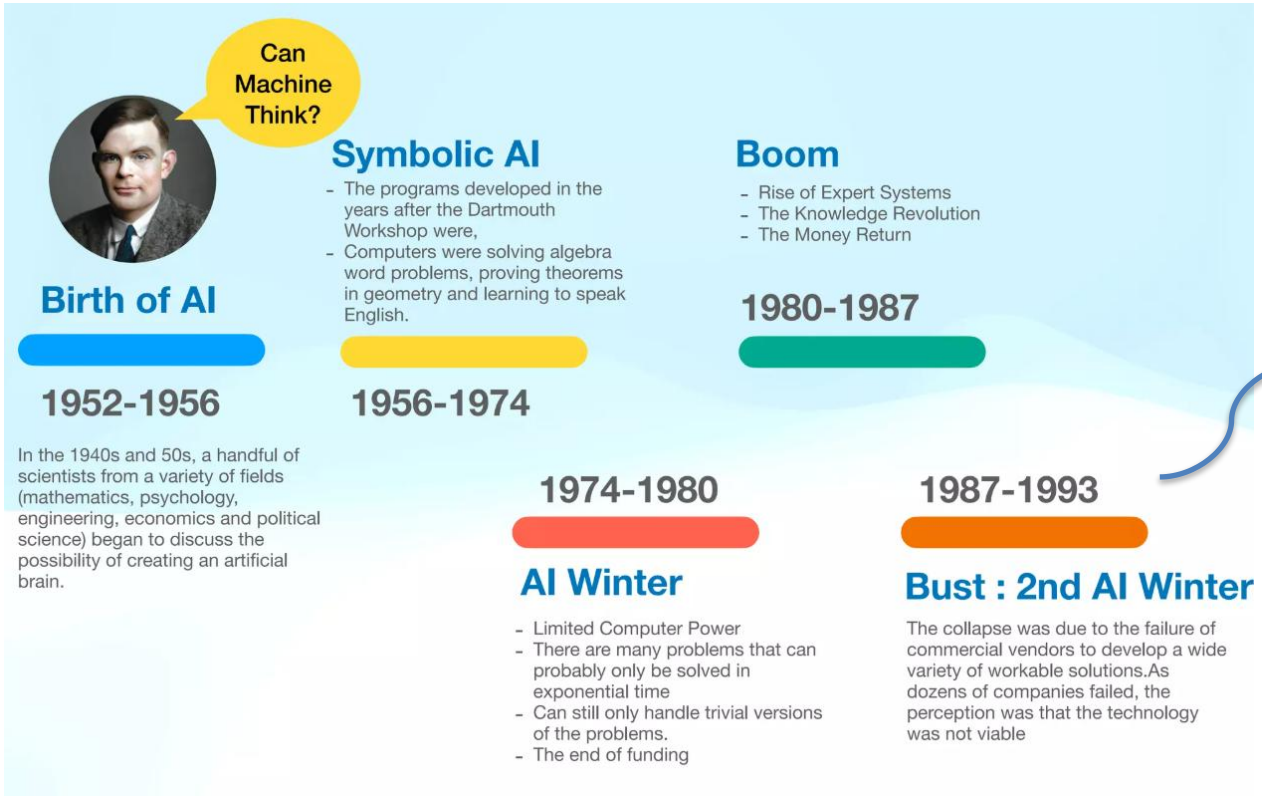
Journal Club

Meisam Ghasemi Bostanabad

School of Particles and Accelerators:
2025-5-5

historical evolution of Artificial Intelligence



And then rise of Neural network!

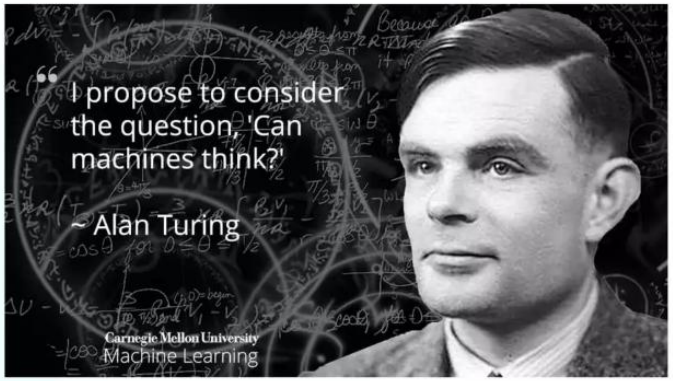
- Artificial Intelligence (AI) is the simulation of human intelligence in machines that are designed to think, learn, and make decisions.
- Machine Learning (ML) is an approach to AI where the algorithms **learn from the data**.

Turing Test

1950 - Alan Turing

Can Machine Think?

1950: Alan Turing publishes "Computing Machinery and Intelligence" which proposes the Turing Test, a method for determining whether a machine can exhibit intelligent behaviour equivalent to, or indistinguishable from, that of a human.



Computing Machinery and Intelligence

A. M. Turing

1950

1 The Imitation Game

I propose to consider the question, "Can machines think?" This should begin with definitions of the meaning of the terms "machine" and "think." The definitions might be framed so as to reflect so far as possible the normal use of the words, but this attitude is dangerous, If the meaning of the words "machine" and "think" are to be found by examining how they are commonly used it is difficult to escape the conclusion that the meaning and the answer to the question, "Can machines think?" is to be sought in a statistical survey such as a Gallup poll. But this is absurd. Instead of attempting such a definition I shall replace the question by another, which is closely related to it and is expressed in relatively unambiguous words.

https://web.iitd.ac.in/~sumeet/Turing50.pdf

So many criticisms of the Turing Test!

GPT-4.5 was judged to be human 73% of the time during five-minute text-based conversations

Basis of Neural Network

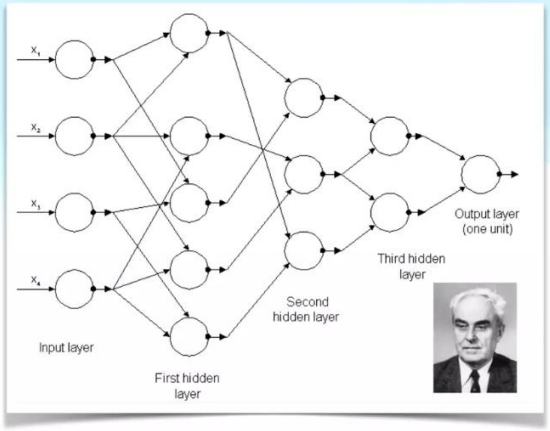
1965 - Anatolii Gershman, Alexey Ivakhnenko, Valentin Lapa

Deep Learning - Multi Layered Perceptron

1965 : **The Group Method of Data Handling** (GMDH) is a data-driven approach to modeling that is based on a multi-layered architecture of interconnected polynomial models.

A multi-layer perceptron (MLP) is a type of artificial neural network (ANN) that is composed of multiple layers of interconnected nodes, or "neurons". MLPs are typically used for supervised learning tasks, such as classification and regression.

Ivakhnenko is often considered as the father of **deep learning.**



Hinton's application of NN

2006 - Geoffrey Hinton

Improvement in Speech and Image Recognition

2006: Geoffrey Hinton and his team develop deep learning algorithms that significantly improve speech recognition and image recognition

Deep Belief Networks, which allows for efficient and effective training of large-scale neural networks for machine learning tasks.

A fast learning algorithm for deep belief nets *

Geoffrey E. Hinton and **Simon Osindero**
Department of Computer Science University of Toronto
10 Kings College Road
Toronto, Canada M5S 3G4
{hinton, osindero}@cs.toronto.edu

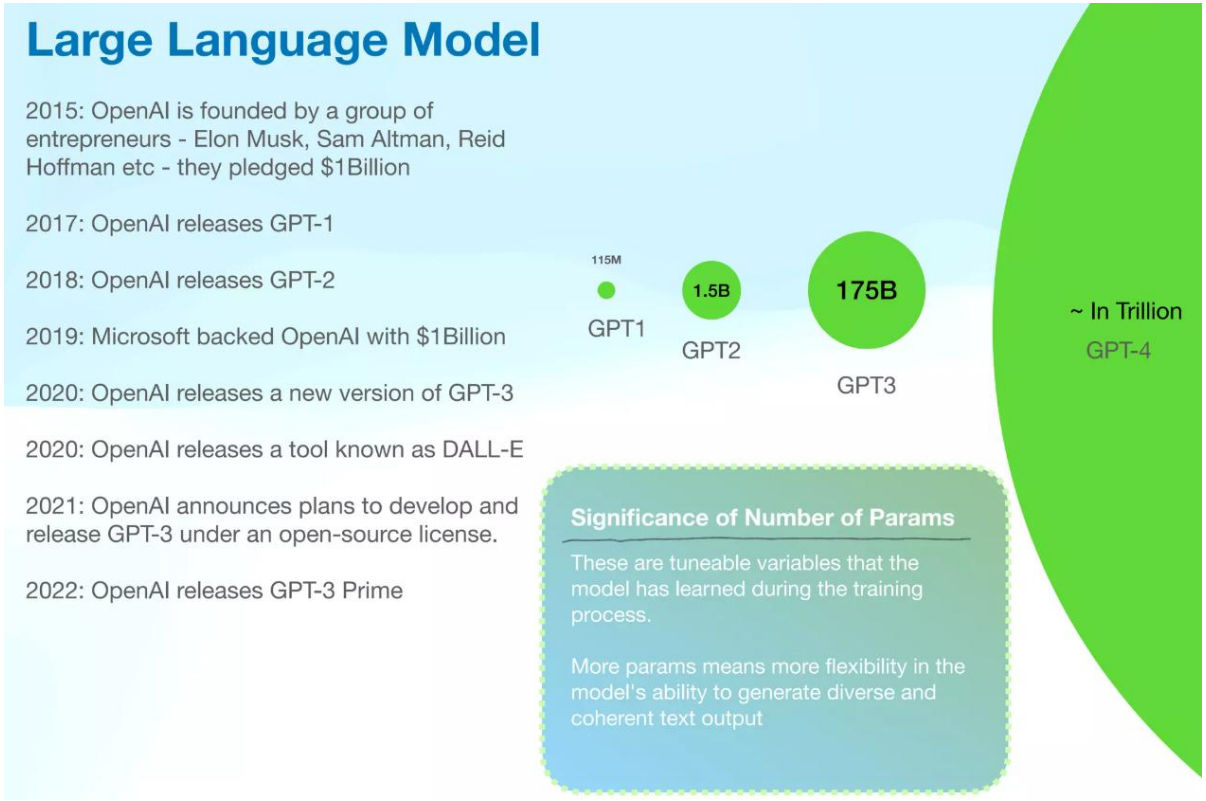
Yee-Whye Teh
Department of Computer Science
National University of Singapore
3 Science Drive 3, Singapore, 117543
tehyw@comp.nus.edu.sg

Abstract

We show how to use "complementary priors" to eliminate the explaining away effects that make inference difficult in densely-connected belief nets that have many hidden layers. Using complementary priors, we derive a fast, greedy algorithm that can learn deep, directed belief networks one layer at a time, provided the top two layers form an undirected associative memory. The fast, greedy algorithm is used to initialize a slower learning procedure that fine-tunes the weights using a contrastive version of the wake-sleep algorithm. After fine-tuning, a network with three hidden layers forms a very good generative model of the joint distribution of handwritten digit images and their labels. This generative model gives better digit classification than the best discriminative learning algorithms. The low-dimensional manifolds on which the digits lie are modelled by long ravines in the free-energy landscape of the top-level associative memory, and it is easy to explore these ravines by using the directed connections to display what the associative memory has in mind. Finally, we show how the model can be trained discriminatively and used both as a classifier and as a generative model.

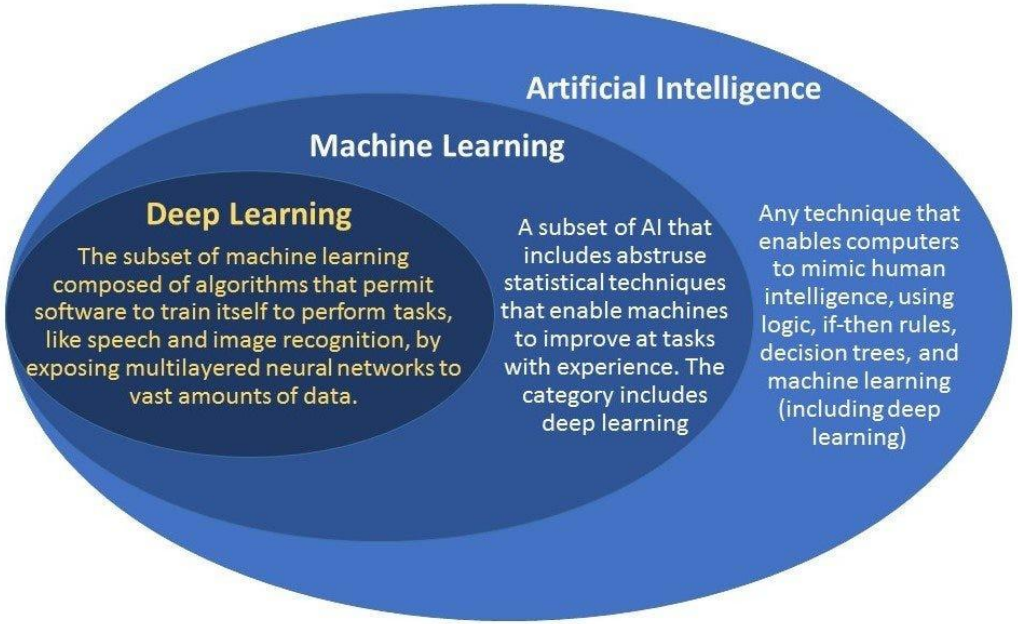
- Hinton's work: Backpropagation, AlexNet and "Attention is All you need" like transformers.

AI Applications



GPT3 trained on NVIDIA V100 GPUs **VS** GPT4 trained on NVIDIA H100 chips

AI, ML and DL relation

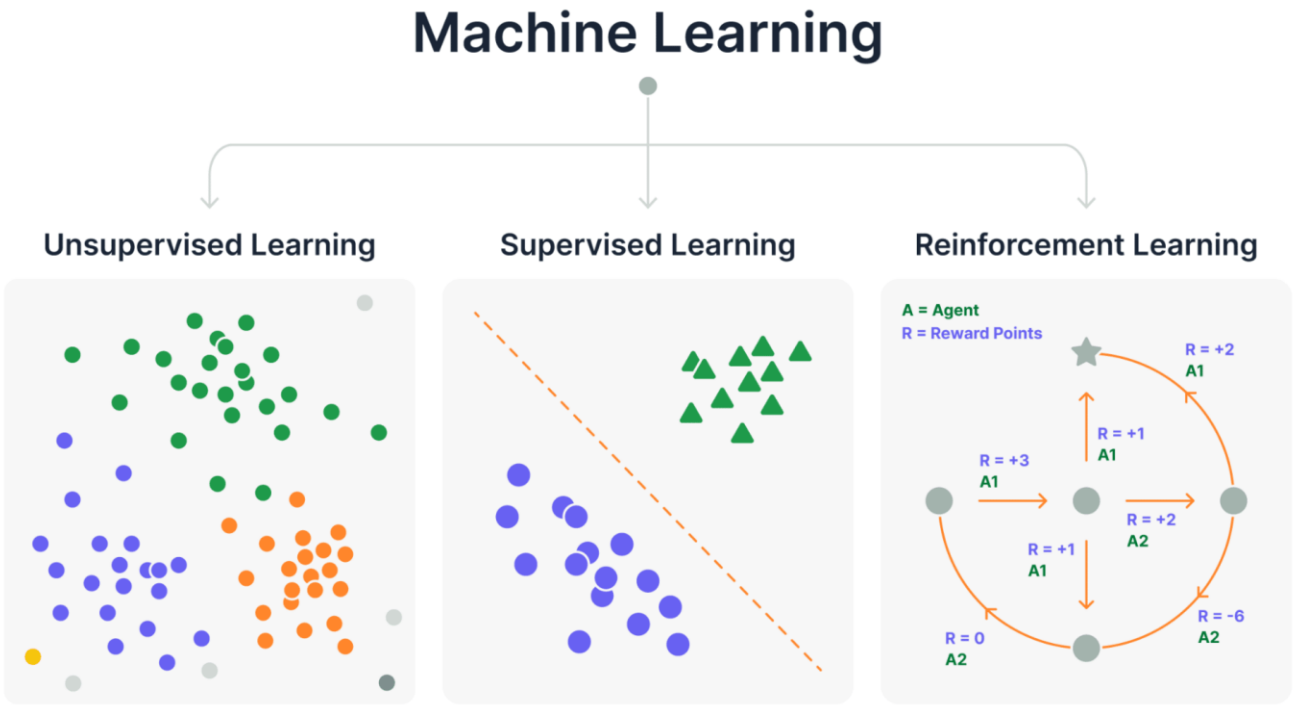


AI: Broad goal of mimicking human intelligence.

ML: Achieving AI through data-driven learning.

DL: Achieving ML using neural networks with multiple layers.

ML Types



Unlabeled data:
- LHC data
- Clustering

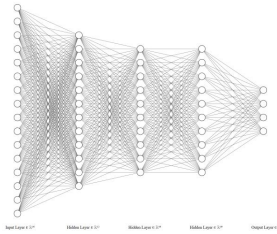
Data with correct label:
- MC simulation of signal and bkg
- Classification and Regression

Agent interact with env:
- No static dataset
- Used in robotics

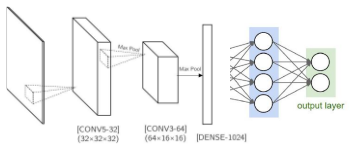
What part of ML We going to Cover

- Introduce key machine learning architectures: Deep Neural Networks (DNNs), Convolutional Neural Networks (CNNs), Quantum Machine Learning (QML), and Generative Adversarial Networks (GANs). The training process is covered for each model.
- Highlight the core structure and intuition behind each model.
- Demonstrate how these models are applied in particle physics, with real-world researches from classification of signal from background and event simulation.
- Graph Neural Networks (GNN), Recurrent Neural Networks (RNN), Transformers, Unsupervised learning and Reinforcement learning are not covered in this talk!

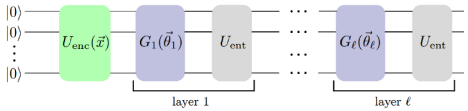
DNN



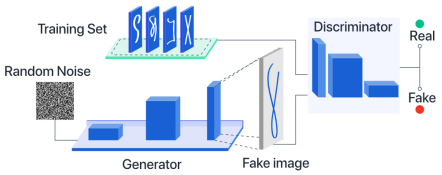
CNN



QML



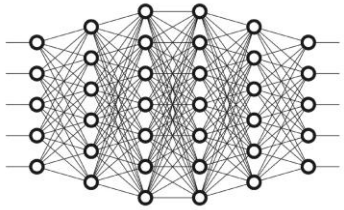
GAN

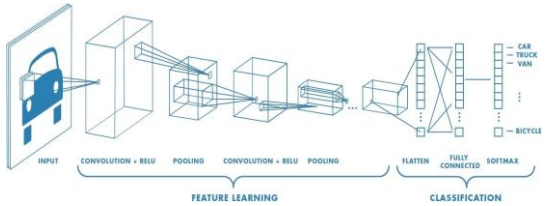


Deep Neural Network
(Multi layer perceptron)

Why Deep Neural Network?

