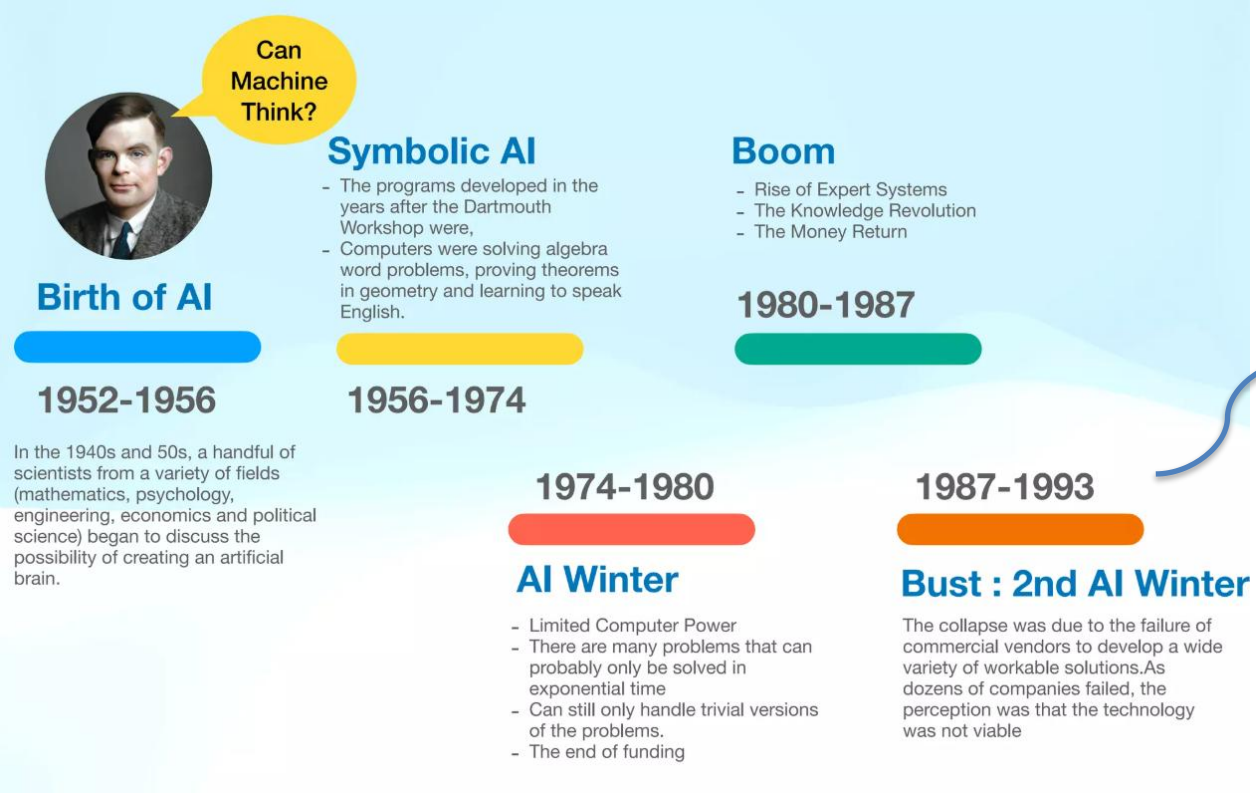
Journal Club

Meisam Ghasemi Bostanabad

School of Particles and Accelerators:
2025-4-14

historical evolution of Artificial Intelligence



And then rise of Neural network!

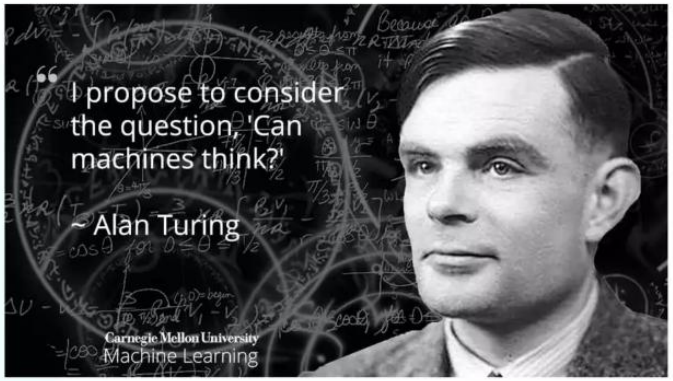
- Artificial Intelligence (AI) is the simulation of human intelligence in machines that are designed to think, learn, and make decisions.
- Machine Learning (ML) is an approach to AI where the algorithms **learn from the data**.

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!

