Let Machine remember the past and simulate physics

Xiangyang Ju

Lawrence Berkeley National Laboratory

Physics 290E, Spring 2022

Success of deep learning

Computer vision (CNNs)

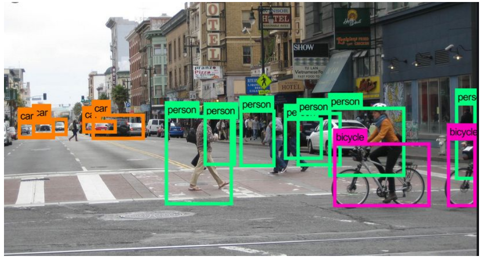
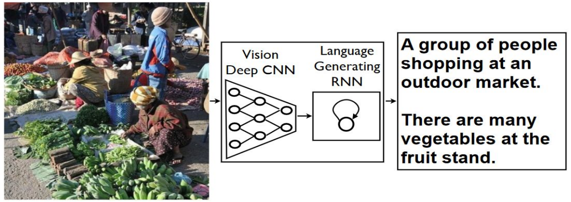
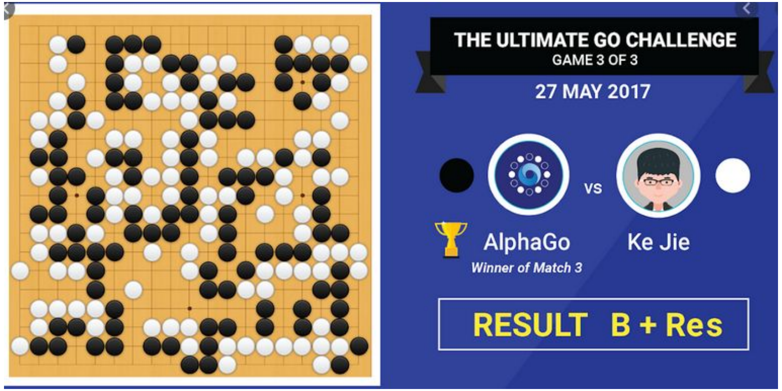


Image processing + language processing arXiv:1411.4555



Gaming
(via deep reinforcement learning)



Three driving factors:

1. Algorithmic innovation
2. Data
3. Amount of compute available for training

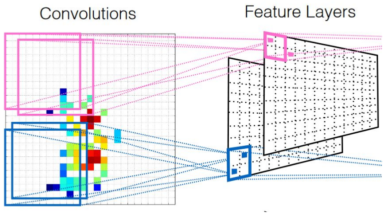
Data representation and tools

Data is a vector
→ multilayer perceptrons (MLPs)

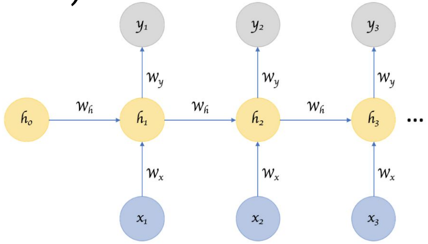
$$\begin{pmatrix} x_1 & x_2 & \cdots & x_n \end{pmatrix} \times \begin{pmatrix} w_{11} & w_{12} & \cdots & w_{1m} \\ w_{21} & w_{22} & \cdots & w_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ w_{n1} & w_{m2} & \cdots & w_{nm} \end{pmatrix}$$

Data is an image or grid
→ Convolutional Neural Network (CNNs)

arXiv:1511.05190



Data is a sequence
→ Recurrent Neural Network (RNNs)



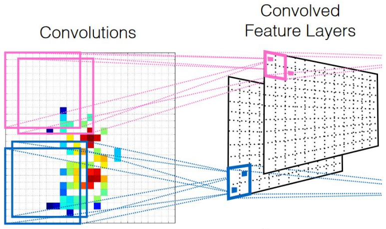
Data representation and tools

Data is a vector
→ multilayer perceptrons (MLPs)

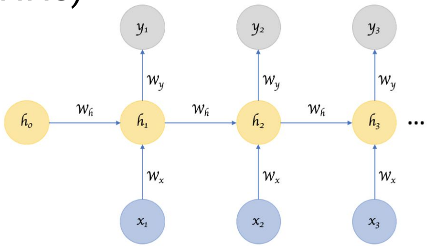
$$\begin{pmatrix} x_1 & x_2 & \cdots & x_n \end{pmatrix} \times \begin{pmatrix} w_{11} & w_{12} & \cdots & w_{1m} \\ w_{21} & w_{22} & \cdots & w_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ w_{n1} & w_{m2} & \cdots & w_{nm} \end{pmatrix}$$

Data is an image or grid
→ Convolutional Neural Network (CNNs)

arXiv:1511.05190

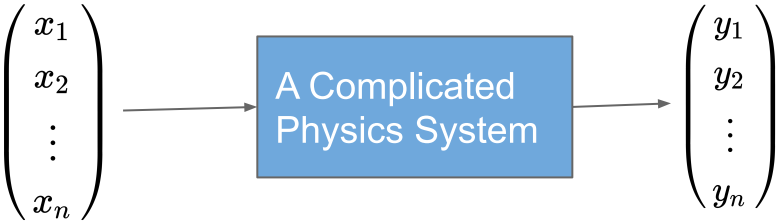


Data is a sequence
→ Recurrent Neural Network (RNNs)



- What is Neural Network (NN)?
- How to “train” a neural network?
- How to make a neural network remember?
- How does neural network simulate particle physics?