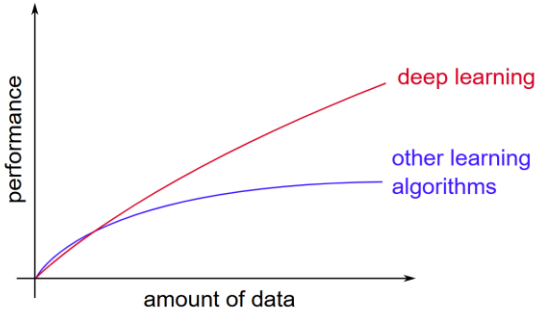
1. Better algorithm and understanding





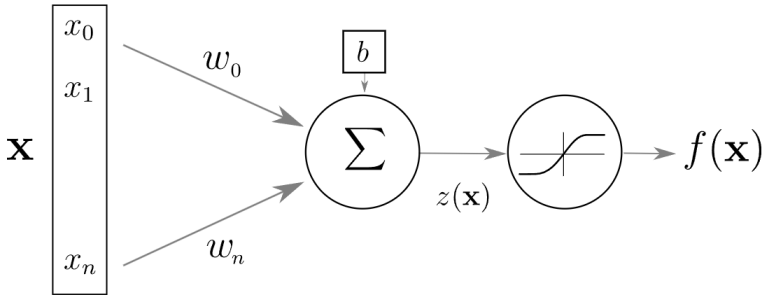
2. Computing power (GPU and TPU)

3. Data with labels



4. Open source tools and models

Simplest DNN



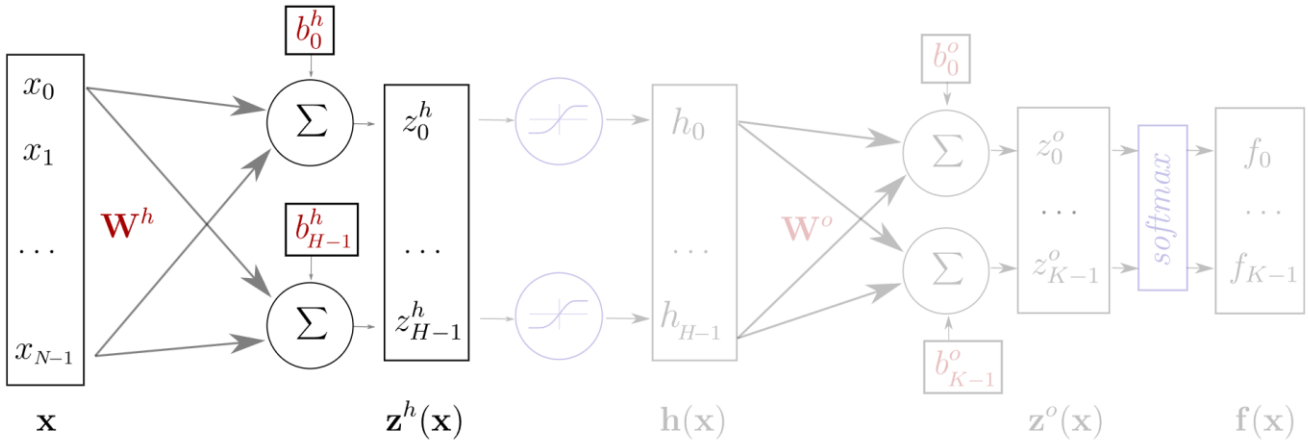
$$z(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b$$

$$f(\mathbf{x}) = g(\mathbf{w}^T\mathbf{x} + b)$$

- $\mathbf{x}, f(\mathbf{x})$ input and output
- $z(\mathbf{x})$ pre-activation
- $\mathbf{w}, b$ weights and bias → **Learnable params**
- g activation function → **Adding non linearity**

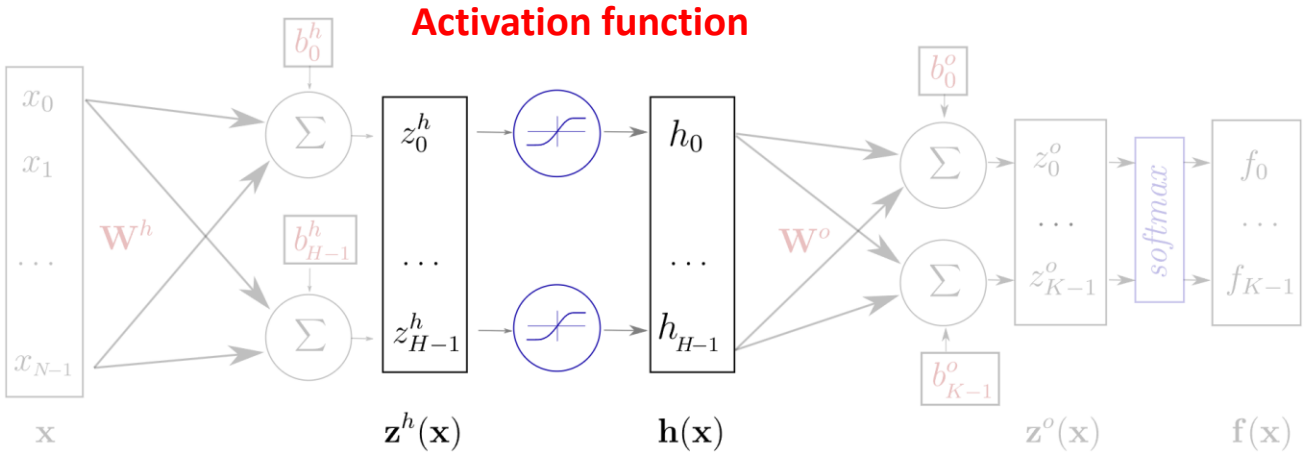
DNN Forward Propagation

First hidden layer



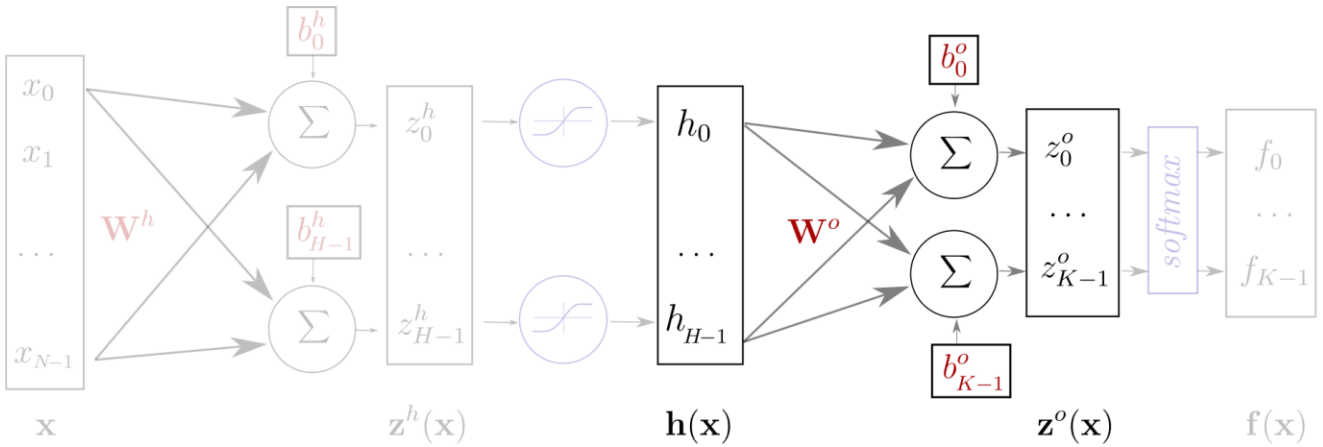
- $\mathbf{z}^h(\mathbf{x}) = \mathbf{W}^h\mathbf{x} + \mathbf{b}^h$
- $\mathbf{h}(\mathbf{x}) = g(\mathbf{z}^h(\mathbf{x})) = g(\mathbf{W}^h\mathbf{x} + \mathbf{b}^h)$
- $\mathbf{z}^o(\mathbf{x}) = \mathbf{W}^o\mathbf{h}(\mathbf{x}) + \mathbf{b}^o$
- $\mathbf{f}(\mathbf{x}) = softmax(\mathbf{z}^o) = softmax(\mathbf{W}^o\mathbf{h}(\mathbf{x}) + \mathbf{b}^o)$

DNN Forward Propagation



- $\mathbf{z}^h(\mathbf{x}) = \mathbf{W}^h\mathbf{x} + \mathbf{b}^h$
- $\mathbf{h}(\mathbf{x}) = g(\mathbf{z}^h(\mathbf{x})) = g(\mathbf{W}^h\mathbf{x} + \mathbf{b}^h)$
- $\mathbf{z}^o(\mathbf{x}) = \mathbf{W}^o\mathbf{h}(\mathbf{x}) + \mathbf{b}^o$
- $\mathbf{f}(\mathbf{x}) = softmax(\mathbf{z}^o) = softmax(\mathbf{W}^o\mathbf{h}(\mathbf{x}) + \mathbf{b}^o)$

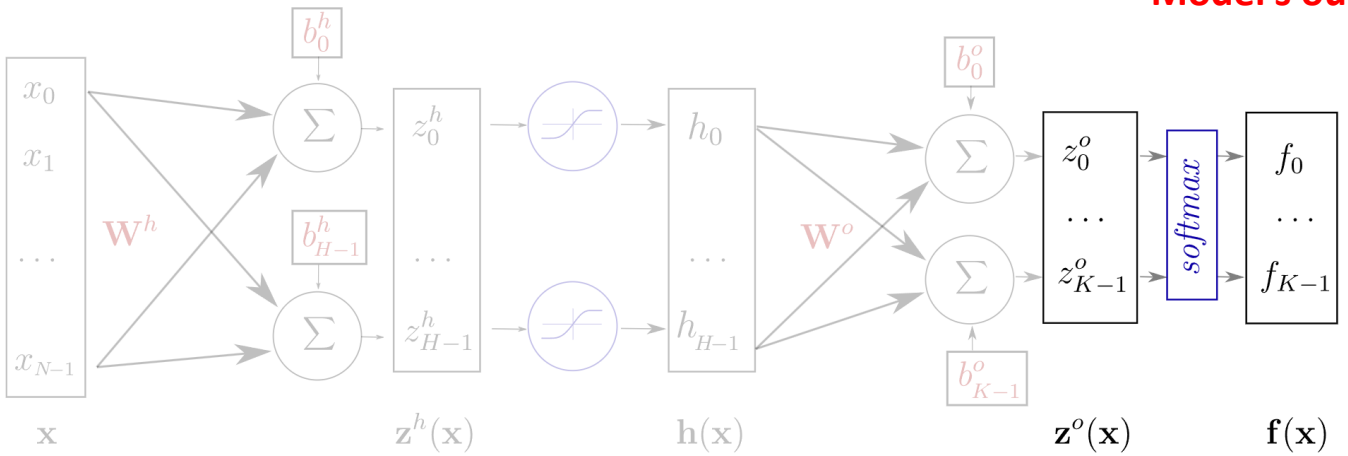
DNN Forward Propagation



- $\mathbf{z}^h(\mathbf{x}) = \mathbf{W}^h\mathbf{x} + \mathbf{b}^h$
- $\mathbf{h}(\mathbf{x}) = g(\mathbf{z}^h(\mathbf{x})) = g(\mathbf{W}^h\mathbf{x} + \mathbf{b}^h)$
- $\mathbf{z}^o(\mathbf{x}) = \mathbf{W}^o\mathbf{h}(\mathbf{x}) + \mathbf{b}^o$
- $\mathbf{f}(\mathbf{x}) = softmax(\mathbf{z}^o) = softmax(\mathbf{W}^o\mathbf{h}(\mathbf{x}) + \mathbf{b}^o)$

DNN Forward Propagation

Model's output

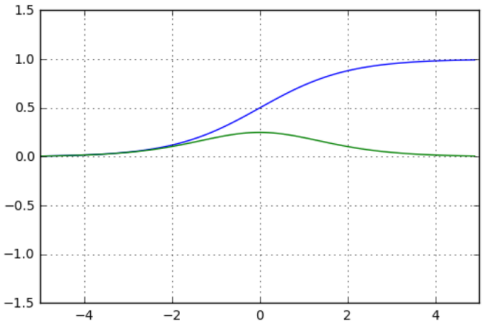


- $\mathbf{z}^h(\mathbf{x}) = \mathbf{W}^h\mathbf{x} + \mathbf{b}^h$
- $\mathbf{h}(\mathbf{x}) = g(\mathbf{z}^h(\mathbf{x})) = g(\mathbf{W}^h\mathbf{x} + \mathbf{b}^h)$
- $\mathbf{z}^o(\mathbf{x}) = \mathbf{W}^o\mathbf{h}(\mathbf{x}) + \mathbf{b}^o$
- $\mathbf{f}(\mathbf{x}) = softmax(\mathbf{z}^o) = softmax(\mathbf{W}^o\mathbf{h}(\mathbf{x}) + \mathbf{b}^o)$

Normalized probabilities

Activation Functions

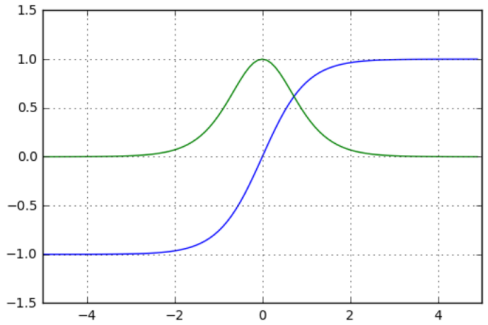
Binary classification



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

$$\text{sigm}'(x) = \text{sigm}(x)(1-\text{sigm}(x))$$

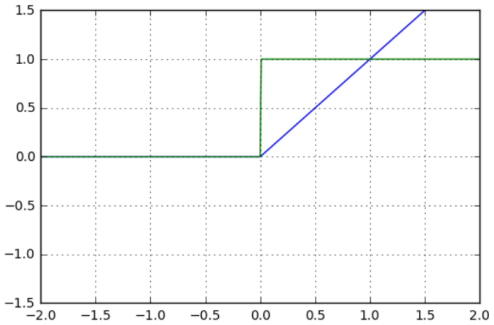
Zero-centered



$$\tanh(x) = \frac{e^{2x}-1}{e^{2x}+1}$$

$$\tanh'(x) = 1-\tanh(x)^2$$

Fast to compute



$$\text{relu}(x) = \max(0,x)$$

$$\text{relu}'(x) = 1_{x>0}$$

$$softmax(\mathbf{x}) = \frac{1}{\sum_{i=1}^{n} e^{x_i}} \cdot \begin{bmatrix} e^{x_1} \\ e^{x_2} \\ \vdots \\ e^{x_n} \end{bmatrix}$$

Multiclass classification

Loss Function as NLL

- Adjust the model's parameters θ (W;b) to minimize a loss (negative log likelihood).

Predicted prob

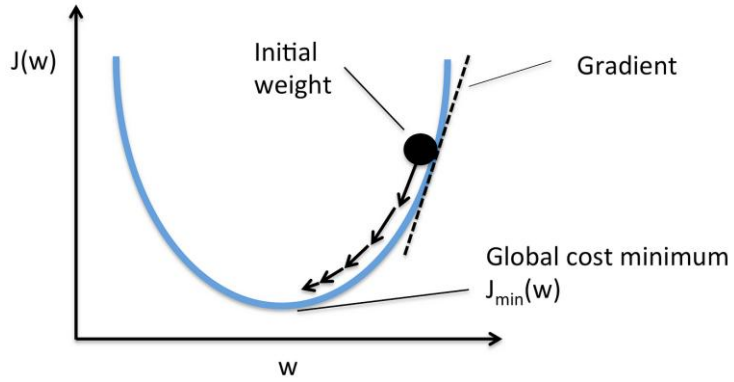
$$l(\mathbf{f}(\mathbf{x}^s;\theta), y^s) = nll(\mathbf{x}^s, y^s;\theta) = -\log \mathbf{f}(\mathbf{x}^s;\theta)_{y^s}$$

example $y^s = 3$

$$l(\mathbf{f}(\mathbf{x}^s;\theta), y^s) = l\left(\begin{bmatrix} f_0 \\ \cdots \\ f_3 \\ \cdots \\ f_{K-1} \end{bmatrix}, \begin{bmatrix} 0 \\ \cdots \\ 1 \\ \cdots \\ 0 \end{bmatrix}\right) = -\log\ f_3$$

f_3 close to 1 would make loss close to 0.

DNN Backward Propagation (SGD)



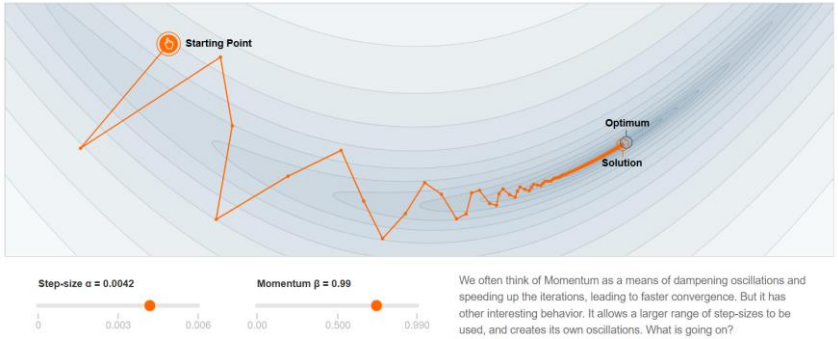
Initialize θ randomly

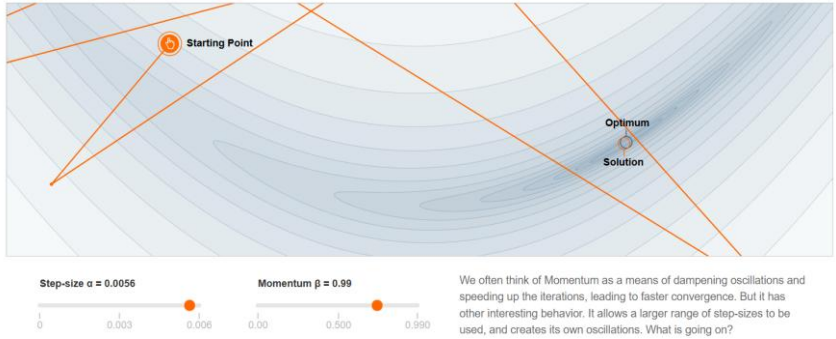
For E epochs perform:

- Randomly select a small batch of samples $(B \subset S)$
  - Compute gradients: $\Delta = \nabla_\theta L_B(\theta)$
  - Update parameters: $\theta \leftarrow \theta - \eta\Delta$
  - $\eta > 0$ is called the learning rate
- Repeat until the epoch is completed (all of S is covered)

Hyper Parameters

1. Number of Hidden Layers:
   - More layers → model can learn complex patterns, but harder to train.
   - Too deep → risk of overfitting or vanishing gradients.
2. Number of Neurons per Layer:
   - More neurons → more capacity, but also more computation.
   - Too few → underfitting, too many → overfitting.
3. Learning Rate:
   - Controls how fast the model updates Too small → slow convergence
   - Too large → unstable training, may overshoot minima.
4. Number of Epochs: How many times the model sees the full training set.

