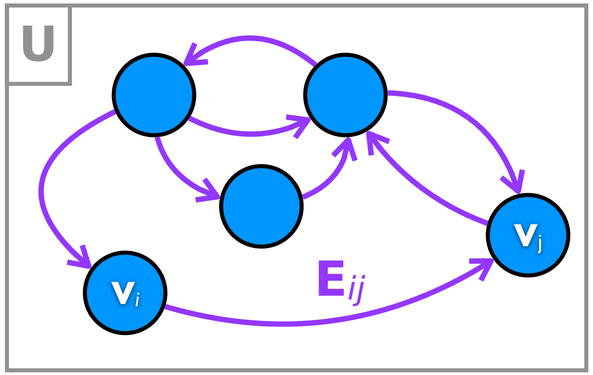
ML Reconstruction



Calorimetric Particle Reconstruction: ML Approach

- Requires all pixels (cells) in a given layer are the same “size”
  - Condition only satisfied for central region of detector (~25% of acceptance)

Addressed by graph neural networks

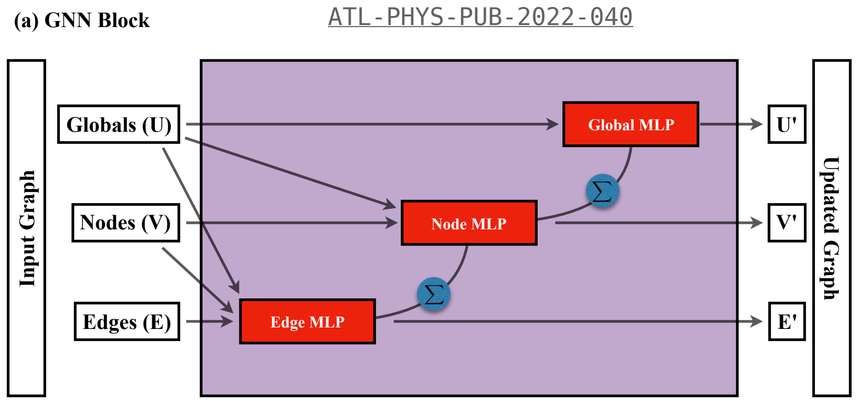


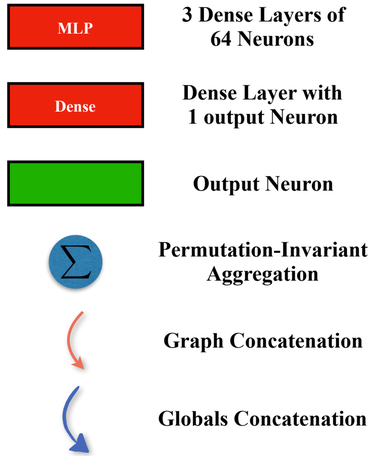
Globals (**U**) information about the entire graph (total energy)

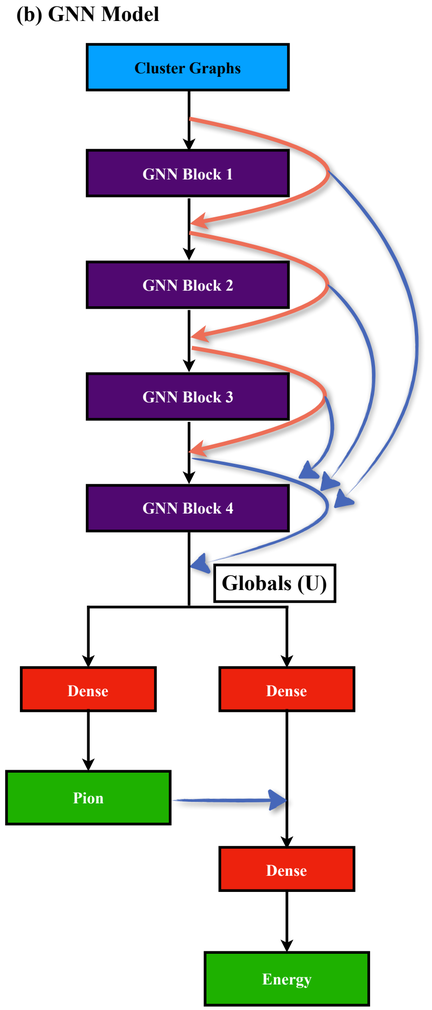
Nodes ($\mathbf{V}_i$) ↔ cells
Features: energy, coordinates, size etc.