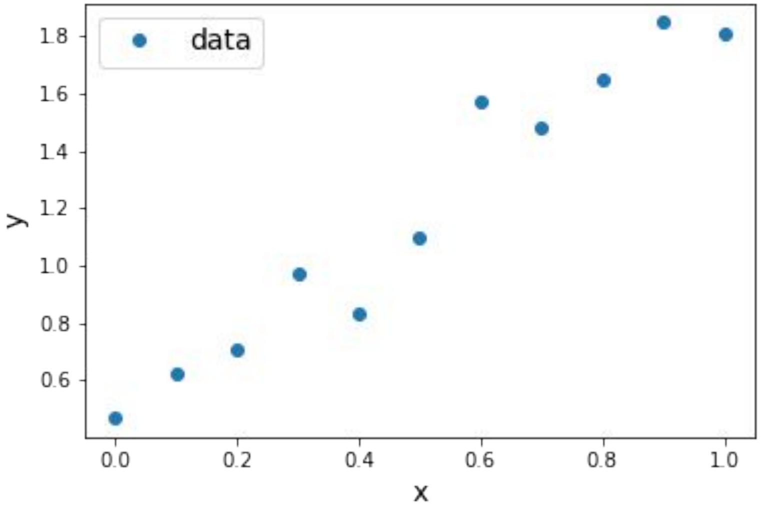
A regression task



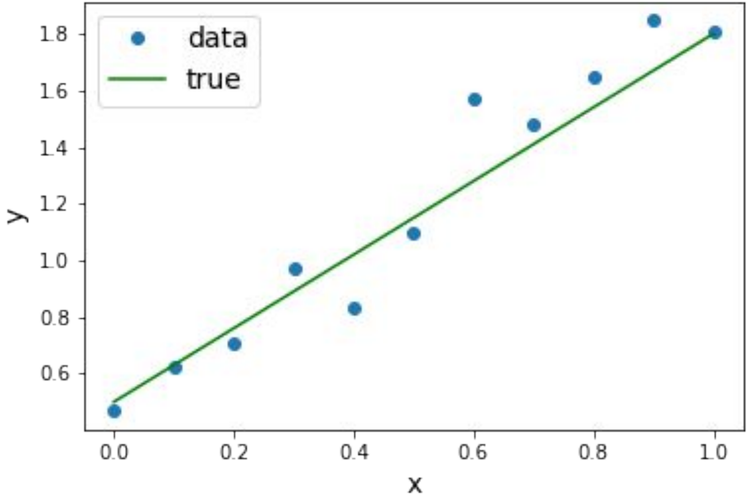
A complicated physics system takes a input ***x*** and produces an output ***y***, and it is ***time consuming*** for the system to calculate ***y***.

Now the task is to develop a ML model to “simulate” the system.

Experimental data collected for different inputs.



Solve the regression task



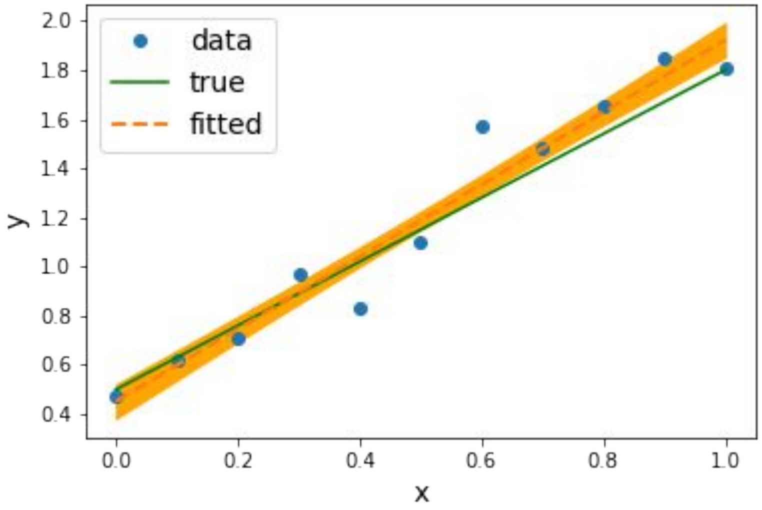
ML model: $f(x; w, b) = w \cdot x + b$

In a matrix form

$$f(x; w, b) = \begin{pmatrix} x_1 & 1 \\ x_2 & 1 \\ \vdots & \vdots \\ x_n & 1 \end{pmatrix} \times \begin{pmatrix} w & b \end{pmatrix}$$

Q: how would you estimate the weights (***w***) and bias (***b***)?

Least squares



ML model: $\boxed{f(x; w, b) = w \cdot x + b}$

In a matrix form

$$\boxed{f(x; w, b) = \begin{pmatrix} x_1 & 1 \\ x_2 & 1 \\ \vdots & \vdots \\ x_n & 1 \end{pmatrix} \times \begin{pmatrix} w & b \end{pmatrix}}$$

Q: how would you estimate the weights (***w***) and bias (***b***)?

Linear Least squares.

In addition, it estimates uncertainties associated with the weights and biases! [code]

What is the loss function and how did we minimize the loss function?