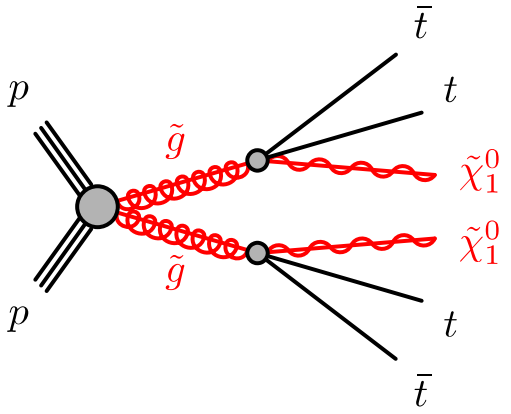


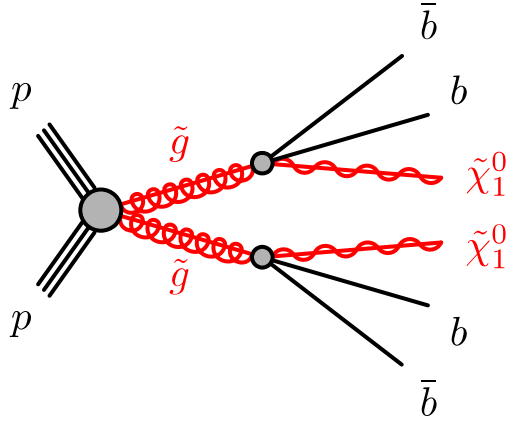


SUSY Simplified Models

We target gluino pair production (from a strong SUSY interaction) with off-shell top and bottom squarks in the decay products

Gtt





Gbb

0 lepton (0L)

1 lepton (1L)

channels

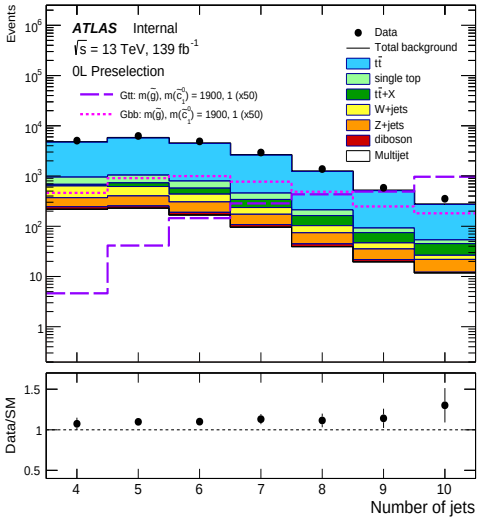
0 lepton (0L)

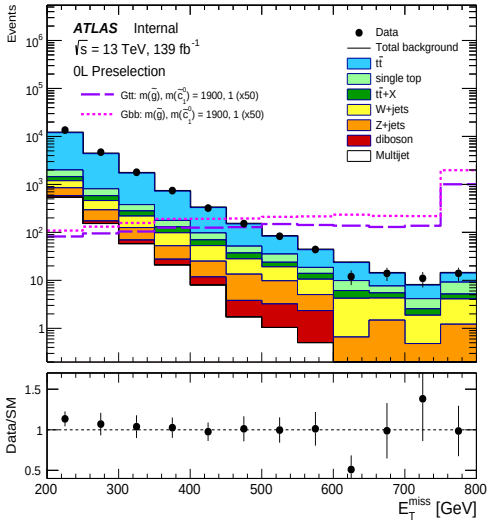
~~1 lepton (1L)~~

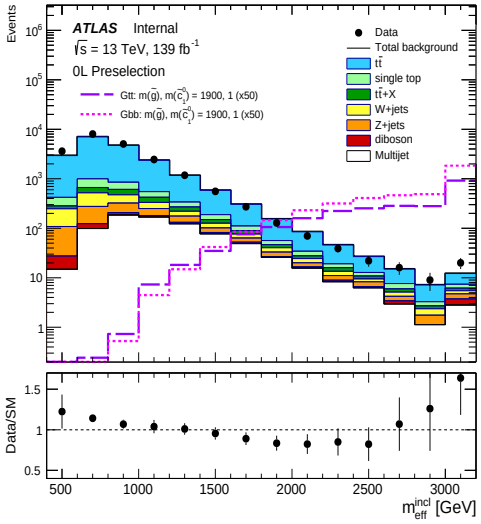
[Paper link]

- Potentially high cross-section for gluino pair production
- Target final states with large amount of E_T^{miss} and $N_{jet} \geq 4, N_{bjet} \geq 3$.
- Main background is **top pair production $t\bar{t}$**

Preselection Distributions







- Inclusive effective mass, to select highly energetic events:

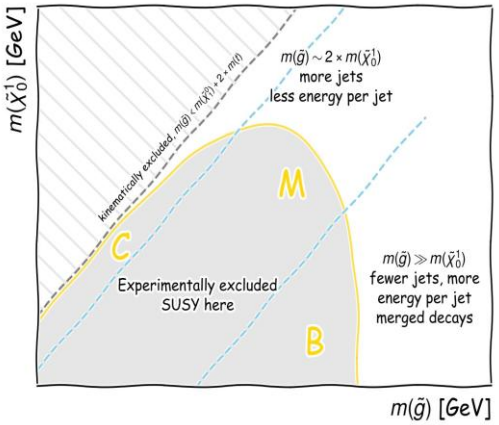
$$m_{eff}^{incl} = \Sigma_{i \le n}\, p_T^{j_i} + \Sigma_{j \le m}\, p_T^{lep_j} + E_T^{miss}$$

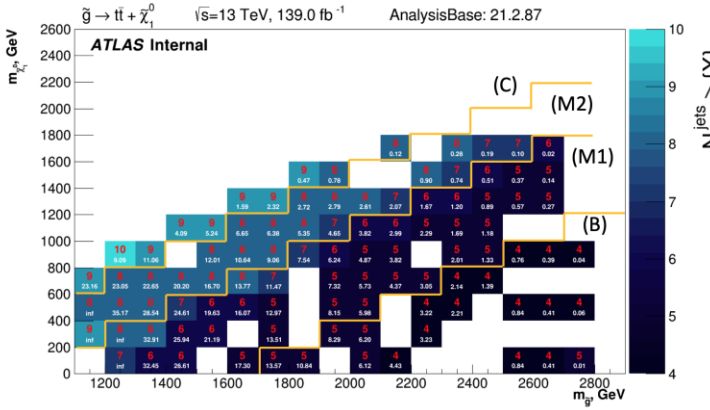
- Transverse mass to remove semileptonic $t\bar{t}$ and W+jets events:

$$m_T = \sqrt{2 p_T^l E_T^{miss} (1 - \cos\Delta\phi(\vec{p}_T^{\,miss}, \vec{p}_T^{\,l}))}$$

Cut-and-count

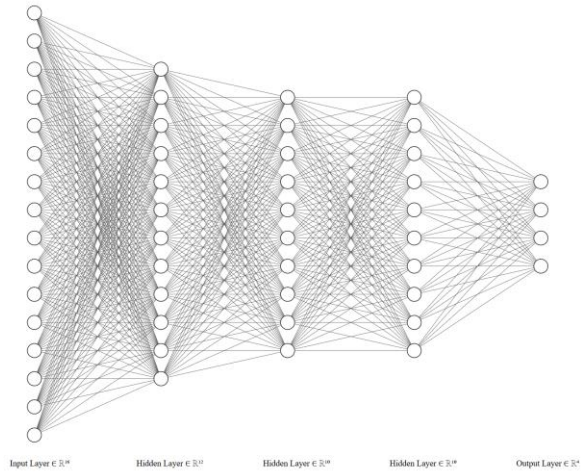
- Define **signal regions** (SRs) targeting different regions of the signal grid that is parametrized by the mass splitting between the gluino and lightest SUSY particle (LSP):
  - optimized to maximize the significance of a discovery
- Non-overlapping **control regions** (CRs):
  - low signal contamination
  - high ttbar purity
  - used to derive scale factors by fitting to data
- Non-overlapping **validation regions** (VRs):
  - validate CRs normalization factors
  - check the extrapolations between signal and control region
- If there are no large excesses or deficits in the VRs, then open the box (unblind)!



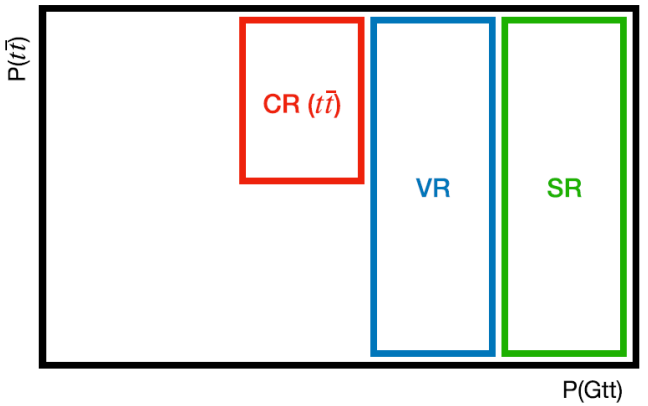


DNN Architecture and Signal Region

- DNN to discriminate Gtt or Gbb events from SM:
  - Inputs: four-momenta of the 10 leading jets, 4 leading leptons, 4 leading fat jets and $\overrightarrow{p_T^{miss}}$
  - Three fully hidden layers with 128 neurons
  - ReLU activation function for hidden layers
  - P(Gtt), P(Gbb), P($\bar{t}t$), P(Z+jets), other P(bkg)



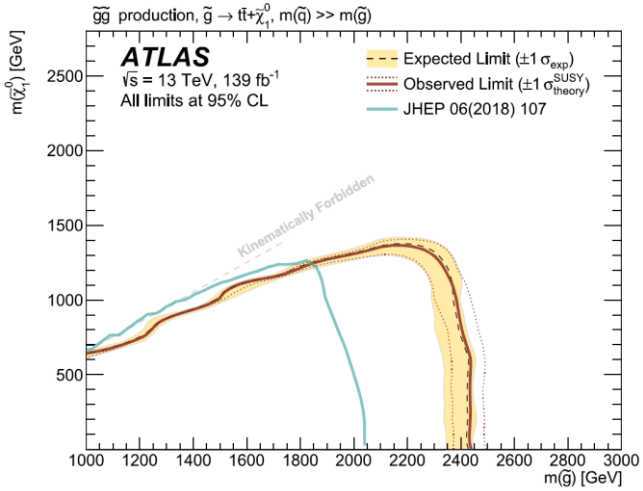
- Same policy to define kinematic regions:
  - CR: defined at high $P(\bar{t}t)$ and low $P(Gtt)$. Enriched in $\bar{t}t$ background
  - VR: intermediate values of $P(\bar{t}t)$ and $P(Gtt)$
  - SR: defined at high $P(Gtt)$ and low $P(\bar{t}t)$. DNN threshold with max significance is chosen



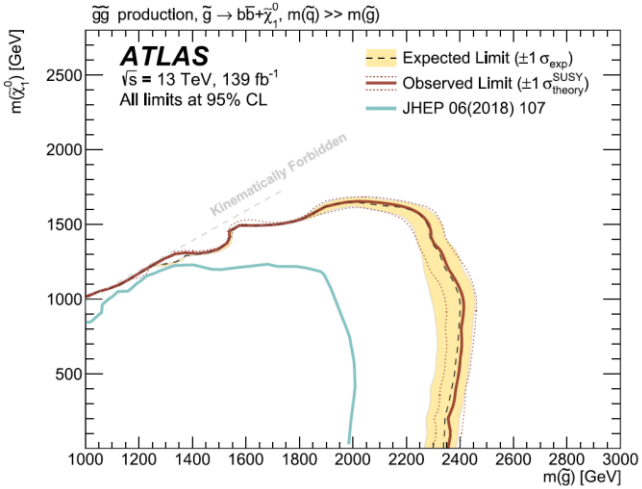
Signal Exclusion Limits

- For each signal mass point, the trained parameterized NN is applied to estimate the expected signal and background yields in optimized SRs.
- A statistical test - CL*s* method - compares observed data with the background-only and signal+background hypotheses.
- Mass points are excluded at 95% confidence level if the CLs is less than 0.05.

Gtt exclusion limits



Gbb exclusion limits



Convolutional Neural Network (CNN)

Application of CNNs (input images)

Image classification

[Krizhevsky 2012]

Handwritten recognition

[Ciresan et al. 2013]

[Faster R-CNN - Ren 2015]

[NVIDIA dev blog]

Object detection (autonomous driving)

Why not DNN for images?

Motivations

Standard Dense Layer for an image input:

```
x = Input((640, 480, 3), dtype='float32')
# shape of x is: (None, 640, 480, 3)
x = Flatten()(x)
# shape of x is: (None, 640 x 480 x 3)
z = Dense(1000)(x)
```

Oops so many parameters

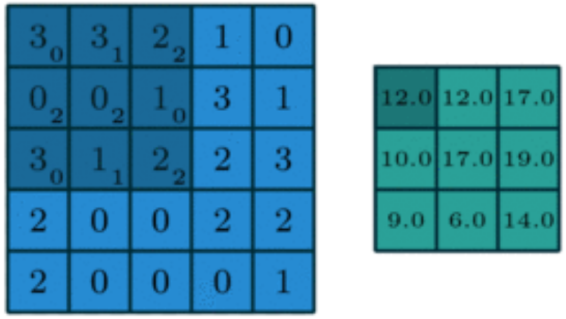
How many parameters in the Dense layer?

$$640 \times 480 \times 3 \times 1000 + 1000 = 922M!$$

Spatial organization of the input is destroyed by `Flatten`

We never use Dense layers directly on large images. Most standard solution is **convolution** layers

CNN Kernel and Convolution

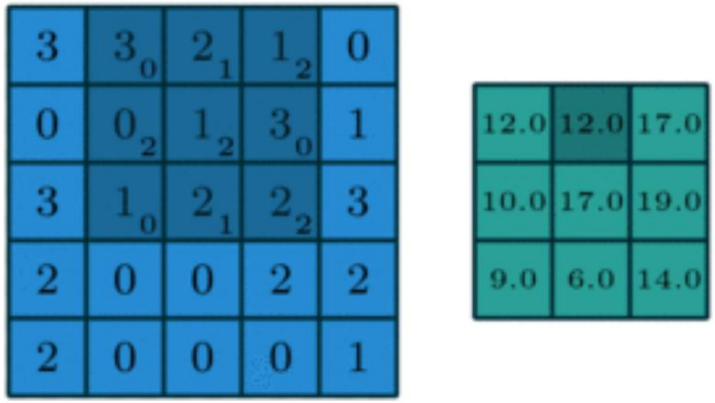


- x is a 3×3 chunk (dark area) of the image *(blue array)*
- Each output neuron is parametrized with the 3×3 weight matrix $\mathbf{w}$ *(small numbers)*

The activation obtained by sliding the 3×3 window and computing:

$$z(x) = relu(\mathbf{w}^T x + b)$$

What about ML not included



2D-convolutions (actually 2D cross-correlation):

$$(f \star g)(x,y) = \sum_n \sum_m f(n,m).\, g(x+n, y+m)$$

Kernel Input

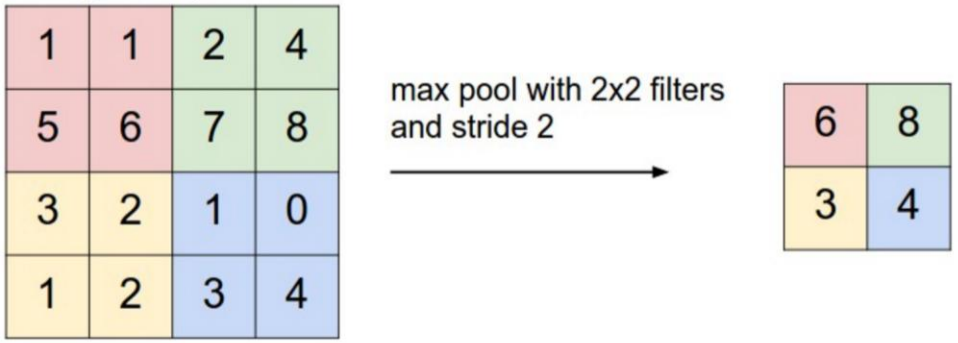
f is a convolution **kernel** or **filter** applied to the 2-d map g (our image)

CNN and Spatial relation in an image

- In images (and other grid-like data), nearby pixels are correlated.
- Patterns like edges, textures, and shapes are built from local groupings of pixels.
- Convolutional layers apply small local filters (e.g., 3×3) across the image.
- This lets the network learn local patterns first, then combine them into more complex structures in deeper layers.
- Fully connected layers treat all inputs as independent, losing spatial info.