Edges ($\mathbf{E}_{ij}$)
Features: type of neighbor

ATL-PHYS-PUB-2022-040

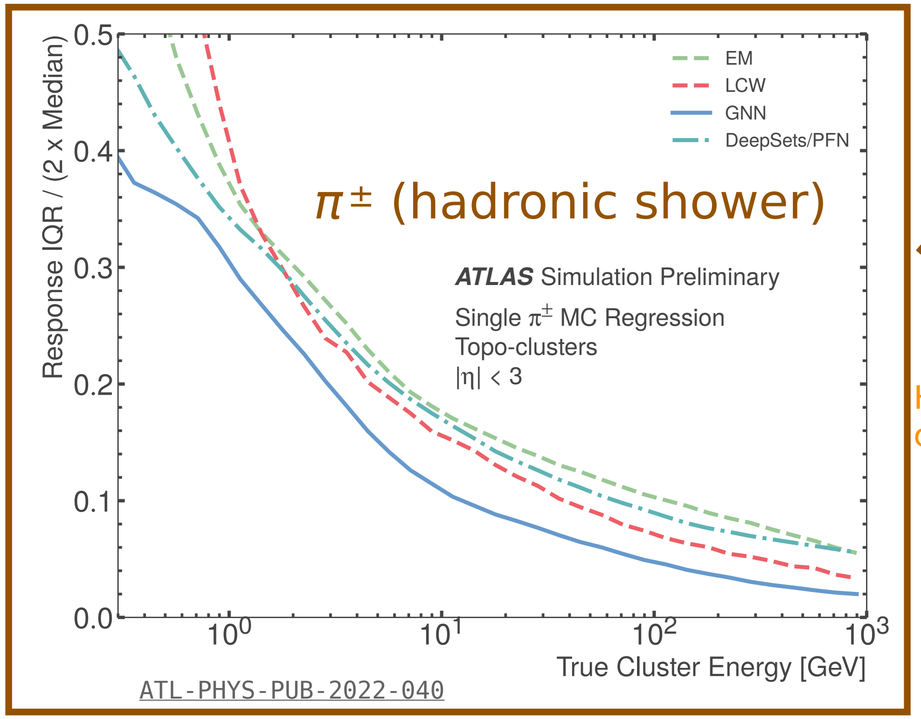


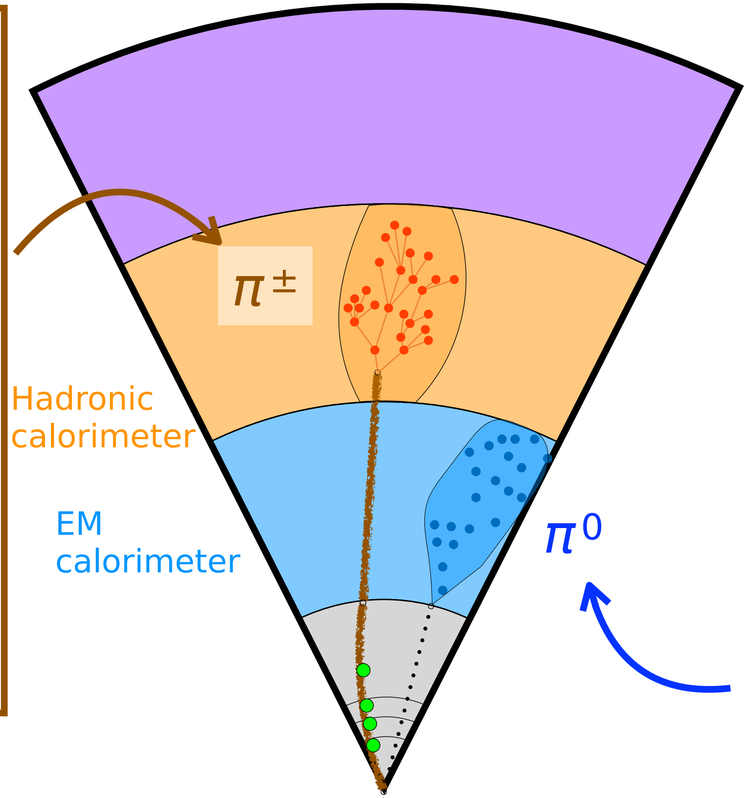


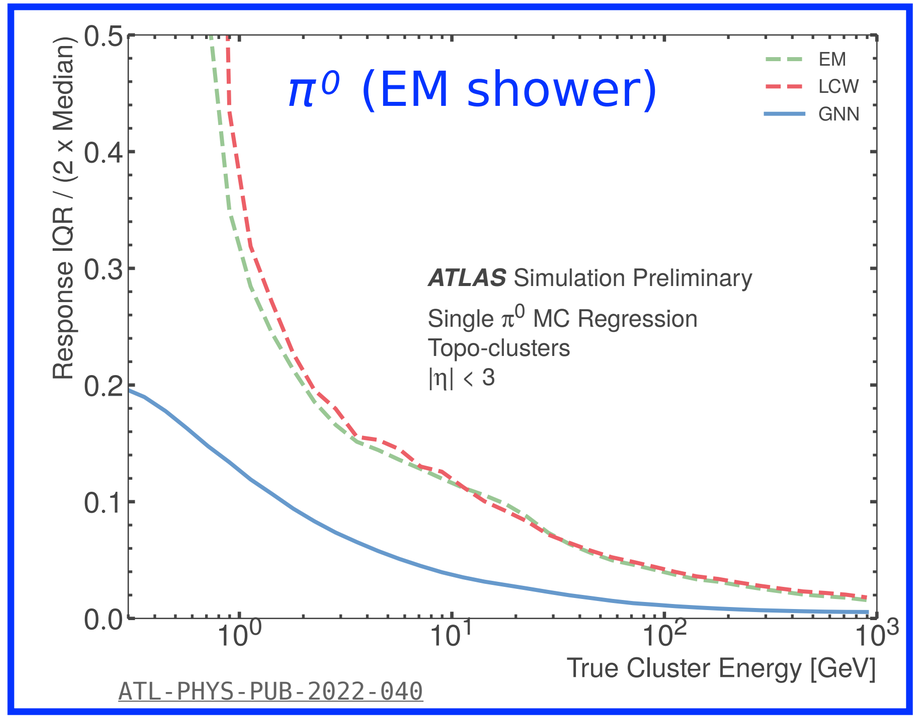


Terminology from DeepMind's paper:
arXiv:1806.01261 [cs.LG]
and built with GraphNets library