Loss functions

In case of Least Squares method, the loss function is uniquely defined:

$$\mathcal{L} = \sum_{i=1}^{n} (y_i - f(x_i; w))^2 = ||y - f(x; w)||^2$$

However, in ML, the loss function takes various forms:

- Log-loss for binary classification
- ||▪||^n, where n can be 1, 2, or other integers
- and whatnot

$$LogLoss = -\frac{1}{n}\sum_{i=0}^{n}[y_i log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)]$$

The minimum value of the loss L occurs when all gradient are zero.

$$\frac{\partial \mathcal{L}}{\partial W} = 0$$

$$2\sum_{i=1}^{n} (y_i - f(x_i; w)) \cdot \frac{\partial f_i}{\partial W} = 0$$

If there are ***m*** (trainable) parameters in the model, an optimizer (optimization algorithm) needs to find a set of parameters that simultaneously satisfy the ***m*** equations.

Backpropagation

Backpropagation is to use the “chain of rules” to calculate the gradients of the loss function w.r.t trainable parameters in the NN.

$$\mathcal{L} = \sum_{i=1}^{n}(y_i - f(x_i; w))^2 = ||y - f(x; w)||^2$$

We rewrite this function as:

$$\mathcal{L} = \sum_{i=1}^{n}(y_i - o_i)^2$$

$$o_i = f(x_i; w)$$

$$\frac{\partial \mathcal{L}}{\partial w} = \frac{\partial L}{\partial o} \cdot \frac{\partial o}{\partial w}$$

Model ***f*** has to be differentiable w.r.t its parameters.

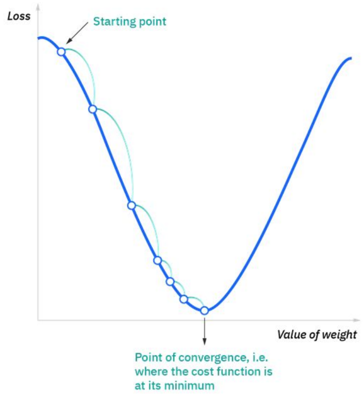
Gradient descent

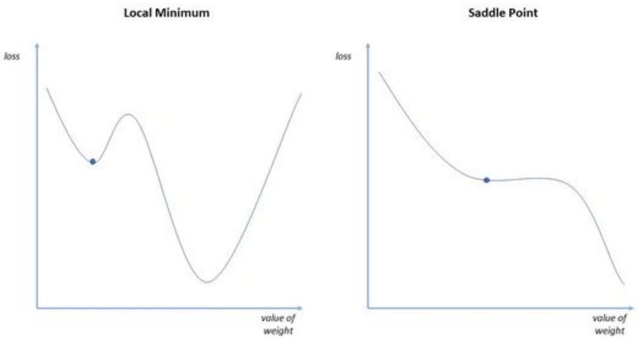
The minimum value of the loss L occurs when all gradients of are zero.

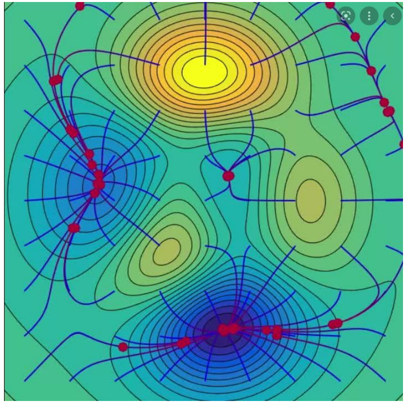
The loss value decreases fastest if its parameters go in the direction of the negative gradient of the loss function

$$w_{n+1} = w_n - \gamma \nabla \mathcal{L}$$

γ is the step size or learning rate.







Optimizers

A optimizer defines how to update the trainable parameters so that the gradient of the loss function w.r.t each trainable parameters is at minimum.

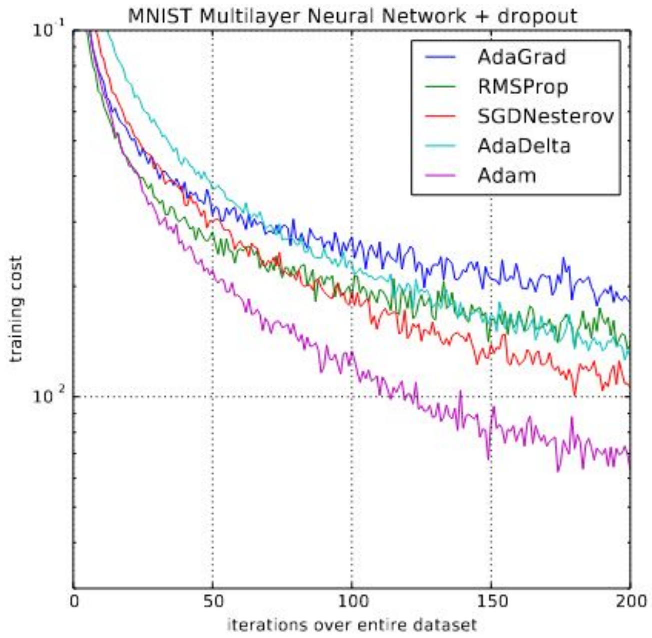
Adam: A method for stochastic optimization, arxiv:1412.6980. It is designed to minimize averaged gradients (average over gradients of all trainable parameters)

Adam

Require: α: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ_0: Initial parameter vector
  $m_0 \leftarrow 0$ (Initialize 1st moment vector)
  $v_0 \leftarrow 0$ (Initialize 2nd moment vector)
  $t \leftarrow 0$ (Initialize timestep)
  while θ_t not converged **do**
    $t \leftarrow t + 1$
    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)
    $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
    $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
    $\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
    $\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
    $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
  end while
  return θ_t (Resulting parameters)

a: learning rate, 0.001
β_1: decay rate of gradient, 0.9
β_2: decay rate of squared gradient, 0.999