Basis of Neural Network

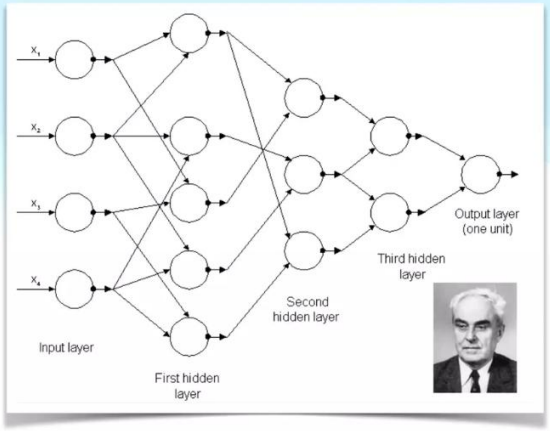
1965 - Anatolii Gershman, Alexey Ivakhnenko, Valentin Lapa

Deep Learning - Multi Layered Perceptron

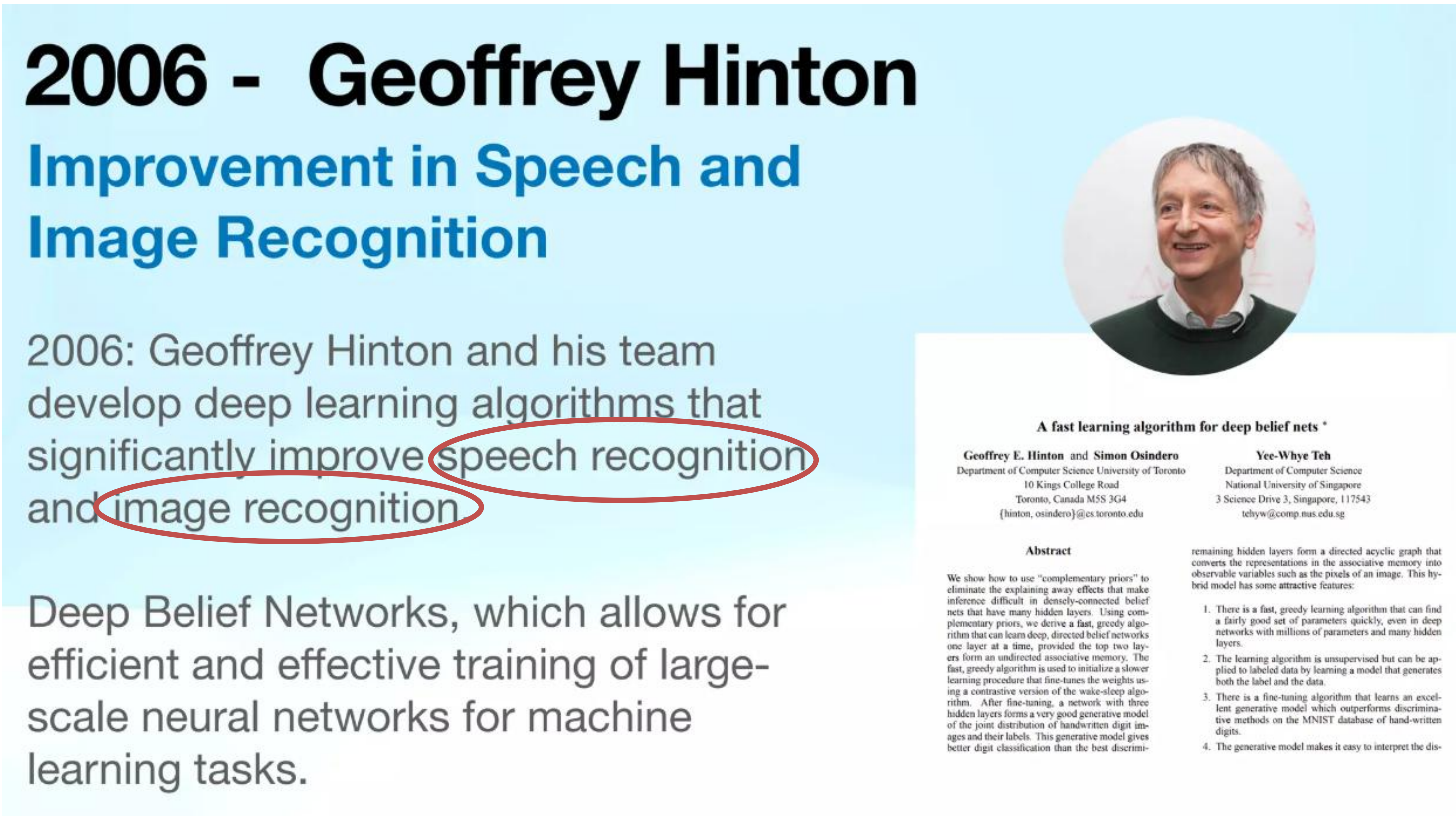
1965 : **The Group Method of Data Handling** (GMDH) is a data-driven approach to modeling that is based on a multi-layered architecture of interconnected polynomial models.