Pooling Layers

- Spatial dimension reduction
- Local invariance
- No parameters: max or average of 2x2 units



Classical CNN Architecture

Classic ConvNet Architecture

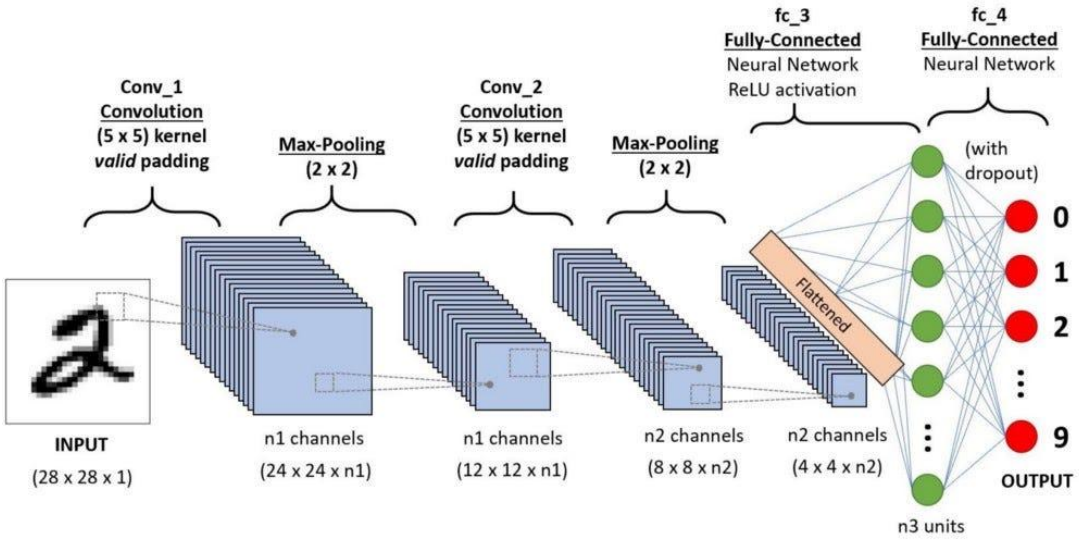
Input

Conv blocks

- Convolution + activation (relu)
- Convolution + activation (relu)
- ...
- Maxpooling 2x2

Output

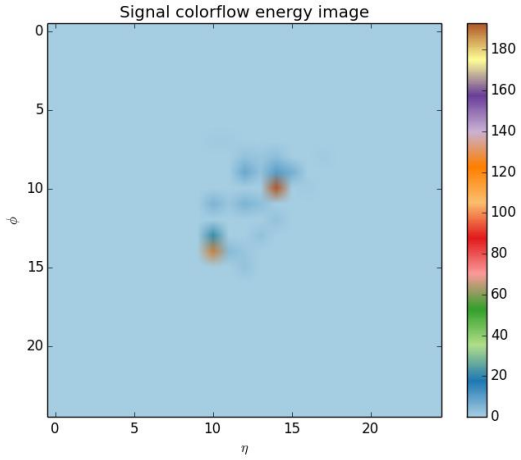
- Fully connected layers
- Softmax



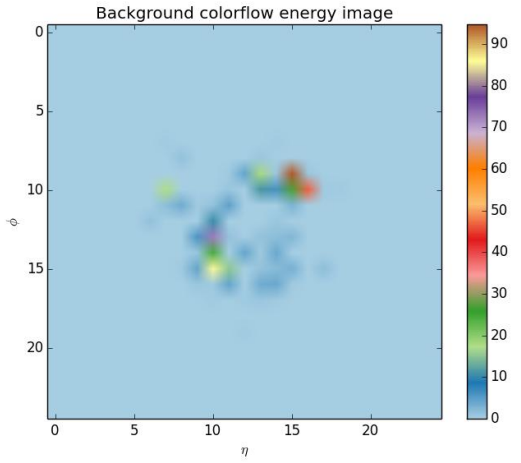
Higgs Identification using CNN

- Goal: Identify **Higgs boson (signal)** decays using jet substructure from **QCD background**.
- Focus on Higgs → $b\bar{b}$, the dominant decay mode.
- Signal forms a single fat jet with 2-prong substructure. Background typically 1-prong, diffuse, asymmetric energy deposition.
- Simulation provided by ATLAS/SLAC collaborators.
- Each event is converted to a 25×25 **energy image** in η–φ space.
- Data preparation: images are centered, zero-padded to 32×32, and normalized.

Higgs

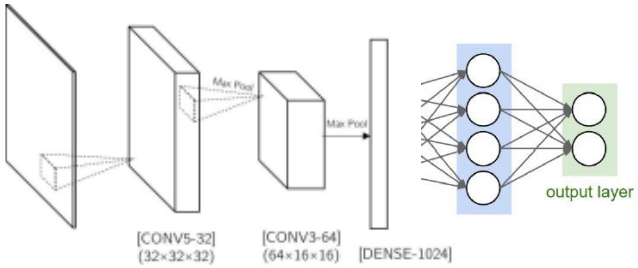


QCD

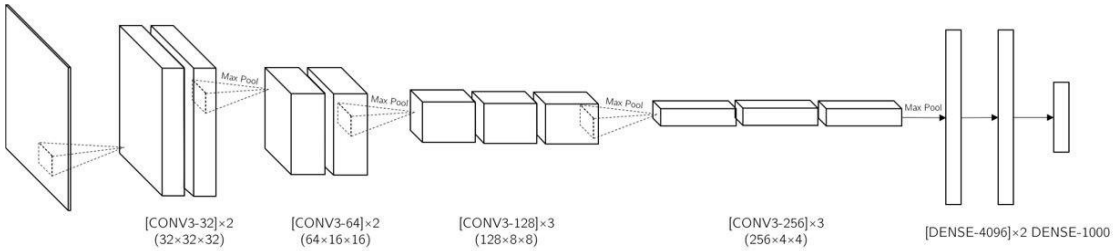


All CNN Models

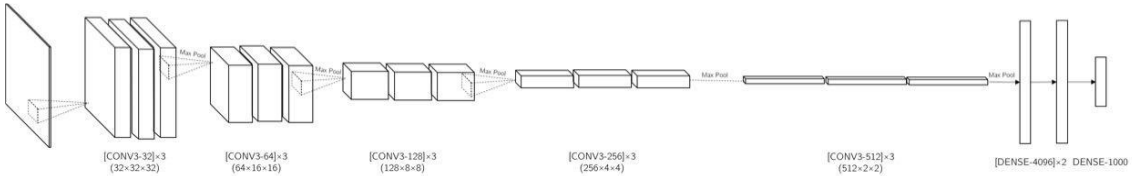
Baseline model



LaNet1 model



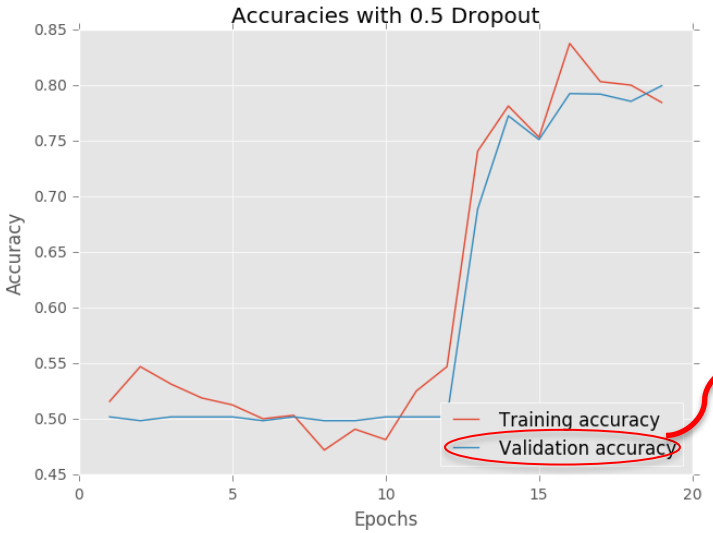
LaNet2 model



LaNet3 same as the base model with wider CNN layers (more channels)

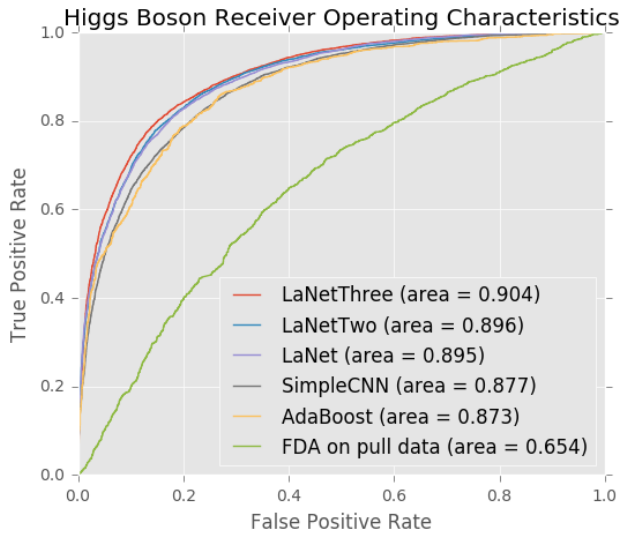
CNN Results (ROC and AUC)

Learning plot



Good to have it

ROC plot

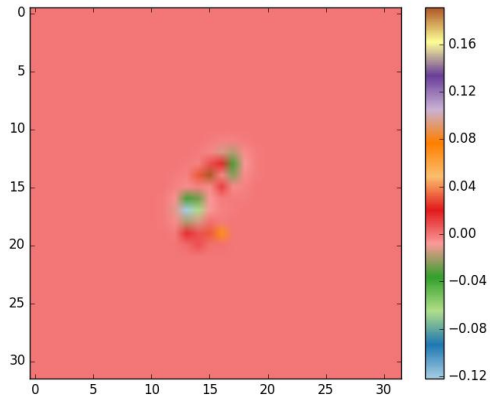


Model	AUC	Accuracy
FDA	0.654	-
AdaBoost	0.873	0.798
SimpleModel	0.877	0.774
LaNet	0.895	0.812
LaNetTwo	0.896	**0.825**
LaNetThree	**0.904**	0.820

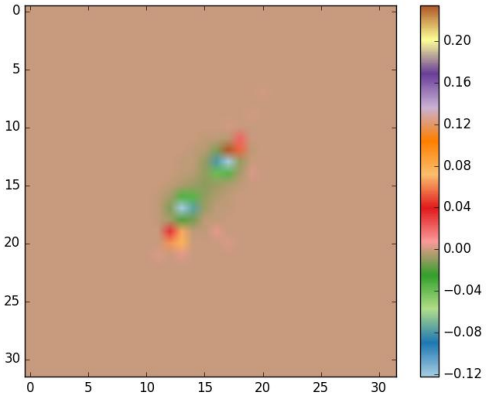
A high AUC was achieved despite the **small image sizes** and without relying on **transfer learning** for jet classification.

Misclassified Images and Suggestions

Truth label: Background
Prediction: Higgs



Truth label: Higgs
Prediction: Background



- Data augmentation might help:
  - Small random rotations
  - Small translations
  - Random flipping
- Combine model's outputs (majority vote) to improve generalization.

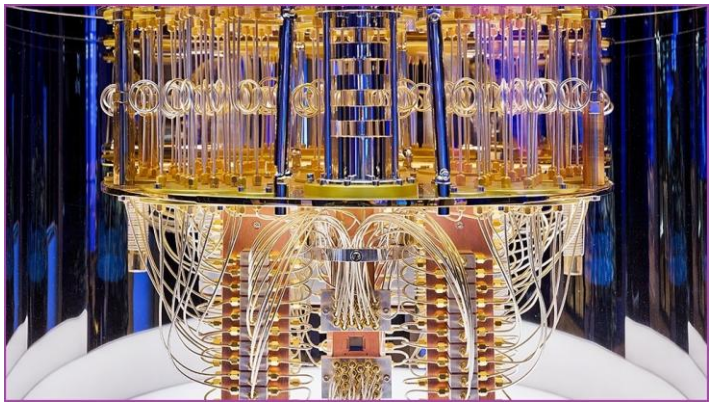
Quantum Machine Learning (QML)

QML: A Rapidly Emerging Field!

- Quantum Machine Learning is an incredibly **new and evolving area of science**.
- Finding comprehensive references is still challenging.
- IBM is currently one of the few organizations offering structured documentation and educational material, primarily through Qiskit Learning.
- QML involves both **hardware** and **software** components.
- Hardware topics such as **qubit technologies**, quantum gates at the physical level, and **noise mitigation** will not be covered.
- In this presentation, I will focus only on the software side.

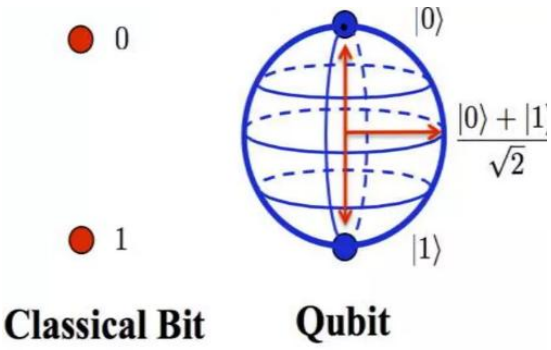
Quantum Processor Unit (QPU)

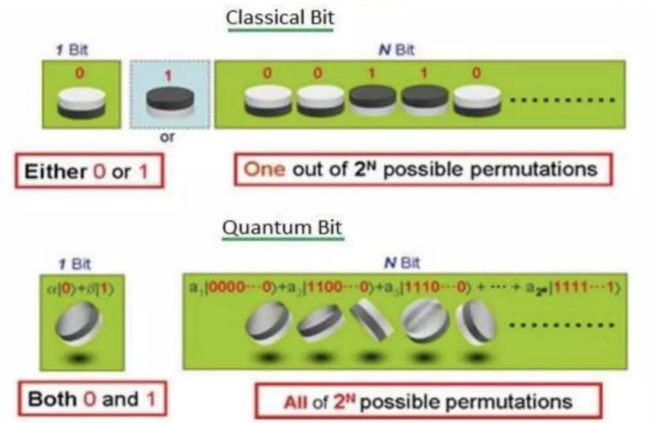
- Quantum computing is an area of computer science that uses the principles of quantum theory. Quantum theory explains the behavior of energy and material on the atomic and subatomic levels.
- Quantum computing uses subatomic particles, such as **electrons or photons** in hardware and computation:
  - **superconducting qubits** (like IBM's and Google's devices) involve electrons indirectly through superconductivity
  - **photonic qubits** directly use the polarization of a single photon to encode quantum information.



Qubits vs Bits

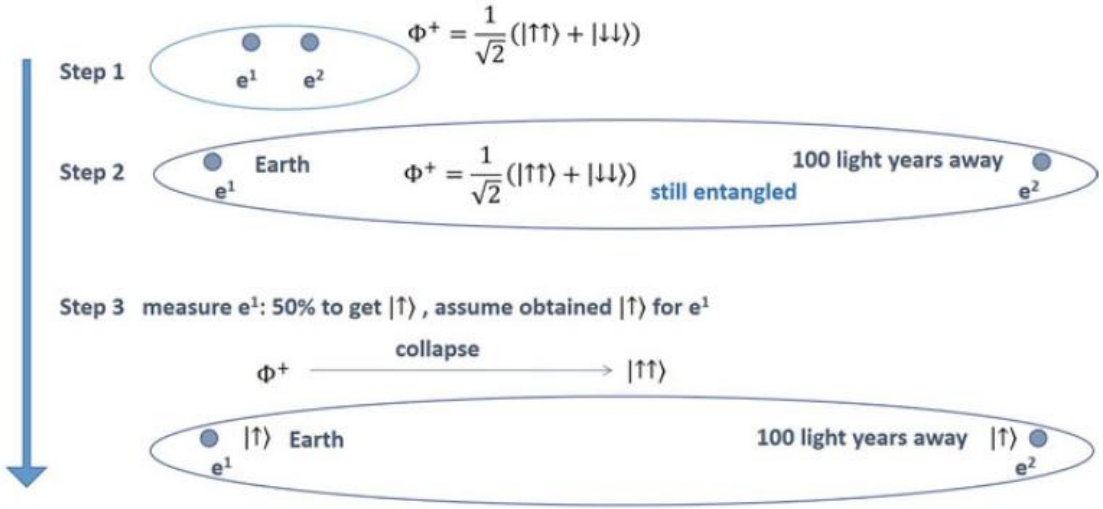
- A qubit is a two-level quantum system where the two basis qubit states are usually written as $|0\rangle$ and $|1\rangle$.
- A qubit can be in state $|0\rangle$ and $|1\rangle$ or (unlike a classical bit) in a linear combination of both states. The name of this phenomenon is superposition ($1/\sqrt{2}\,|0\rangle + 1/\sqrt{2}\,|1\rangle$).
- Qubit superposition allowing **exponential memory representation** with just a few qubits. So, with 3 bits, you can represent only one of these 8 possible combinations, while with 3 qubits, you can represent all 8 combinations simultaneously.





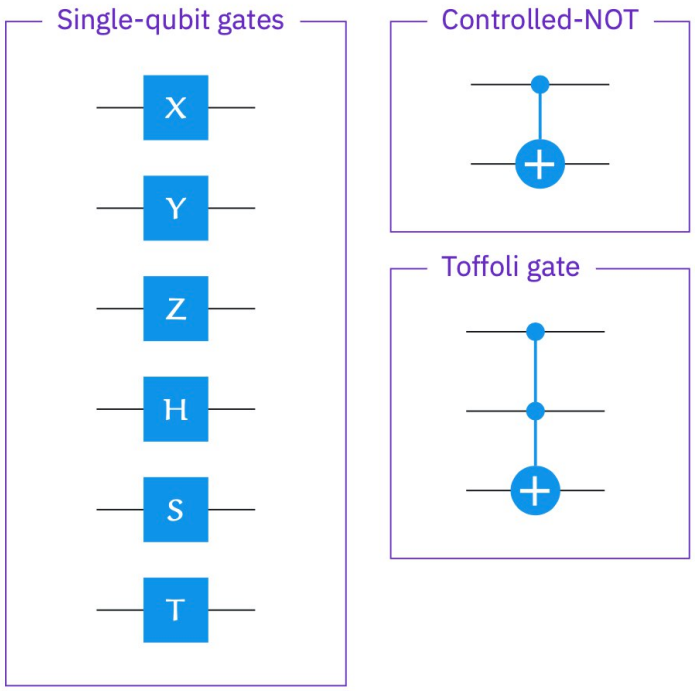
Entanglement

- Quantum entanglement **links two or more qubits** so their states are interdependent, no matter the distance between them.
- Measurement of one entangled qubit instantly determines the state of the other.
- EPR paradox (1935): challenged quantum entanglement, but later experiments confirmed its reality. **Entanglement cannot be used to transmit information**, as measurement outcomes are random and require classical communication to reveal correlations.
- Enables quantum computers to represent and process exponentially more complex information.