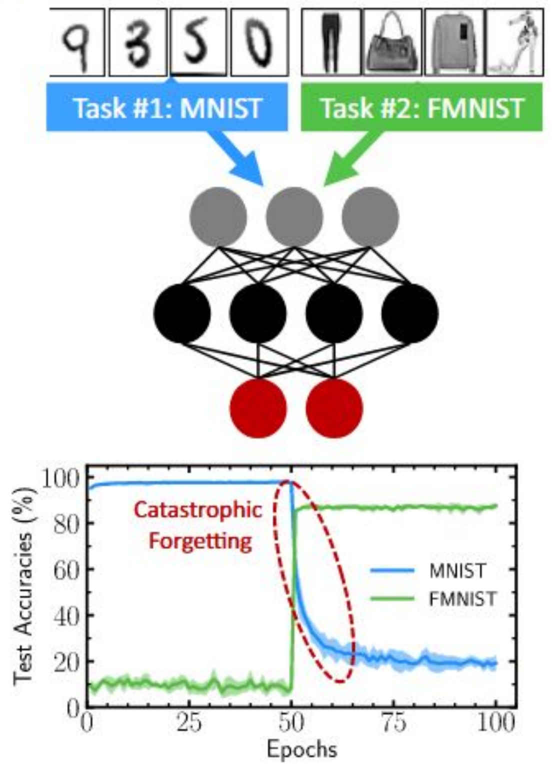
Effective on one task



Every effective in decreasing the training loss when training for solve one type of problem, e.g, train a NN to separate signal events *S* from background events *B*.

However, if one keeps training the NN to separate S from another background events C. The performance of the NN on *S* over C will increase, but that on *S* over B decreases.

Catastrophic forgetting



[Synaptic metaplasticity in binarized neural networks]()

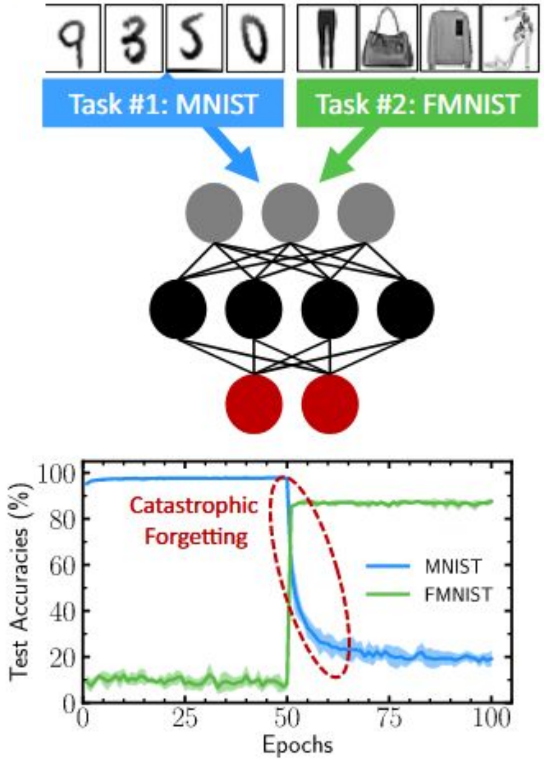
One trains a NN to classify MNIST dataset for the first 50 epochs;

Then keep training the same NN to classify FMNIST dataset

The NN "forgets" its knowledge about MNIST.

How to remedy this?

Catastrophic forgetting



[Synaptic metaplasticity in binarized neural networks]()

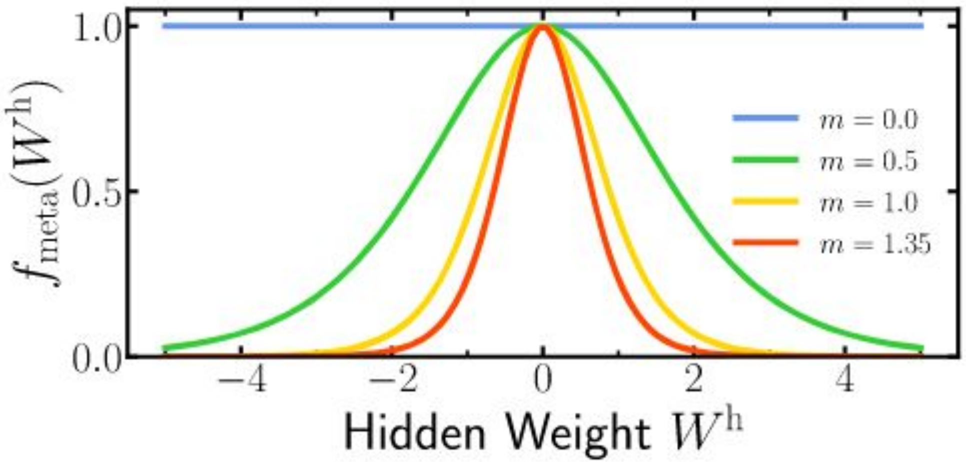
Possible solutions:

- mix the two datasets in the training
  - But when a new dataset comes in, one needs to mix the new dataset with the old one and retrain the model
- “Protect” important message from being updated