Calorimetric Particle Reconstruction: ML Approach



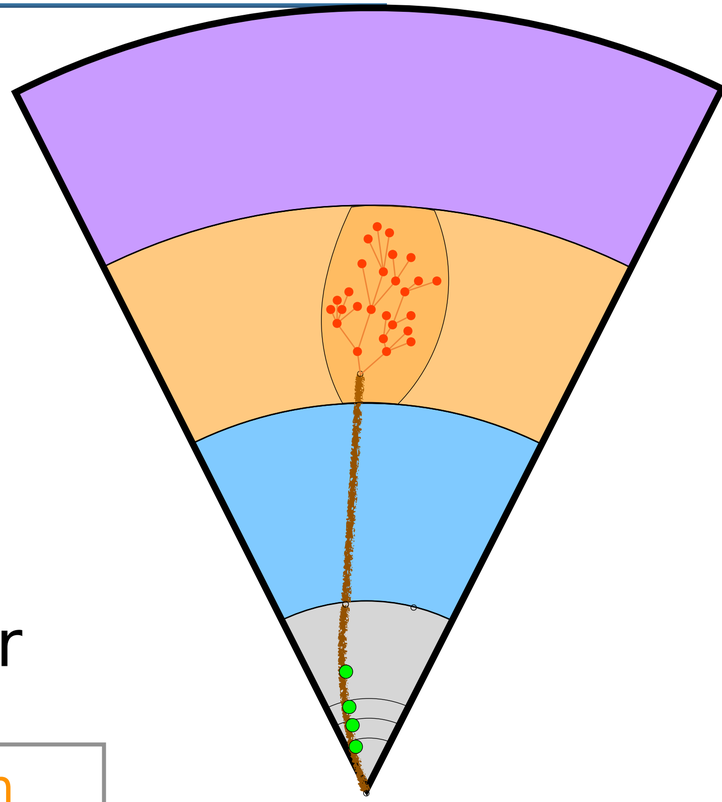


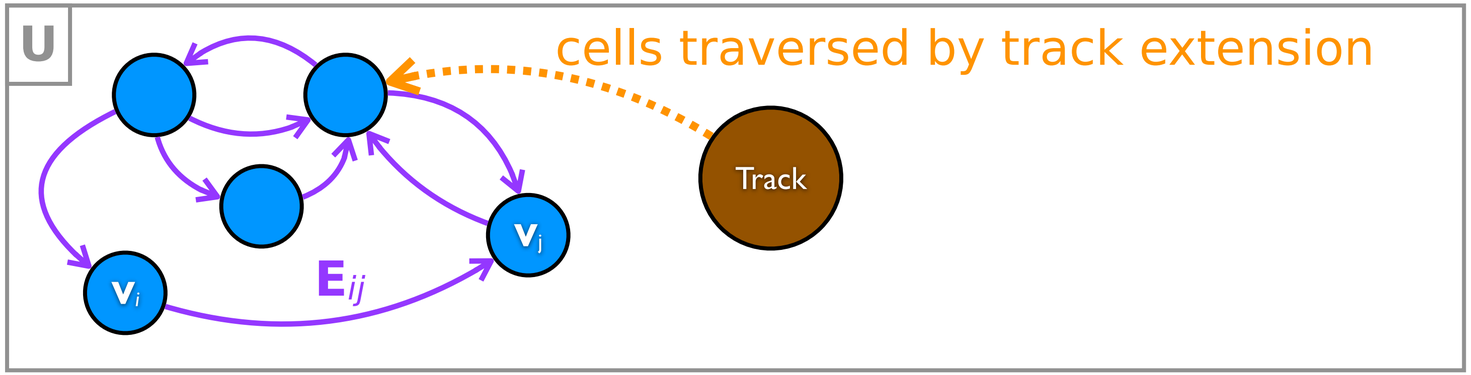


Improved performance wrt CNNs, and now applicable to entire detector

Calorimetric Particle Reconstruction: Future Directions