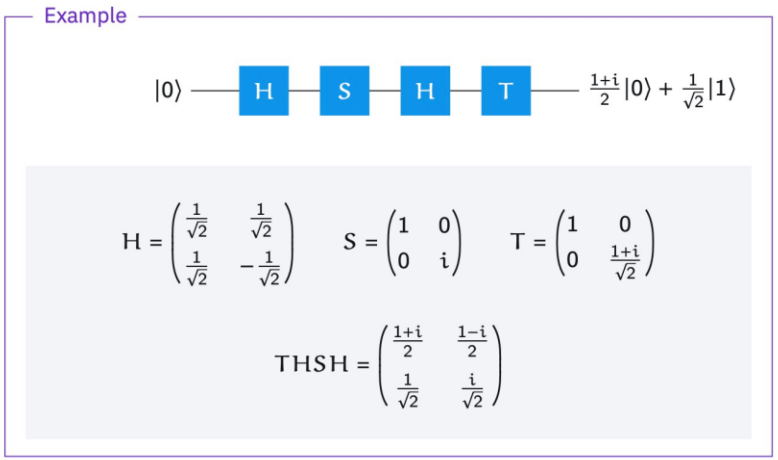


Quantum Operators (unitary matrices)

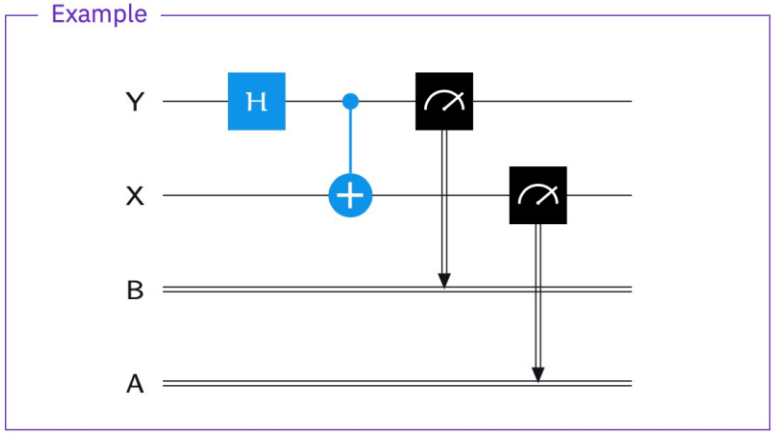
Basic quantum operators



H gate make superposition



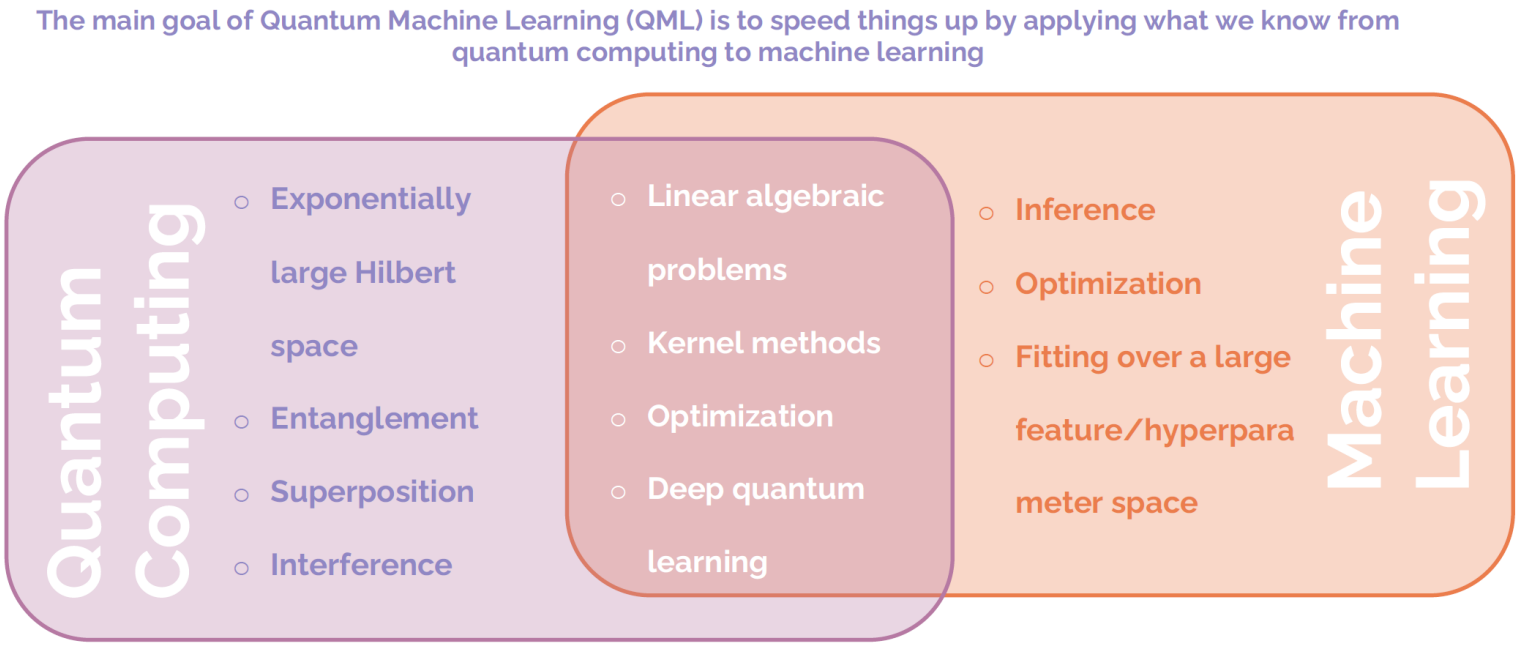
H gate and then CNOT gate to make entanglement



Computation in backup

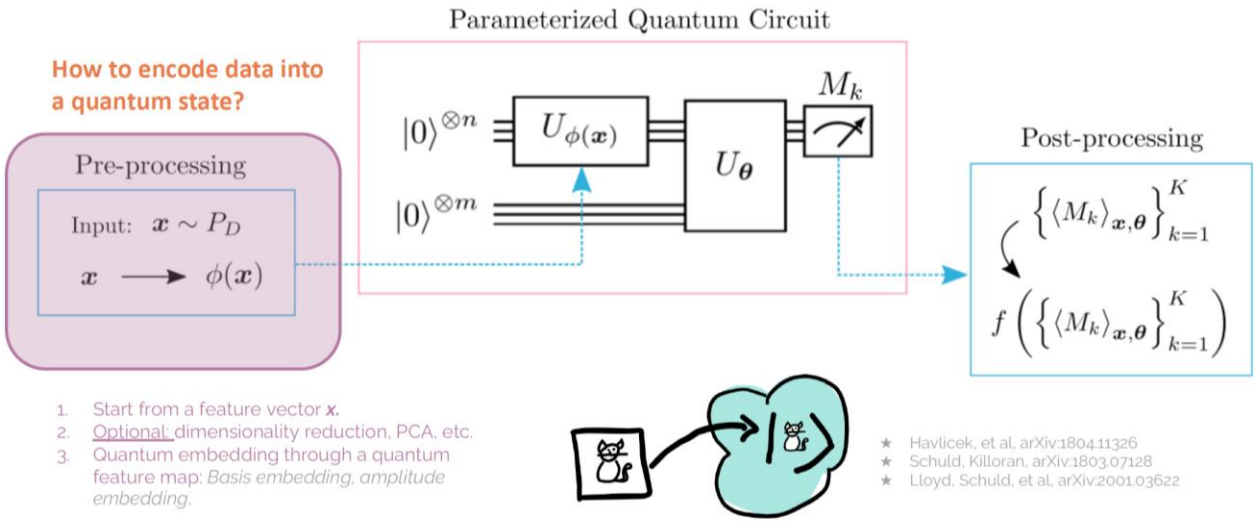
What is QML?

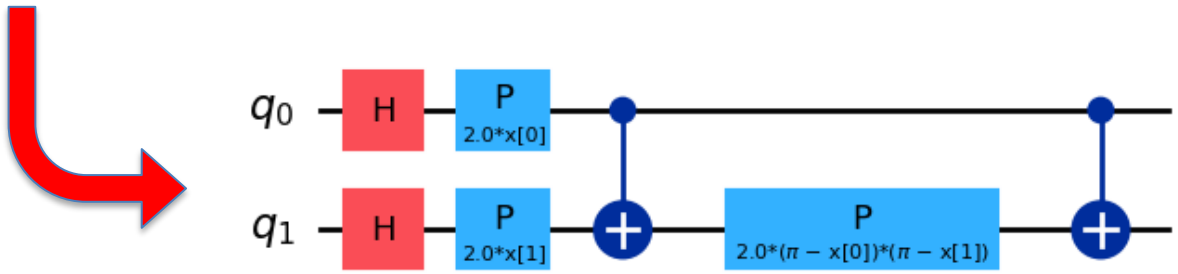
- QML combines quantum computing's ability to handle complex, high-dimensional problems with machine learning's need for optimization and pattern recognition, aiming for faster and more powerful data processing.



Step 1: Feature Mapping (Data Encoding)

- Perform quantum embedding to encode x into a quantum state.

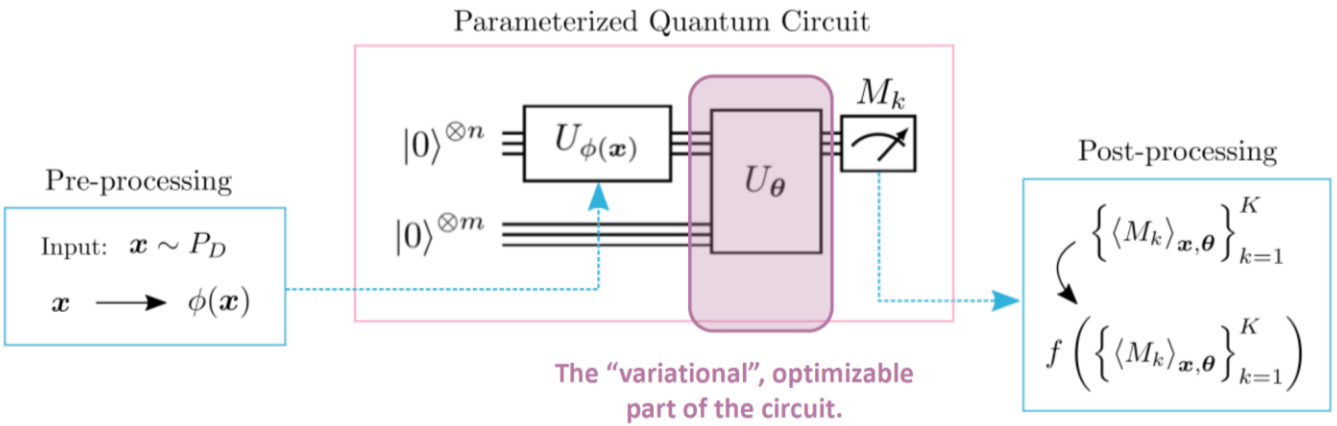




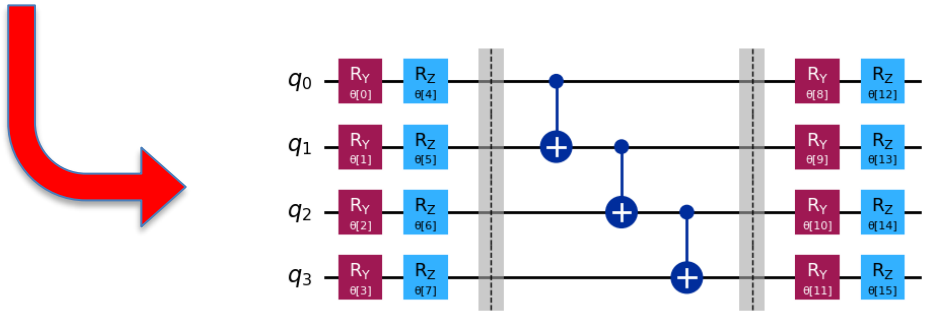
2 qubit angle encoding using H, Phase (P) and CNot gates

Step 2: Variational Ansatz

- Train the model (Ansatz) using quantum circuits with free weights.

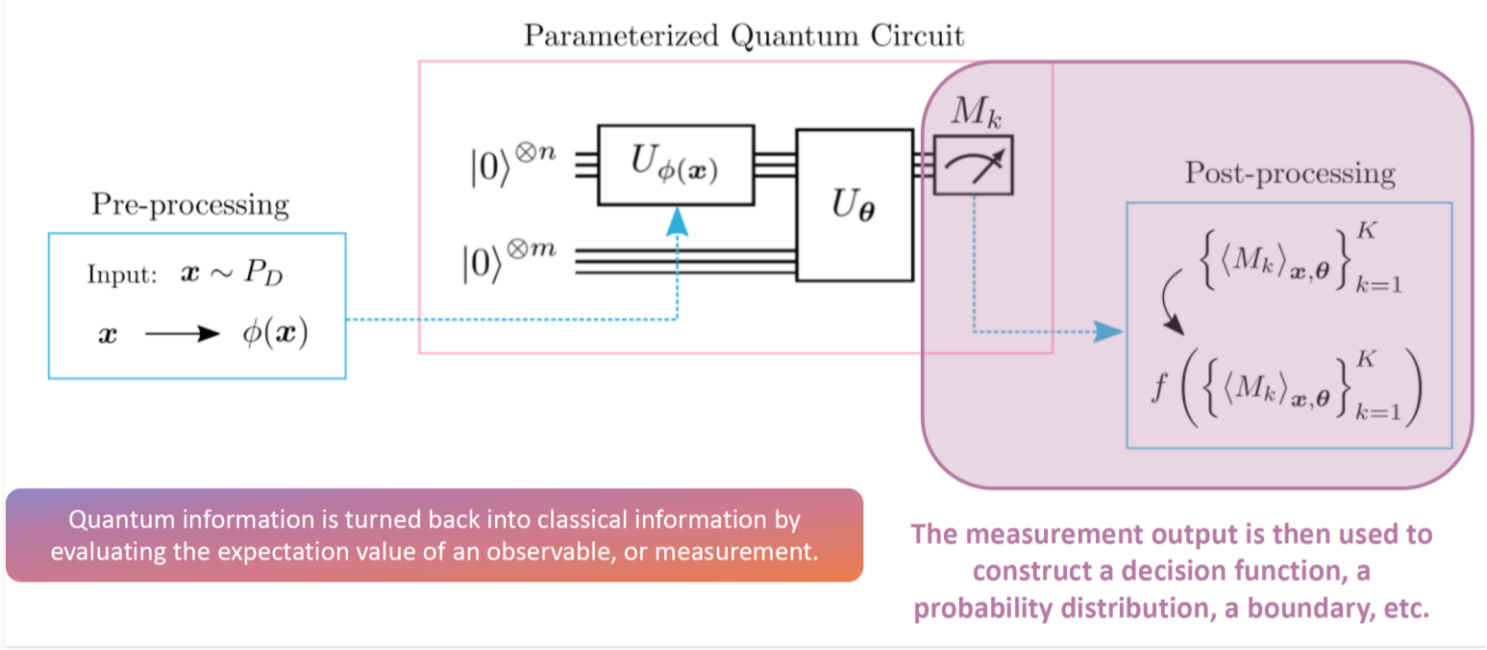


The "guess" or trial function is the unitary U parameterized by a set of free parameters θ that will be updated during training.



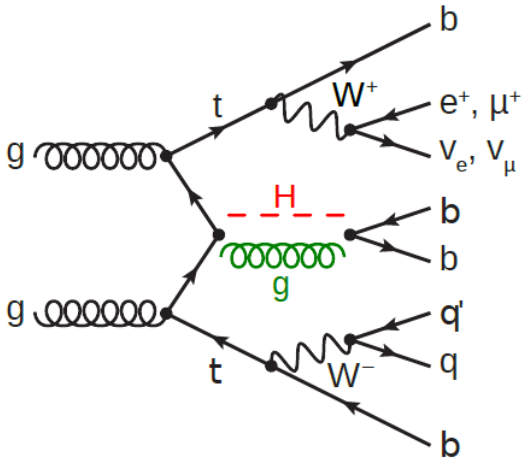
Free parameters inside operators to learn and minimize the loss

Step 3:Post Processing



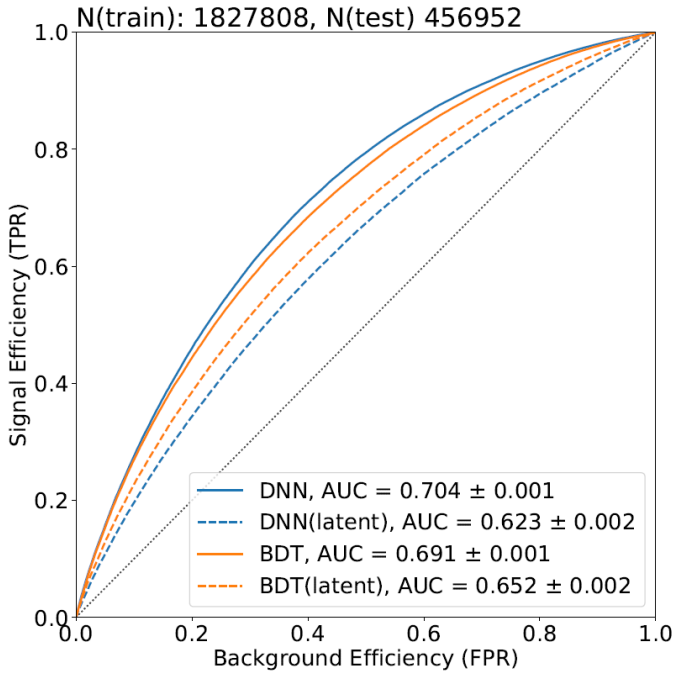
QML in ttH Analysis

- Identify the Higgs boson production in association with to top quark – antiquark pairs.
- Allow studying the **Yukawa coupling of the Higgs** boson in a purely fermionic process.
- Semi leptonic channel to cancel QCD. $\bar{t}tg$ is the dominant background.
- Preselection: $n^{jet} \geq 4, n^{b-tag} \geq 2, n^{lepton} = 1$.
- Principal Component Analysis used to reduce the dimensionality of input features by projecting them onto a set of uncorrelated components that capture the most variance, enabling more efficient input encoding for **quantum models with less qubits**.

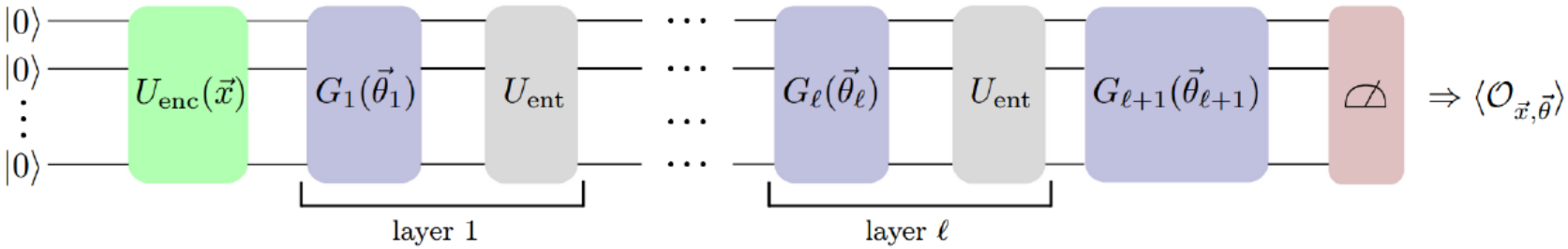


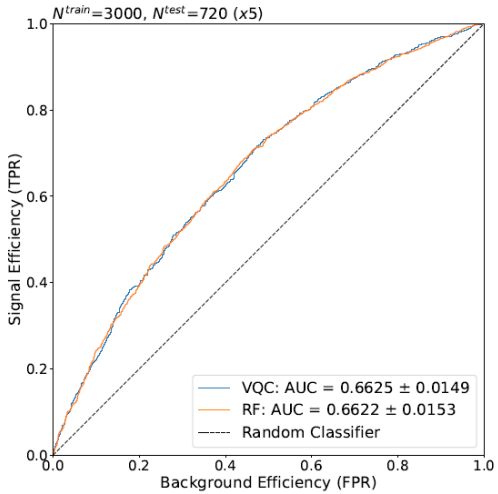
Benchmark Models

- DNN and BDT models are trained in two ways:
  - Using the full 67 input feature
  - Using only 16 latent features from PCA
- Classical models perform well even on reduced feature sets.
- Classical models in were trained on a **large dataset (2M)** enabling stable and high-performance results.