Metaplasticity function

The idea is to only update trainable parameters of small values through the meta function, with m being hyperparameter.

$$f_{\text{meta}}(m, w) = 1 - \tanh^2(m \cdot w)$$



Optimization algorithm

Start from the Adam algorithm to get a "normal" step size.

$$U = \hat{m}_t / (\sqrt{\hat{v}_t + \epsilon})$$

$$\boxed{w_{n+1} = w_n - \gamma \cdot U}$$

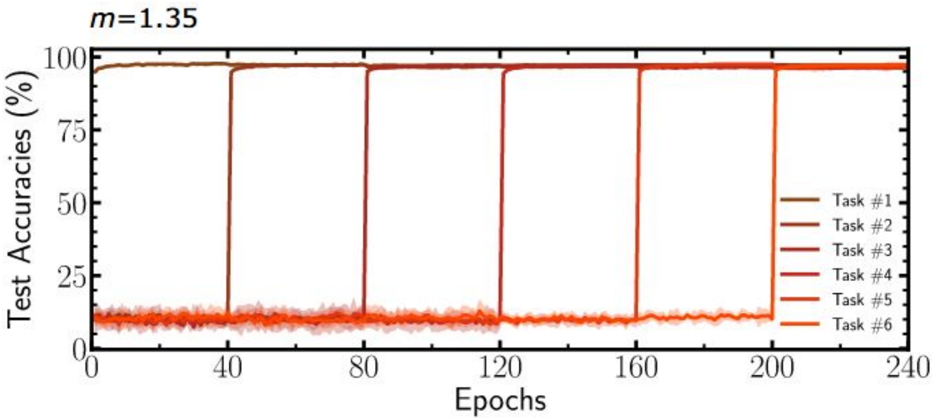
If $U \cdot w_n > 0$ (U prescribes to decrease |w|, use meta update)

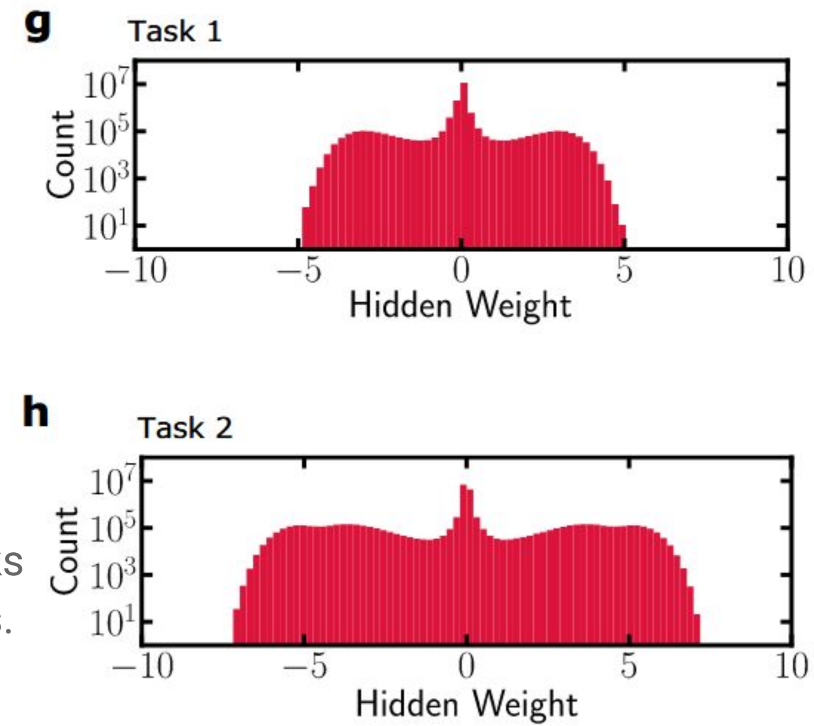
$$w_{n+1} = w_n - \gamma \cdot U \cdot f_{\mathrm{meta}}$$

else:

$$w_{n+1} = w_n - \gamma \cdot U$$

Results from the paper





Binarized neural network (whose weights and activations are constrained to +/- 1 for low computational and memory cost) trained with 6 tasks sequentially and obtained high accuracy for all tasks.

Absolute weight values become larger

Project 1, Let machine remember

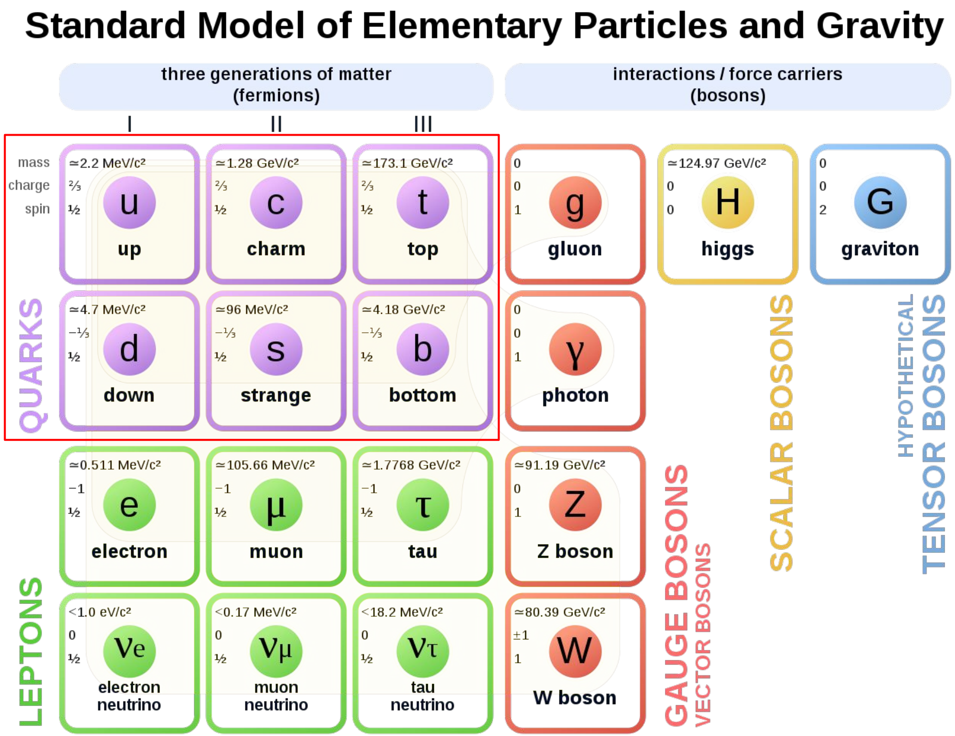
Primary objectives are:

1. to sequentially train a NN to separate signal events from two or three different background events to see if the metaplasticity method work in realistic applications,
2. to compare with conventional way of training NNs, in which one mixes the three backgrounds.

Let Machine Learning Simulate Hadronic Interactions

Elementary particles

BERKELEY LAB



Hadrons are particles that experience the strong force.

Two types of Hadrons: mesons and baryons.

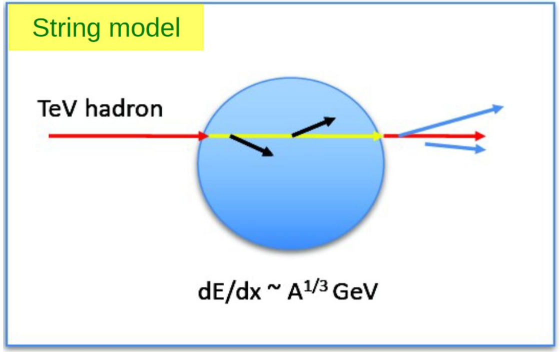
A meson contains one quark and one antiquark, like pions (π±), kaons (K±)

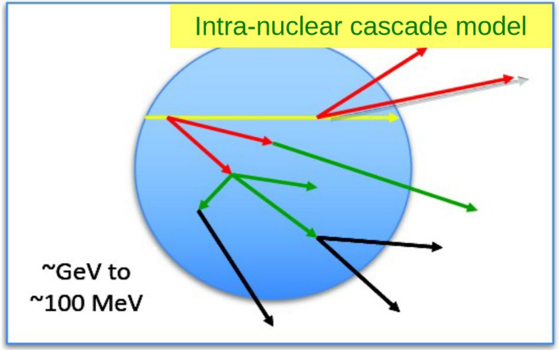
A baryon contains three quarks; proton, neutron…

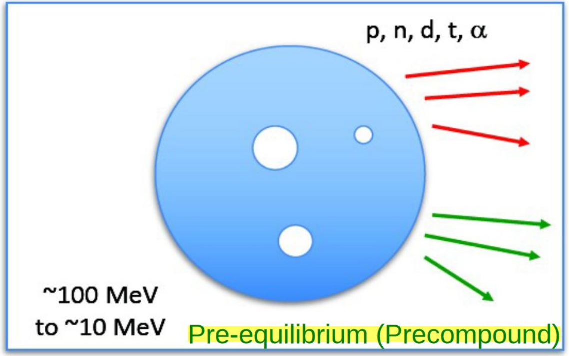
Hadronic interaction

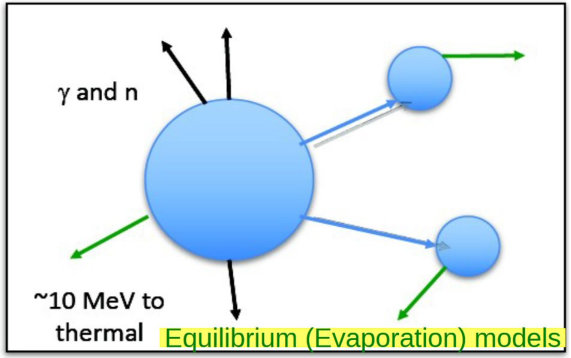
- Hadrons ($\pi\pm$, $K\pm$, K_L, p, n, etc) traverse the detectors (H, C, Ar, Fe, Cu, W, Pb, etc)
- Therefore we need to model hadronic interactions, hadron - nucleus $\rightarrow$ anything, in our detector simulations
- Hadronic models are valid for limited combinations of **particle type, energy, target material**