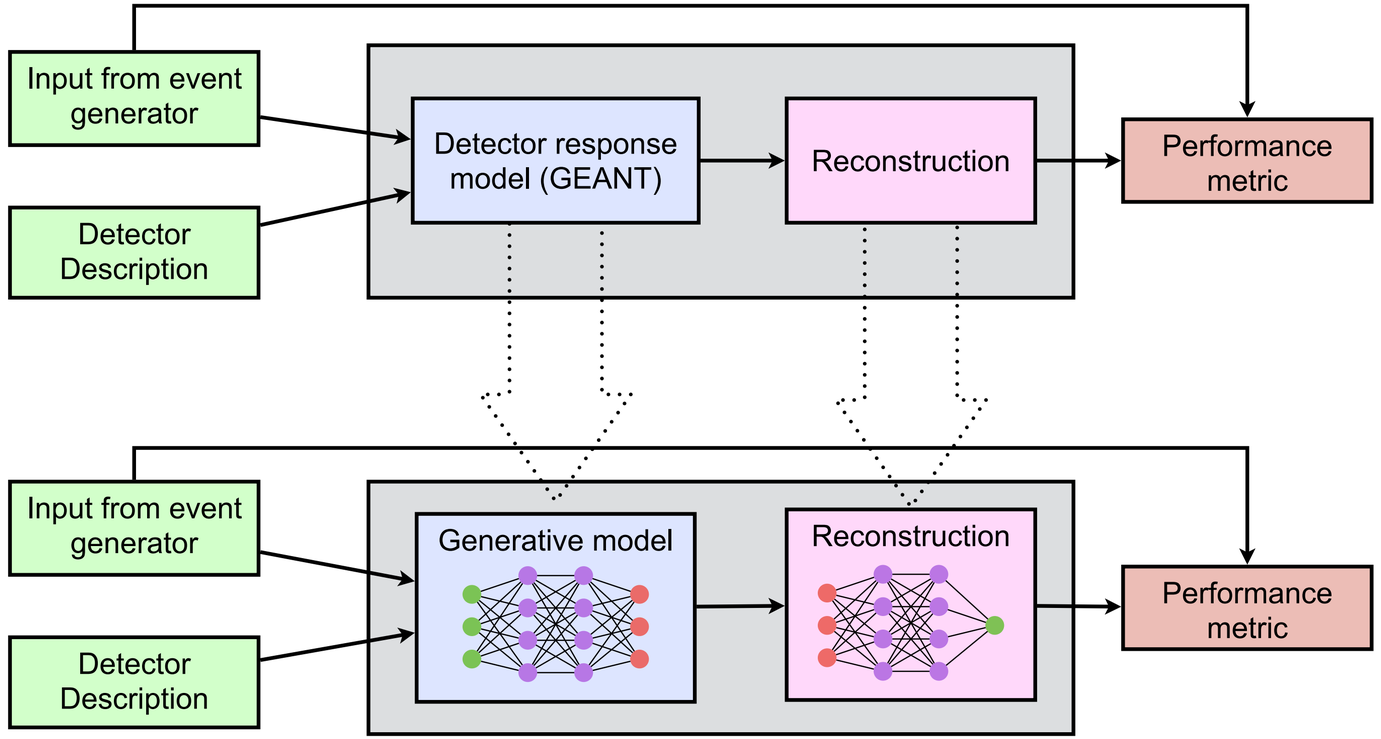
- Studies have been mostly proof of principle
  - Not yet used in physics analysis
  - Deploy in ATLAS reconstruction
- Apply similar methods to other areas of reconstruction
  - Example: include information from tracker