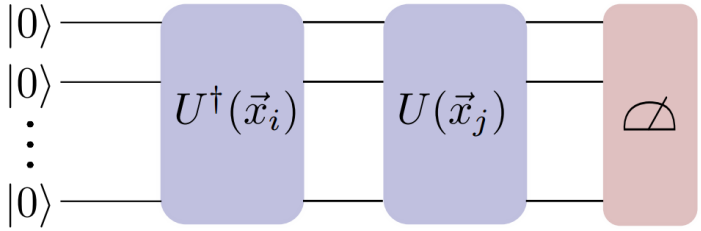


Variational Quantum Classifier



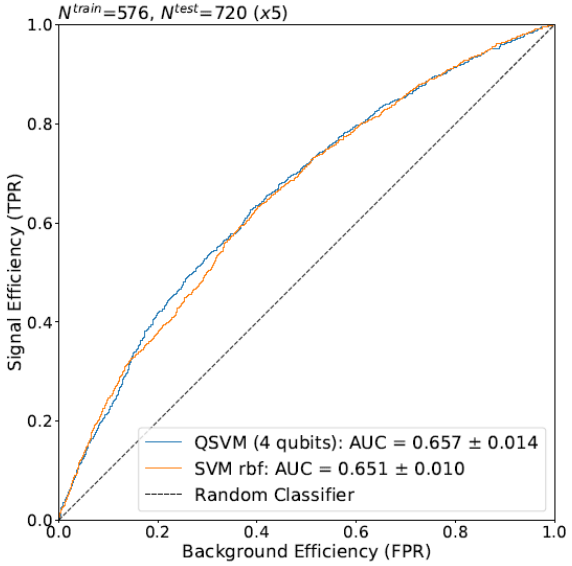


Quantum Support Vector Model



$$\Rightarrow K_{ij} = |\langle 0|U^\dagger(\vec{x}_i)U(\vec{x}_j)|0\rangle|^2$$

$$L(c_1 \dots c_n) = \sum_{i=1}^{n} c_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} y_i c_i (\vec{x}_i \cdot \vec{x}_j) y_j c_j,$$

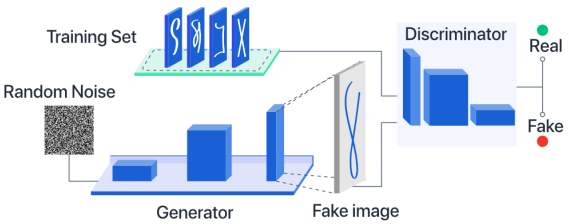


Generative AI

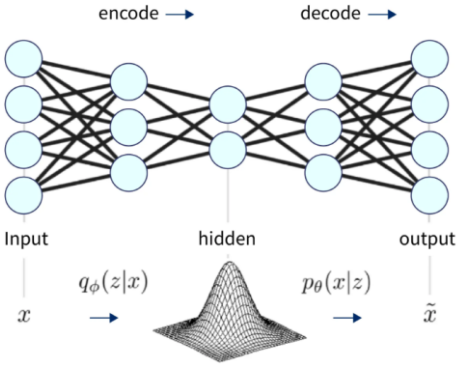
GenAI Importance

- Data Bottleneck:
  - "There's not enough human-made data in the world to train future AI systems." -- Jensen Huang, NVIDIA CEO.
  - GenAI can create synthetic data that mimics **real-world** and **anomaly distributions**.
- GenAI **generates physics events** much faster than traditional simulators.
- GenAI **simulates rare disease** cases for training doctors/AI systems when real data is limited.

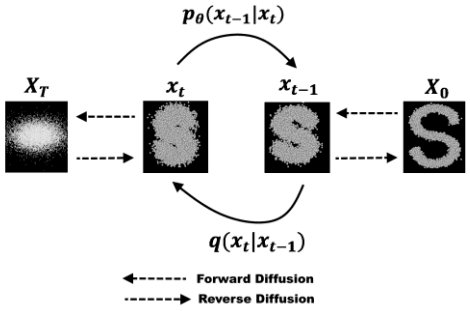
GAN



VAE

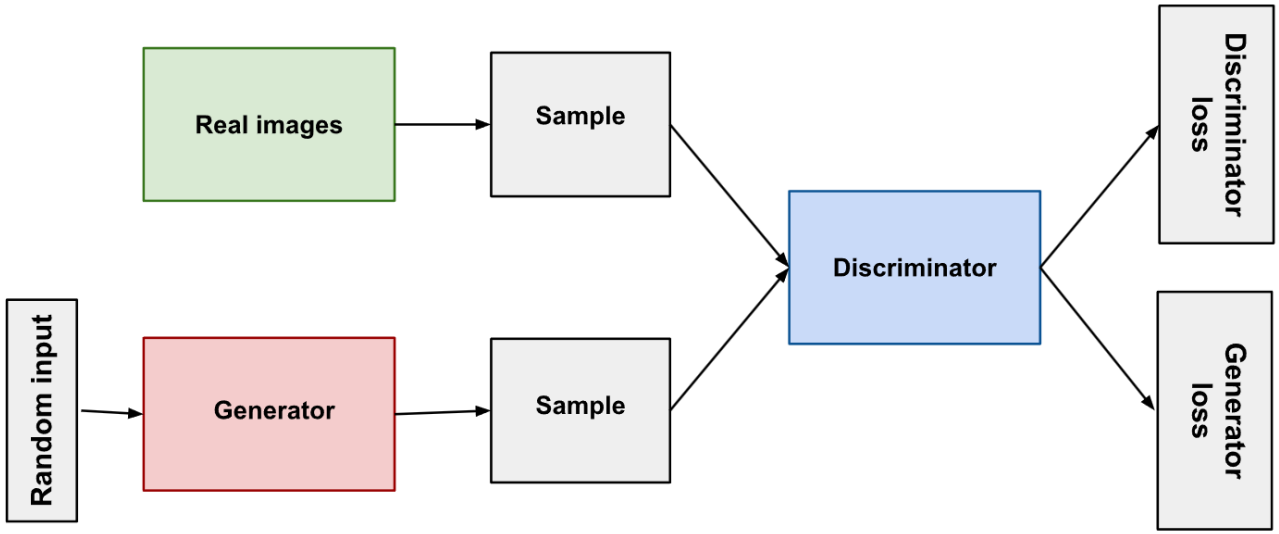


Diffusion



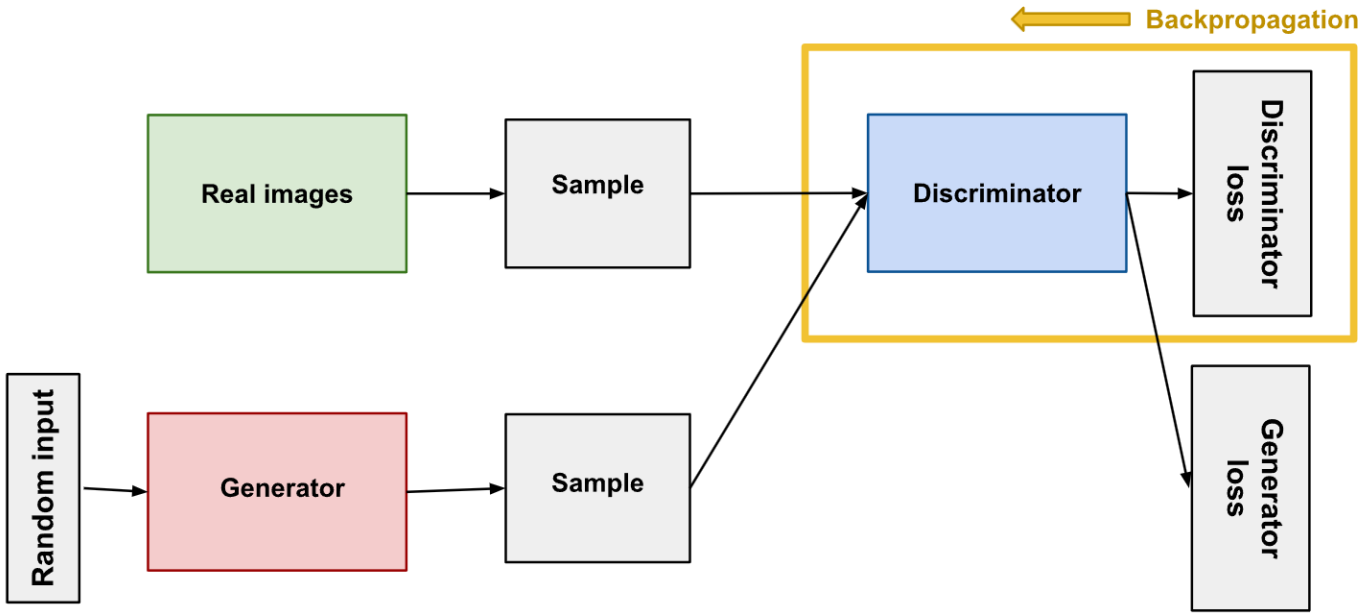
Generative Adversarial Network

- Both the generator and the discriminator are neural networks. The generator output is connected directly to the discriminator input.
- Through backpropagation, the discriminator's classification provides a signal that the generator uses to update its weights.



Discriminator Training

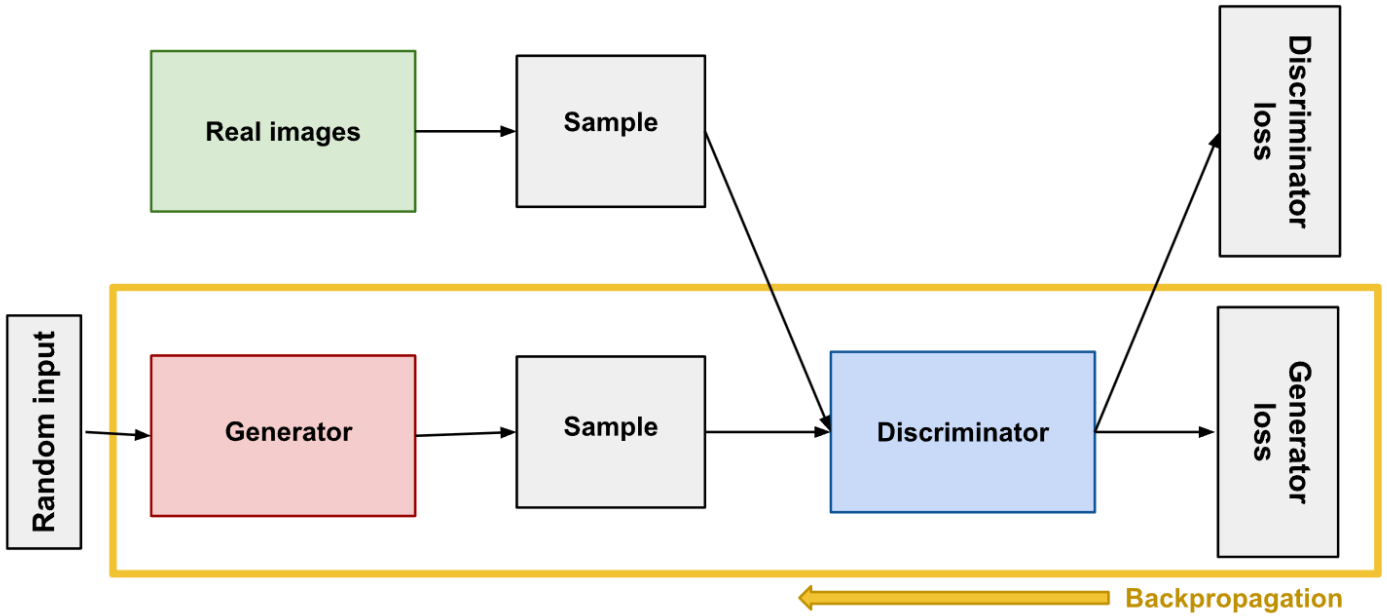
- GAN training proceeds in alternating periods:
  1. The discriminator trains for a lot of epochs. We keep the generator constant during the discriminator training phase. As discriminator training tries to figure out how to distinguish real data from fake



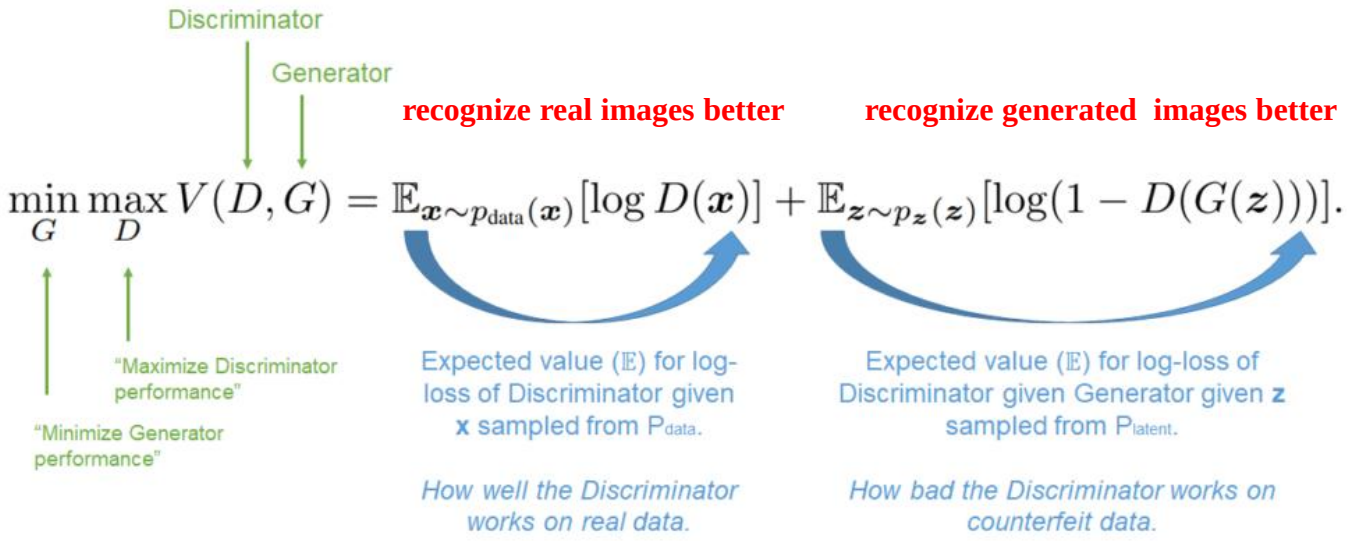
Generator Training

- GAN training proceeds in alternating periods:

  2. The generator trains for one or more epochs. Similarly, we keep the discriminator constant during the generator training phase. Otherwise the generator would be trying to hit a moving target and might never converge.



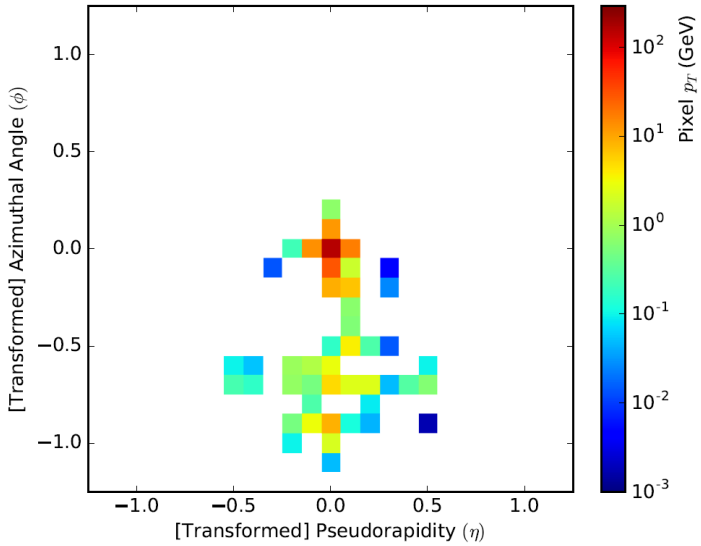
GAN MinMax Loss



- The **discriminator maximizes the loss** (D(x)=1 and D(G(z))=0) by distinguishing real from fake images, aiming for high confidence on real and low on fake.
- The **generator minimizes the loss** (D(G(z))=1) by producing fake images that fool the discriminator into classifying them as real.
- At convergence, the generator perfectly fools the discriminator ($D(x)$=0.5, D(G)=0.5).

GAN for Event Generation

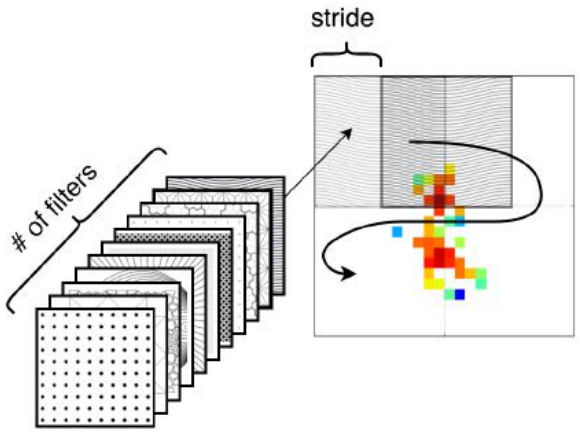
- High Energy Physics simulations, such as those using **Geant4** in ATLAS and CMS, are extremely **computationally expensive**. GAN is faster and scalable
- The dataset consists of jet images, which are 2D histograms of particle energy depositions in the calorimeter, constructed in η-ϕ space grid.
- The data includes boosted W boson jets (signal) and QCD jets (background), both generated using Pythia at14 TeV.



Convolution vs Locally connected

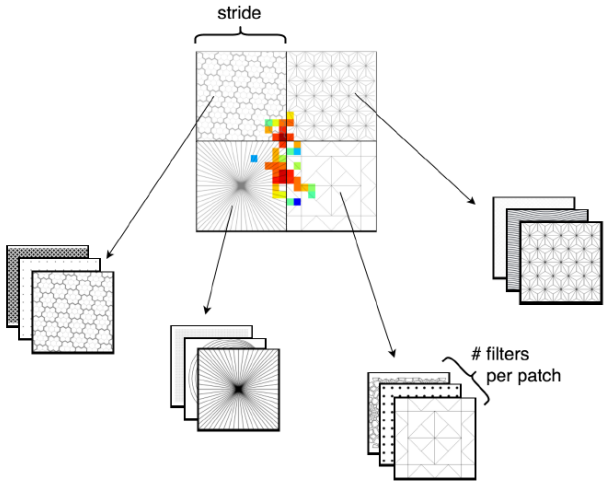
- **CNN layers:**

1. Use shared filters that scan across the entire image.
2. Assume translational invariance — the same feature can appear anywhere.
3. Less suitable when location itself encodes physical meaning.