Hadronic Interactions from TeV to MeV

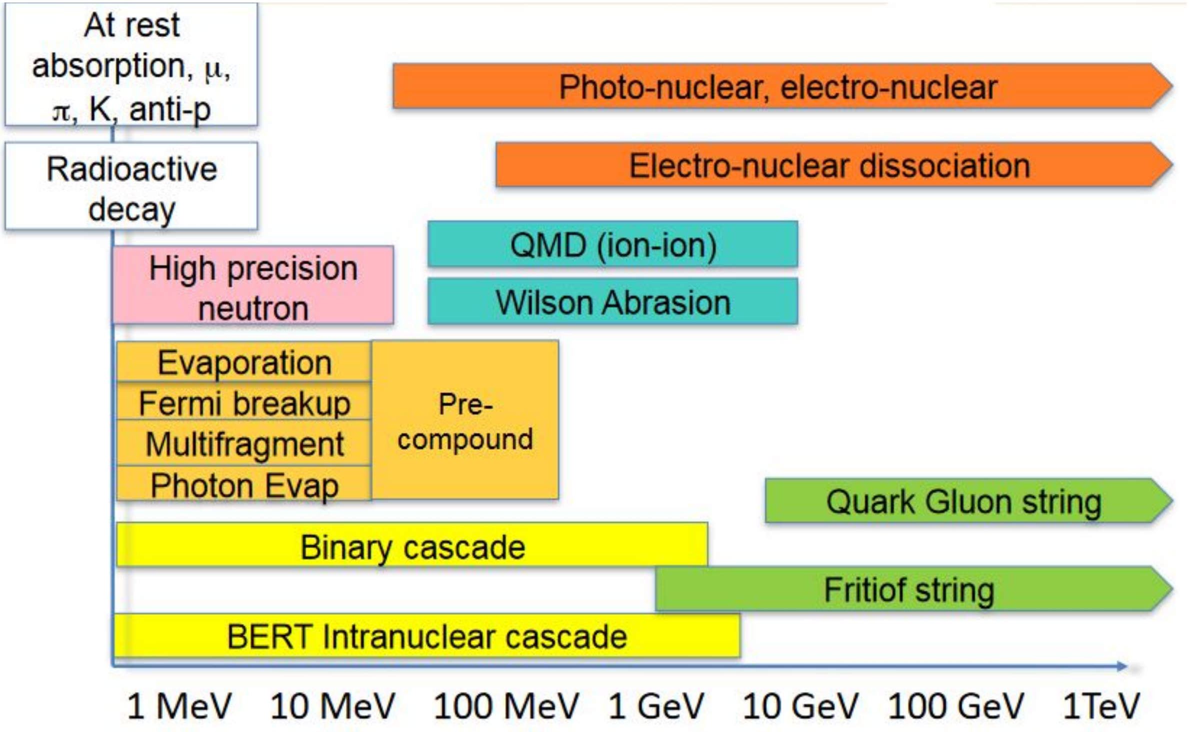








Hadronic models



Energy of incident particle

Each model contains many tunable parameters.

Need to tune those model so as to match experimental data

Why machine learning?

- To reach current levels of performance, many hours were spent in tuning these hadronic models
- Hard to speedup these models with multicore CPUs or GPUs
- Advantages of NN:
    - Good portability, high parallelism, advanced tools for hyperparameter tuning
- Objective for NN:
    - 1) Learn the distribution of the outgoing particle kinematics
    - 2) Produce variable number of outputs
    - 3) Directly trained to data

Experiment setup

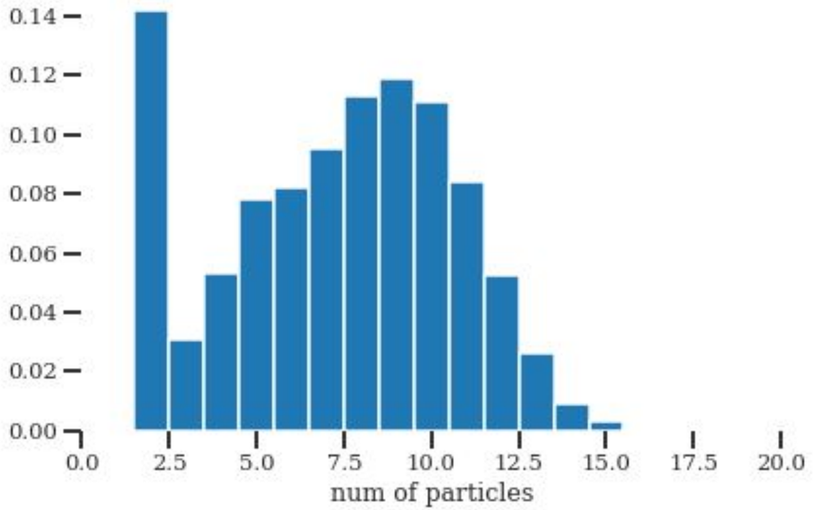
Incoming hadron: pion, with a fixed energy of 25 GeV and direction of [0.6, 0.6, 0.529].

Material: H,

Physics List: FTFP_BERT_ATL

Generate 100,000 events for the study

About 14% chances pion and proton exchange tiny amount of energy



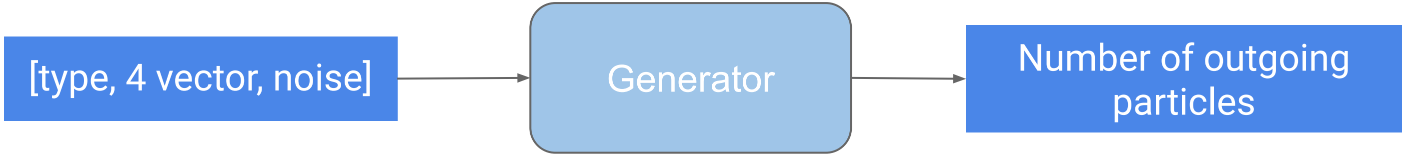
Project 2, Let ML simulate physics

Primary objectives are:

- For a given incoming hadron type and kinematics, simulate the number of outgoing particles distribution
  - One hadron type, different hadron kinematics
  - Different hadron types, different hadron kinematics

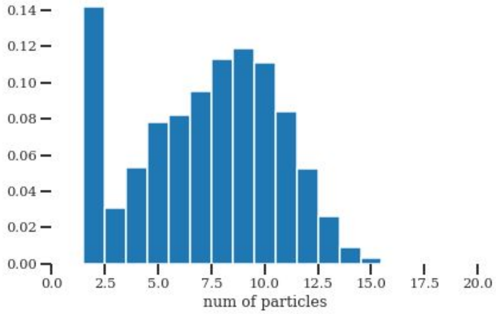
Is it possible to have one ML model that simulates all hadron types across all kinematic ranges?