EIC Detector Design: ML approach

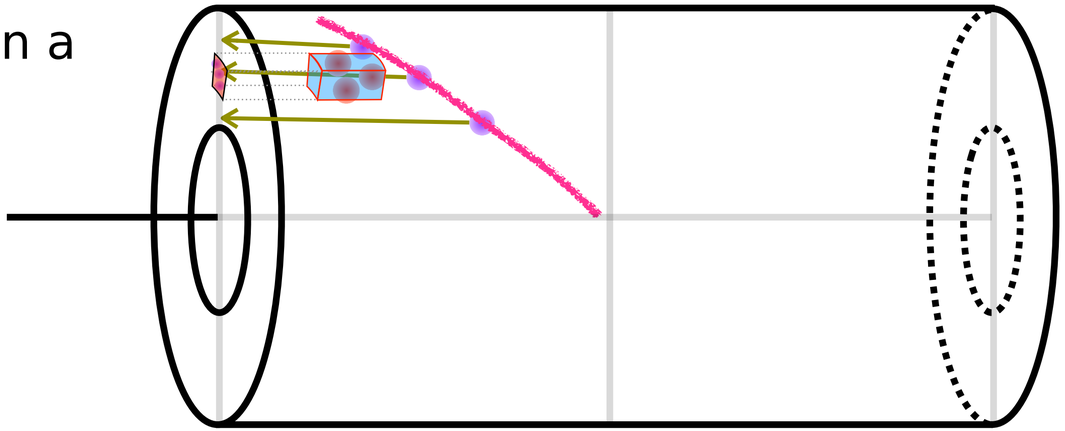
- Replace detector simulations with generative models, trained conditionally on detector parameters
  - Fast surrogate that is *differentiable*
- Also train reconstruction model



- Use differentiability to visualize gradients to have quantitative input into trade off
- Provide fast surrogate simulation for optimal detector configuration and co-optimized reconstruction for that configuration

Tracking with a TPC: Problem Statement and ML approach

- Measure (charged) particle momenta by bending them in a magnetic field and measure the track trajectory
- Track ionizes gas which drifts to pads on end-plate according to applied electric field
- Position and time at pad sensor → 3D coordinates of hit
- Hits used to reconstruct trajectory → momentum



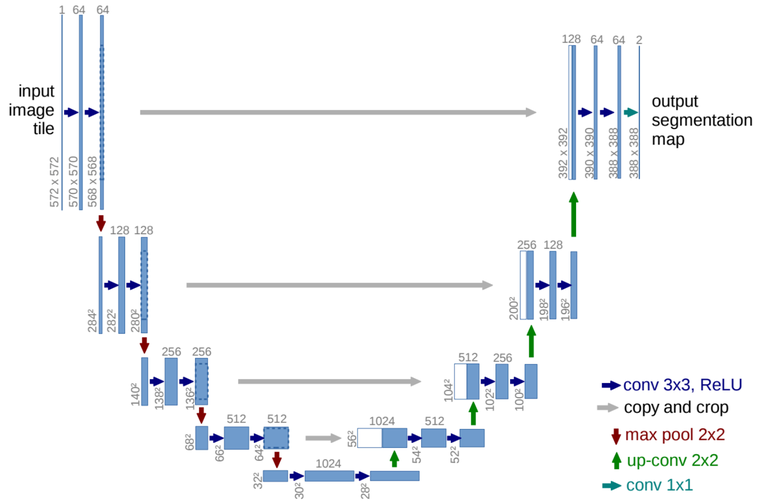


Image from Ronneberger et. al.

- Charge multiplication on pads produces ions which flow backwards
  - "Ion backflow" distorts local electric field
  - Hits need distortion corrections
- Space charge is periodically measured, but applying correction to propagation is extremely resource intensive
  - Train fast ML model that can replace this calculation
  - Using U-Net architecture, good for "pixel-wise regression"
- Major speedup observed using simulations with tolerable errors
- See report from ALICE Experiment