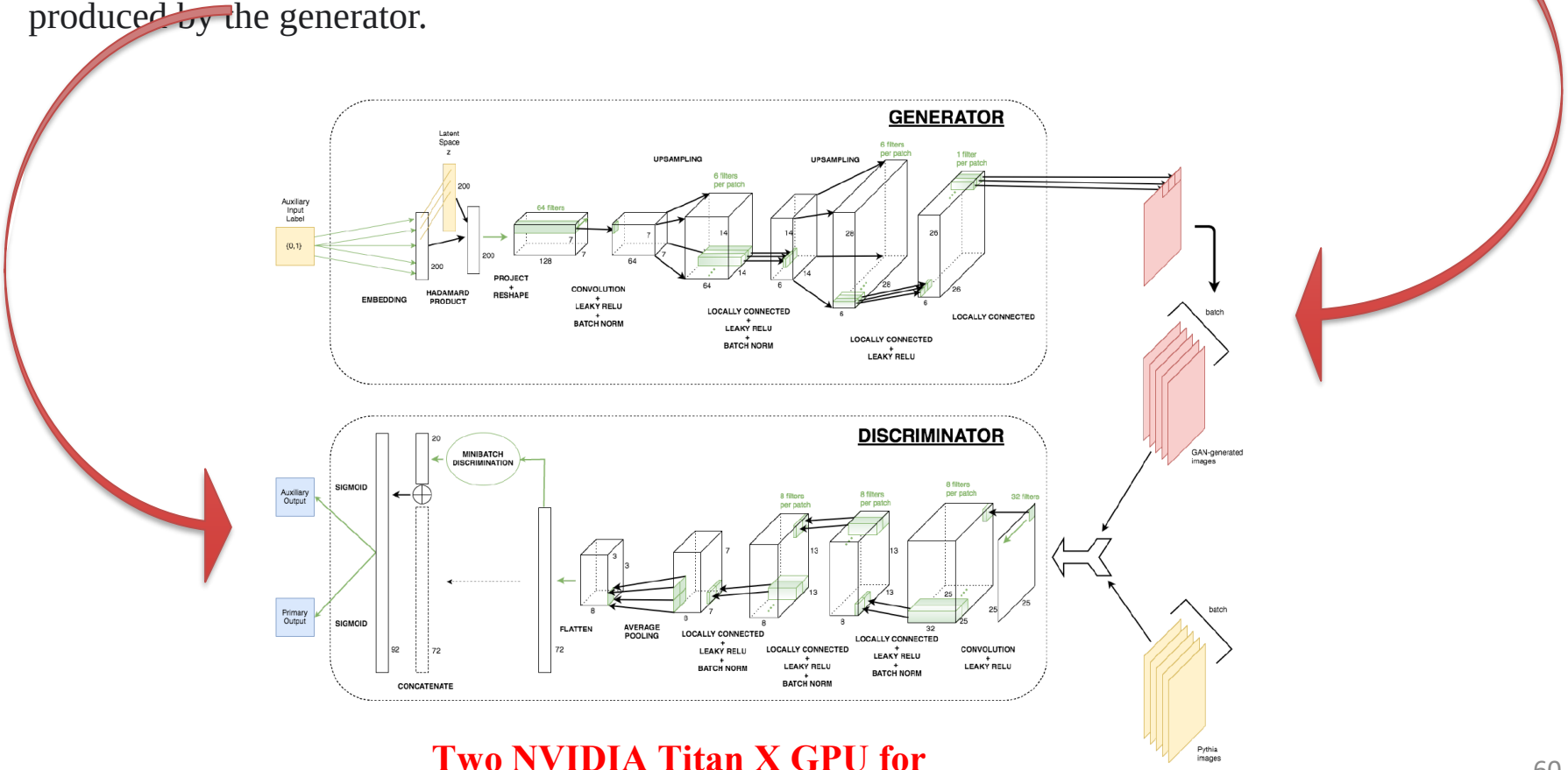
- **Locally connected layers:**

1. Similar to CNNs but do not share weights across positions.
2. Learn unique filters for each spatial region of the input.
3. Better suited when spatial position has physical interpretation
4. More parameters, higher computation cost



GAN Architecture

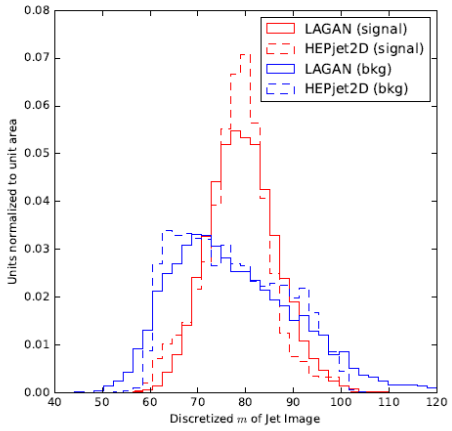
- **Generator**: Starts from a 100-dimensional vector and uses up-sampling to produce realistic jet images of size 25×25 in (η, φ) space, simulating both signal and background events
- **Discriminator**: CNN to distinguish real simulated events (from Geant4) from fake events produced by the generator.



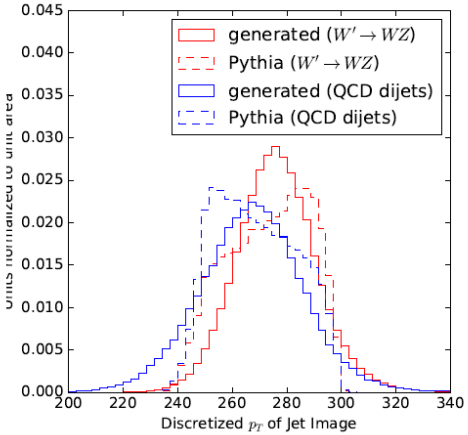
Two NVIDIA Titan X GPU for training!

GAN Event Distributions

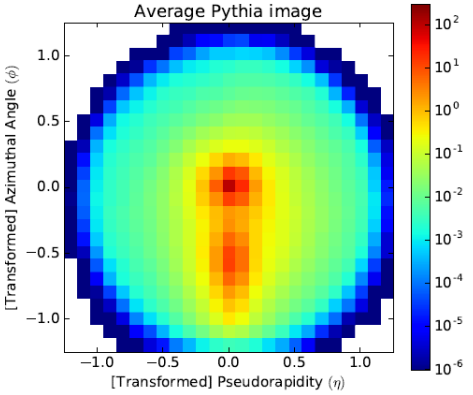
Mass distribution



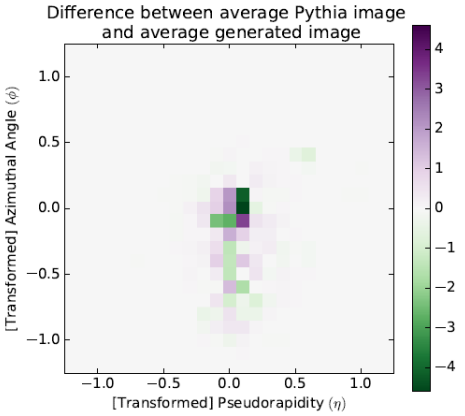
p_T distribution



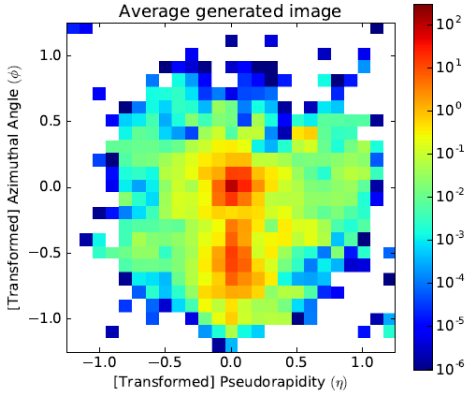
True (simulated) average energy deposition



Difference

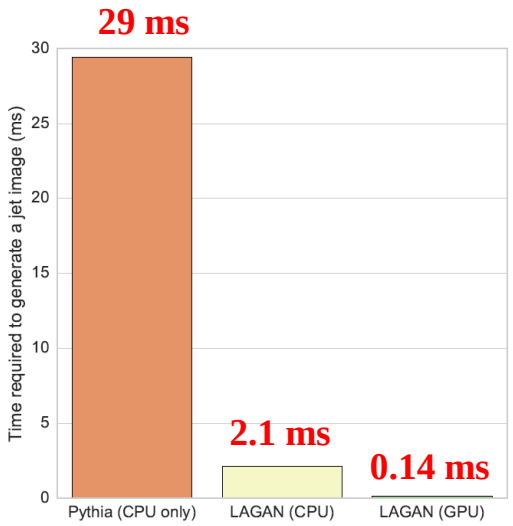


Generated (GAN) average energy deposition



GAN Speed-up

Method	Hardware	# events / sec	Relative Speed-up
Pythia	CPU	34	1
LAGAN	CPU	470	14
LAGAN	GPU	7200	210



LAGAN is much faster!

Summary

- Deep Neural Networks: introduced basic structure and training process using forward and backward propagation. Applied to tasks like classification of events and SUSY-background separation.
- Convolutional Neural Networks: explained how CNNs exploit spatial structure of detector data. Used in jet image analysis from Higgs boson decay and event classification.
- Quantum Machine Learning: discussed principles of qubits and quantum circuits. Highlighted potential applications in high-dimensional data and speedup of certain learning tasks.
- Generative Adversarial Networks: covered how GANs generate realistic physics events. Used for fast detector simulation and event classification.

ML Roadmap

1. Introduction to machine learning:
   - non neural network models: Google course
   - neural network models specially DNN and CNN: Coursera
   - MIT neural network course: link

2. Graph Neural Network: Stanford CS224 course

3. Recurrent neural network: Stanford lecture

4. Generative Deep Learning: O'reilly book

5. Quantum Computation: IBM Qiskit quantum course

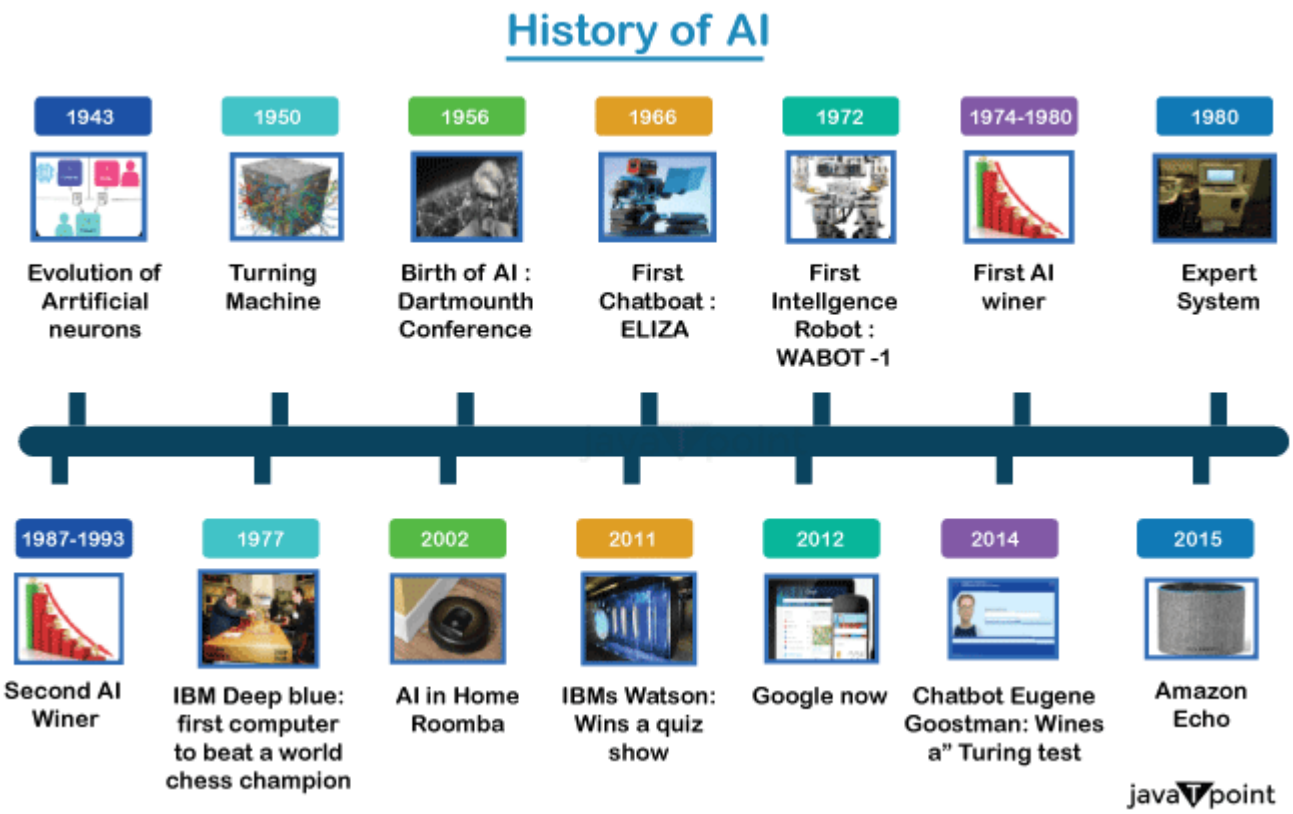
6. Quantum Machine Learning: IBM QML course

7. Tensorflow Tutorial

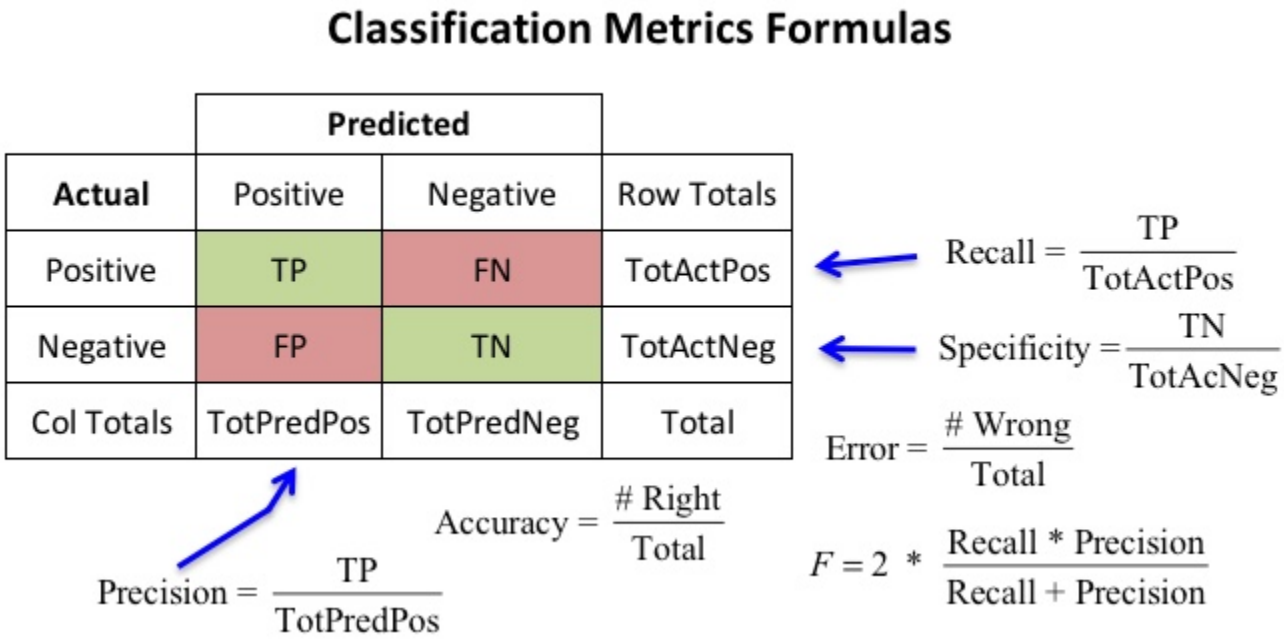
8. Pytorch Tutorial

Backup

AI History


History of AI
1943
Evolution of Arrtificial neurons
1950
Turning Machine
1956
Birth of AI : Dartmounth Conference
1966
First Chatboat : ELIZA
1972
First Intellgence Robot : WABOT -1
1974-1980
First AI winer
1980
Expert System
1987-1993
Second AI Winer
1977
IBM Deep blue: first computer to beat a world chess champion
2002
AI in Home Roomba
2011
IBMs Watson: Wins a quiz show
2012
Google now
2014
Chatbot Eugene Goostman: Wines a" Turing test
2015
Amazon Echo
javaTpoint

Classification metrics in ML



SUSY Definition of Key Variables

$\Delta\phi_{\min}^{4j}$ to suppress multijets in which E_T^{miss} is aligned with one jet:

$$\Delta\phi_{\min}^{4j} = \min(|\phi_1 - \phi_{E_T^{miss}}|, \dots, |\phi_4 - \phi_{E_T^{miss}}|)$$

Inclusive effective mass, to select highly energetic events:

$$m_{eff}^{incl} = \Sigma_{i \le n}\, p_T^{j_i} + \Sigma_{j \le m}\, p_T^{lep_j} + E_T^{miss}$$

m_T to remove semileptonic $t\bar{t}$ and W+jets events (region ≥ 1 lepton) :

$$m_T = \sqrt{2 p_T^l E_T^{miss} (1 - \cos\Delta\phi(\vec{p}_T^{miss}, \vec{p}_T^{\,l}))}$$

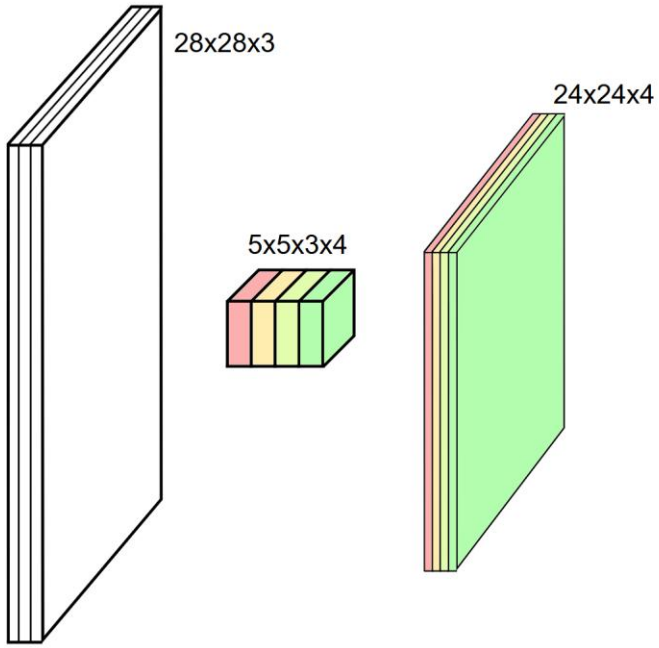
$m_{T,min}^{b-jets}$ min transverse mass between E_T^{miss} and three leading b-jets:

$$m_{T,min}^{b-jets} = \min_{i \le 3}\left(\sqrt{2 p_T^{b-jet_i} E_T^{miss} \left(1 - \cos\Delta\phi\left(\vec{p}_T^{miss}, \vec{p}_T^{b-jet_i}\right)\right)}\right)$$

$M_J^{\Sigma,4}$ sum of the mass of re-clustered jets (higher for Gtt signal):

$$M_J^{\Sigma,4} = \Sigma_{i \le 4} m_{J,i}$$

CNN Stacking



- Kernel size aka receptive field (usually 1, 3, 5, 7, 11)
- Output dimension: `length - kernel_size + 1`

CNN Variables

📐 Equation (5.1): Transverse Momentum of the Jet

$$p_T^2(I) = \left(\sum_i I_i \cos(\phi_i)\right)^2 + \left(\sum_i I_i \sin(\phi_i)\right)^2$$

Explanation:

- p_T is the total transverse momentum of the entire jet.
- Each pixel contributes to the jet's total x- and y-momentum:
  - $p_x = \sum I_i \cos(\phi_i)$
  - $p_y = \sum I_i \sin(\phi_i)$
- So the total p_T is the Euclidean norm:

$$p_T = \sqrt{p_x^2 + p_y^2}$$

🧠 **Physically**, this gives the total transverse momentum vector of the jet, using the per-pixel contributions.