Generative Adversarial Networks (GAN)



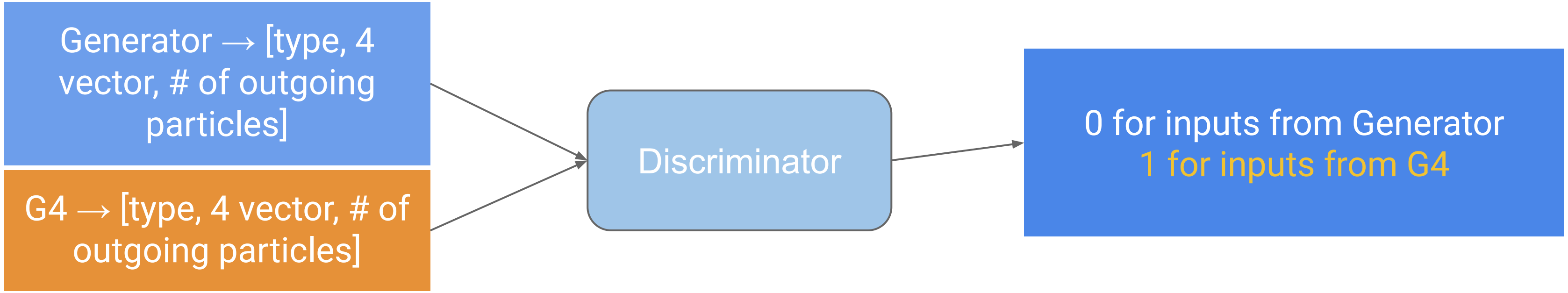
Generator:

- is a Neural Network
- takes inputs about hadrons and random numbers sampled from a distribution (aka noise)
- produces the number of outgoing particles



Once trained, the generator should produce the same distribution as the Geant4 package does.

Discriminator in a GAN

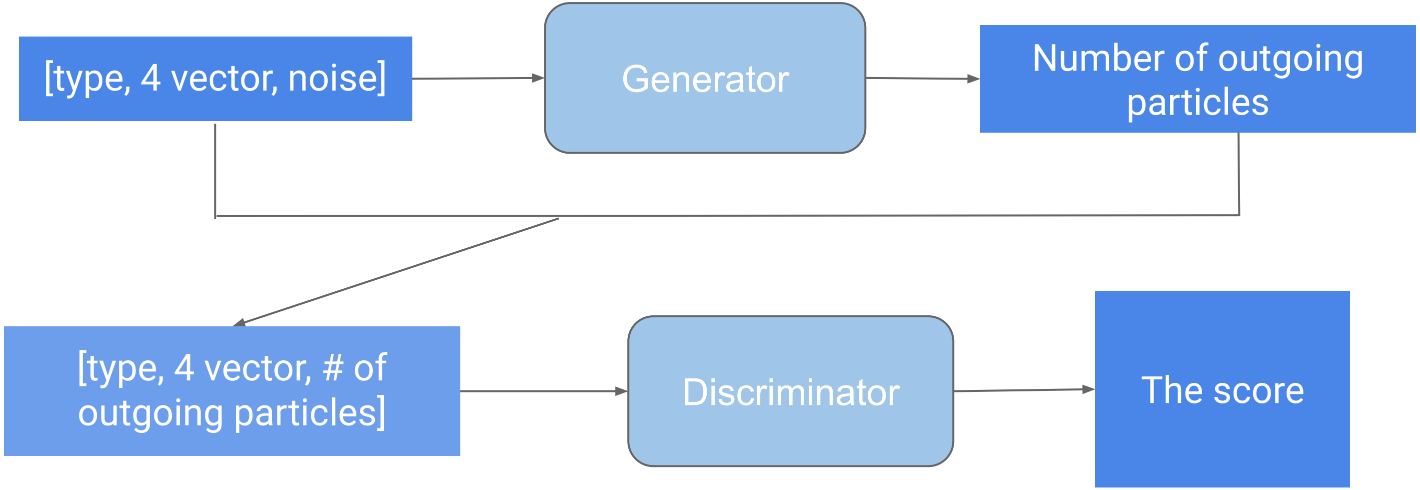


Inputs from Geant4 are signal and those from "Generator" are background

Discriminator:

- Is a Neural Network to perform a binary classification task
- Is to separate signal from background

Train the Generator



- The generator is trained so that “The score” is high.
- In the backpropagation stage, only trainable parameters in the “generator” is updated.

Summary

Let ML remember the past

- Write a customized optimizer to optimize a deep learning model for separating signal events from different background events
- Compare with conventional algorithms

Let ML simulate physics

- Learn GANs.
- Construct and optimize a GAN for the simulation hadronic interactions