⚖️ Equation (5.2): Invariant Mass of the Jet

$$m^2(I) = \left(\sum_i I_i\right)^2 - p_T^2(I) - \left(\sum_i I_i \sinh(\eta_i)\right)^2$$

Explanation: This is a version of the energy-momentum relation:

$$m^2 = E^2 - p_T^2 - p_z^2$$

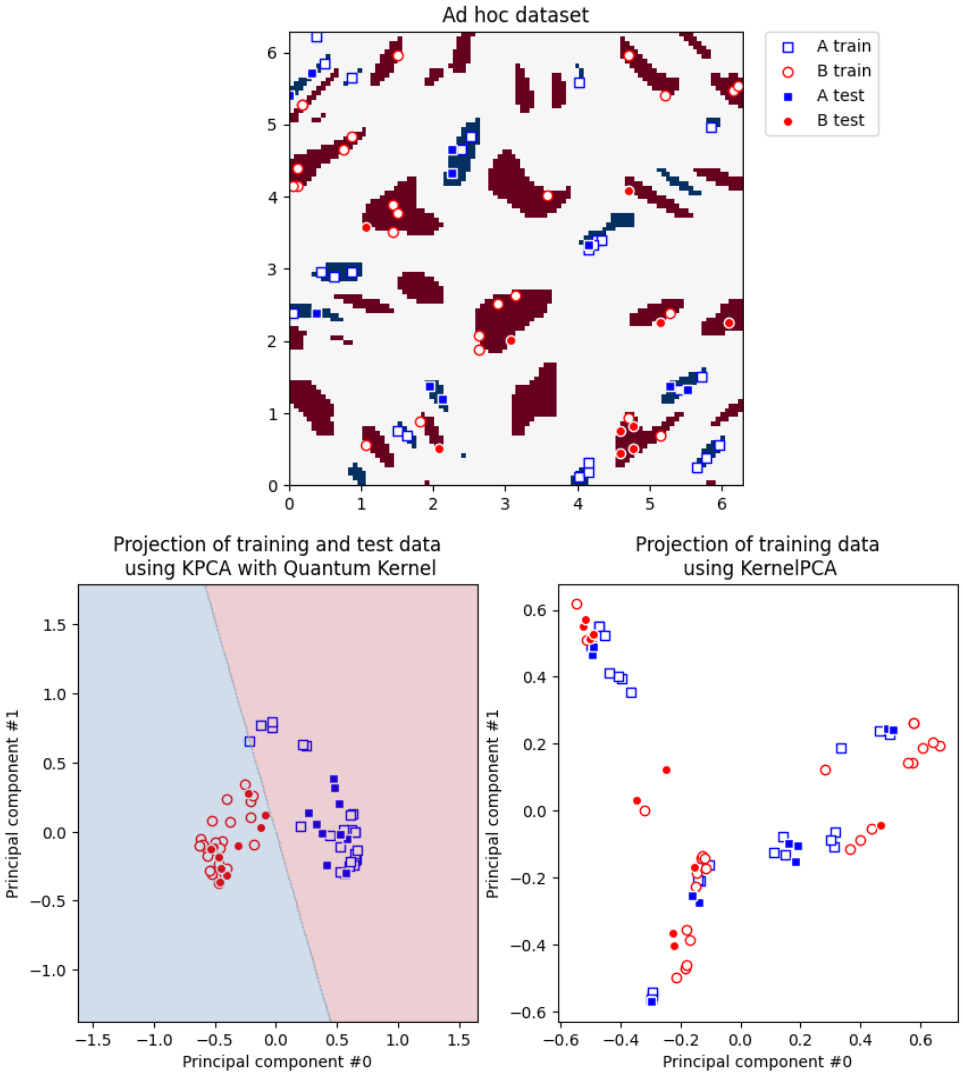
where:

- $\sum_i I_i$ approximates the **total energy** (not exact, but in these images it works since energies are projected using p_T).
- $\sum_i I_i \sinh(\eta_i)$ corresponds to the **longitudinal momentum** p_z, using the identity:

$$p_z = p_T \sinh(\eta)$$

🧠 **So**, the total mass is reconstructed from the sum of pixel intensities and their geometric arrangement in the $\eta - \phi$ plane.

QSVM Power


Ad hoc dataset
A train
B train
A test
B test
Projection of training and test data
using KPCA with Quantum Kernel
Principal component #1
Principal component #0
Projection of training data
using KernelPCA
Principal component #1
Principal component #0

Bell State Computation

🎯 **Goal:**

Produce the Bell state

$$|\Phi^+\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$$

🧮 **Step 1: Initial state**

$$|00\rangle = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

🧮 **Step 2: Apply Hadamard to the first qubit**

The **Hadamard gate** acts on a single qubit:

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

To apply it to the **first qubit** in a 2-qubit system, use the tensor product:

$$H \otimes I = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \end{bmatrix}$$

Apply it to $|00\rangle$:

$$(H \otimes I)|00\rangle = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \end{bmatrix} = \frac{1}{\sqrt{2}}(|00\rangle + |10\rangle)$$

🧮 **Step 3: Apply the CNOT gate**

CNOT gate matrix (control = qubit 0, target = qubit 1):

$$\text{CNOT} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Apply it to the previous result:

$$\text{CNOT} \cdot \left(\frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \end{bmatrix} \right) = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix} = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$$

GAN MinMax Loss

Why the Generator Minimizes and the Discriminator Maximizes

The minimax loss in GANs is defined as a two-player game, where the generator (G) and discriminator (D) have opposing goals. The loss function, as given in the paper (Section 3, Equation 3.1), is:

$$\mathcal{L}(D, G) = \mathbb{E}_{I \sim p_{\text{data}}(I)}[\log D(I)] + \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z)))]$$

The GAN training objective is:

$$\min_G \max_D \mathcal{L}(D, G)$$

- **Discriminator's Goal ($\max_D$)**: The discriminator wants to **maximize** $\mathcal{L}$. It does this by:
  - Making $D(I) \approx 1$ for real images ($I \sim p_{\text{data}}$), so $\log D(I) \approx \log 1 = 0$, contributing a large (less negative) value to the first term.
  - Making $D(G(z)) \approx 0$ for fake images ($G(z)$), so $\log(1 - D(G(z))) \approx \log(1 - 0) = \log 1 = 0$, contributing a large (less negative) value to the second term.
  - A high $\mathcal{L}$ (closer to 0) means D is good at distinguishing real jet images (e.g., Pythia-generated) from fake ones (LAGAN-generated).
- **Generator's Goal ($\min_G$)**: The generator wants to **minimize** $\mathcal{L}$. It only affects the second term, $\mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z)))]$, because G generates $G(z)$. To minimize $\mathcal{L}$:
  - G wants $D(G(z)) \approx 1$, meaning the discriminator thinks fake images are real. This makes $\log(1 - D(G(z))) \approx \log(1 - 1) = \log 0 \to -\infty$, driving the second term (and thus $\mathcal{L}$) to a very negative value.
  - By making fake jet images look real, G "fools" D, reducing $\mathcal{L}$.

GAN Features

One of the unique properties of high energy particle physics is that we have a library of useful *jet observables* that are physically motivated functions $f : \mathbb{R}^{25\times 25} \to \mathbb{R}$ whose features are qualitatively (and in some cases, quantitatively) well-understood. We can use the distributions of these observables to assess the abilty of the GAN to mimic Pythia. Three such features of a jet image I are the mass m, transverse momentum p_{T}, and n-subjettiness τ_{21} [41]:

$$p_{\mathrm{T}}^2(I) = \left(\sum_i I_i \cos(\phi_i)\right)^2 + \left(\sum_i I_i \sin(\phi_i)\right)^2 \tag{5.1}$$

$$m^2(I) = \left(\sum_i I_i\right)^2 - p_{\mathrm{T}}^2(I) - \left(\sum_i I_i \sinh(\eta_i)\right)^2 \tag{5.2}$$

$$\tau_n(I) \propto \sum_i I_i \min_a \left(\sqrt{(\eta_i - \eta_a)^2 + (\phi_i - \phi_a)^2}\right), \tag{5.3}$$

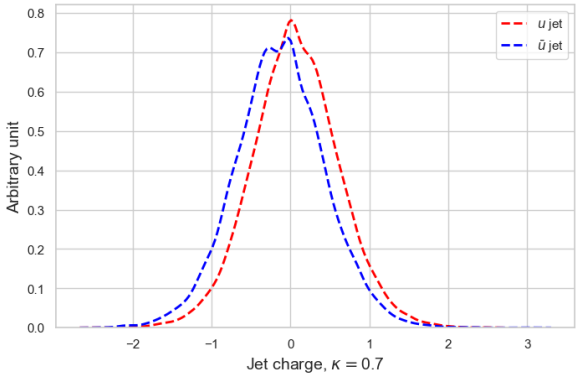
$$\tau_{21}(I) = \tau_2(I)/\tau_1(I), \tag{5.4}$$

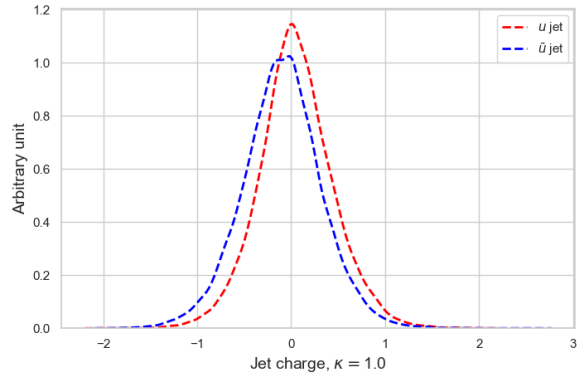
Application of Deep learning to Jet charge

- The jet charge method is crucial in verifying the top quark charge, heavy boson identification (W′/Z′), quark vs. gluon jet discrimination.
- This study explores **classical deep learning** models like DNN, CNN, GNN and **quantum ML models** for improved jet charge classification.
- The jet charge observables are defined as (where is a tunable parameter affecting charge sensitivity):

$$Q_j = \frac{1}{(p_T^{\text{jet}})^\kappa} \sum_{i \in \text{Tr}} Q_i (p_T^i)^\kappa$$

$$Q_{3,\kappa} = \frac{\sum_{i \in Tr} q_i \left|\Delta\eta_i\right|^\kappa}{\sum_{i \in Tr} \left|\Delta\eta_i\right|^\kappa}$$





Momentum weighted jet charge distribution

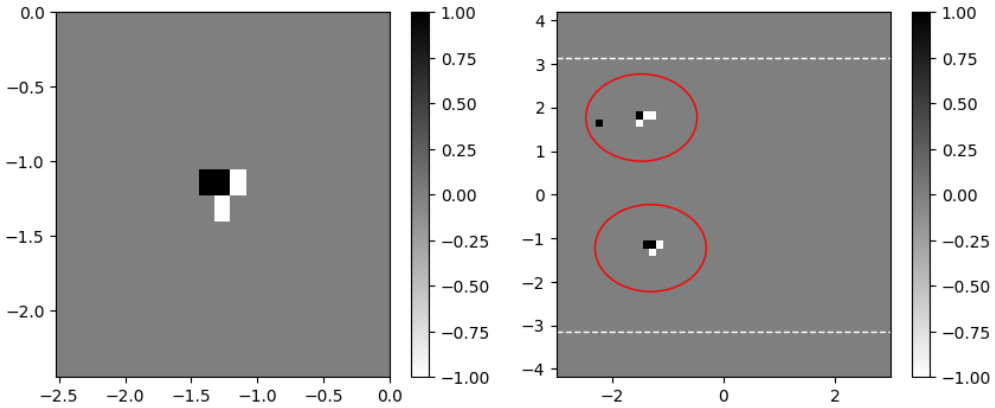
Convolutional NN and Graph NN

Convolutional Neural Networks (CNNs):

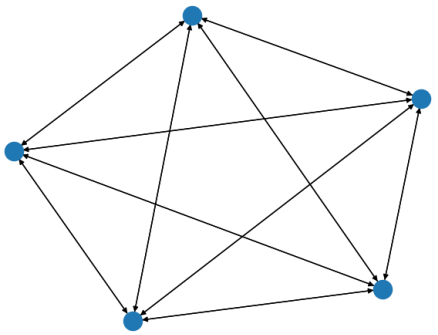
- CNNs process jet images by analyzing pixel charge distributions in **η-φ space**.
- They capture **spatial correlations** of energy deposits to classify jet charge.
- Can differentiate between quark-initiated and gluon-initiated jets using learned spatial features.

Graph Neural Networks (GNNs):

- GNNs model jets as graphs, where **tracks are nodes** and **edges encode relationships**.
- Capture relational and topological information beyond fixed-grid structures.
- Effective in handling **variable-sized track information** per jet.



Pixelated representation of jets in $\eta - \phi$ space

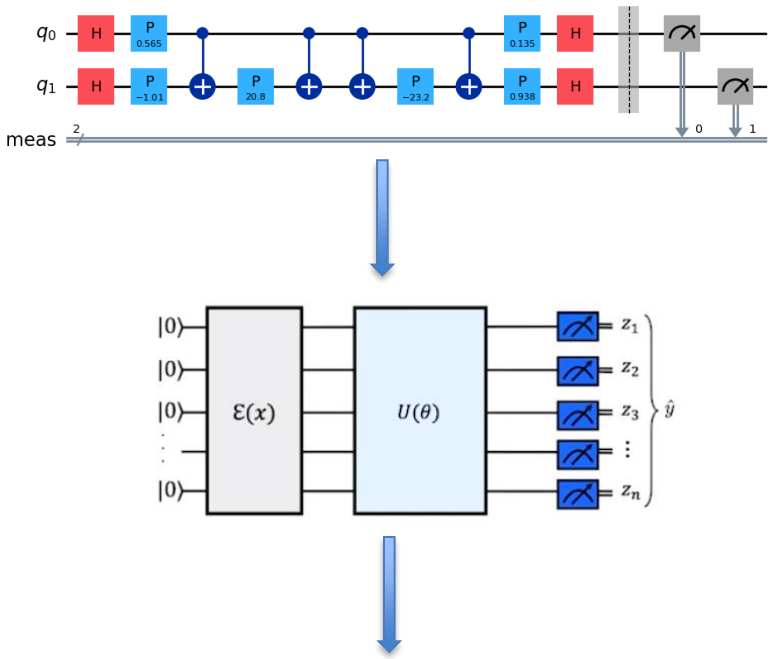


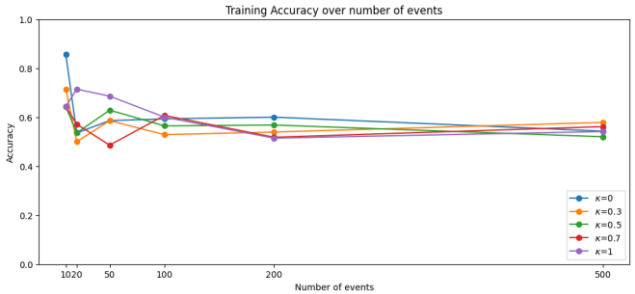
Graph representation of a leading jet

Quantum ML application

- Quantum Feature Map: Maps classical jet charge data into a higher-dimensional quantum Hilbert space using parametrized quantum circuits, enabling more expressive representations.
- Quantum Kernel: Compute similarity between quantum-embedded jets, allowing for efficient discrimination using quantum support vector machines.
- Challenges: Noisy quantum hardware, limited qubit connectivity, and circuit depth constraints impact practical implementation.

Quantum feature map and kernel



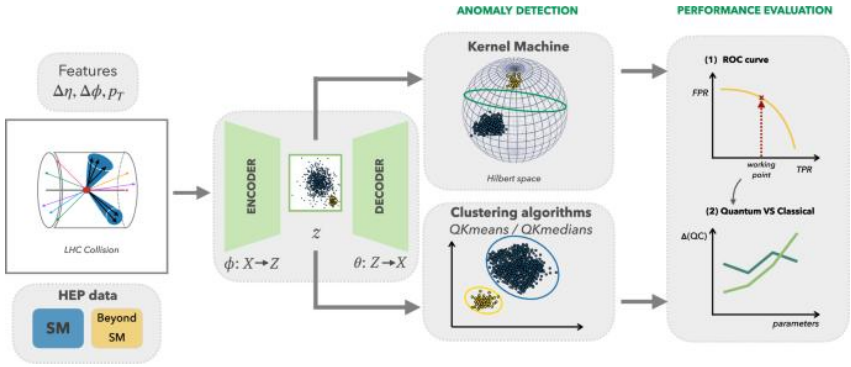


Accuracy for VQC in few events

Accuracy for QSVM in few events

Anomaly detection using Gen AI

- Train Autoencoder or variational AE on SM background events and reconstruct them well.
- New Physics events, which deviate from learned patterns, lead to high reconstruction errors, signaling potential anomalies.



- In GAN-based anomaly detection, the **generator** learns to produce events that resemble Standard Model (SM) data, while the **discriminator** is trained to distinguish between real and generated events.
- if the discriminator assigns a high anomaly score (i.e., the event is unlike both real and generated SM events), it may indicate a **Beyond Standard.**