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.**

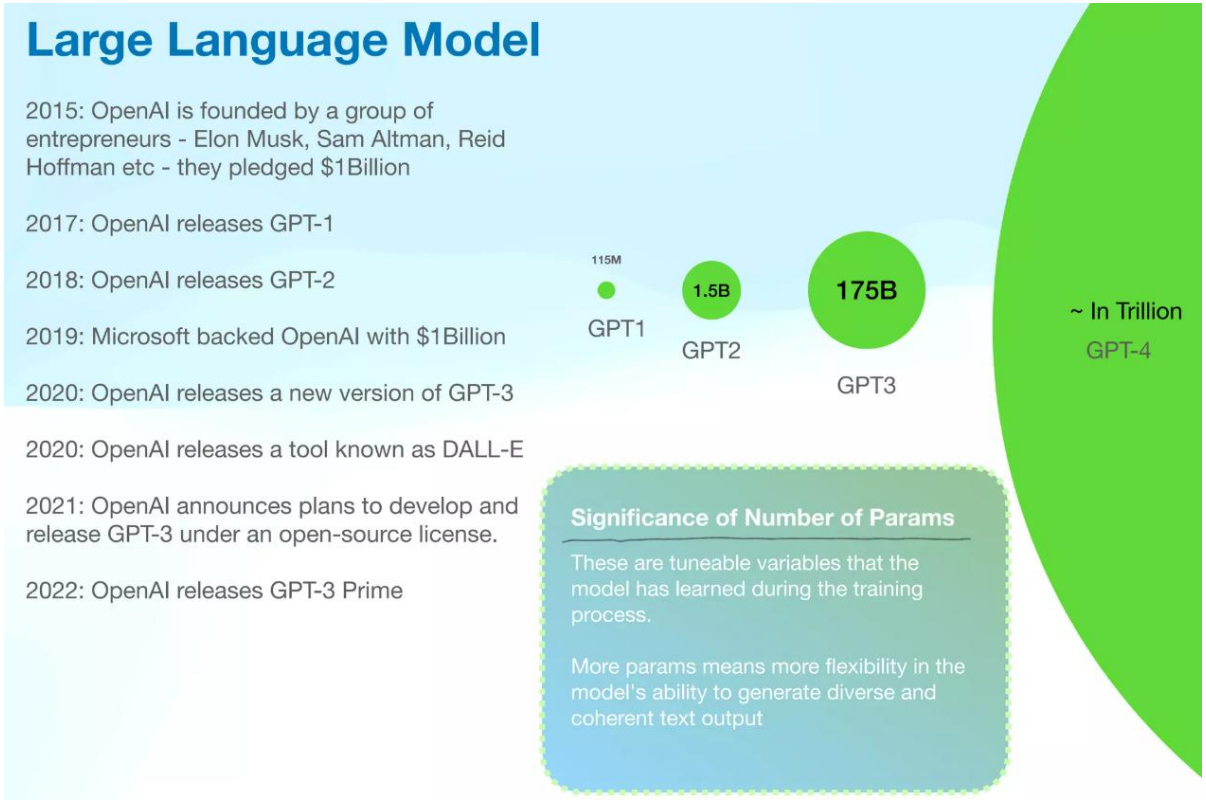


Hinton’s application of NN



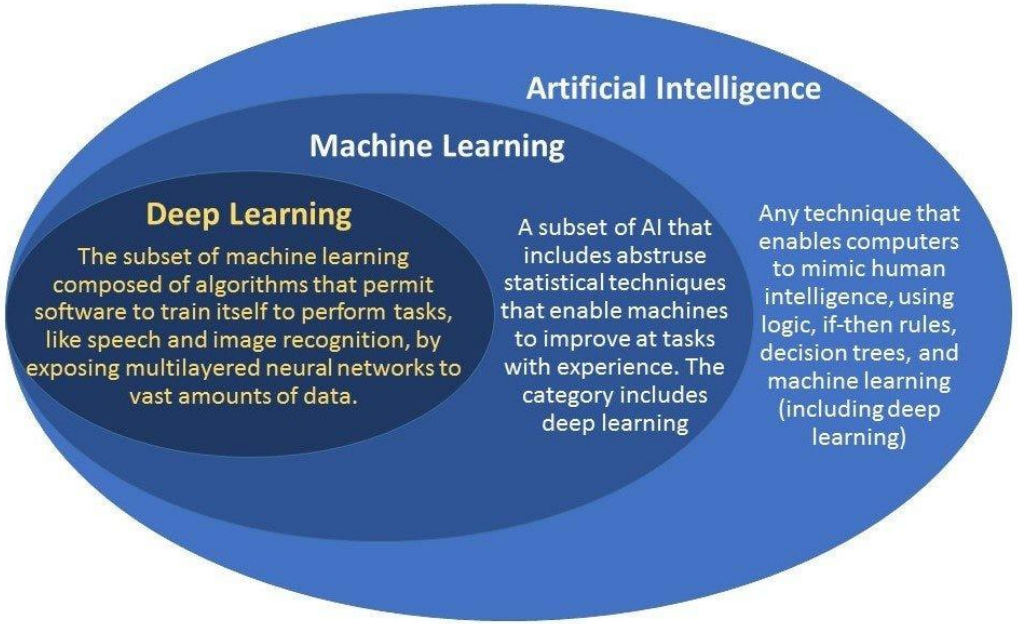
- Hinton’s work: Backpropagation, AlexNet and “Attention is All you need” like transformers.

Overview



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

AI, ML and DL relation

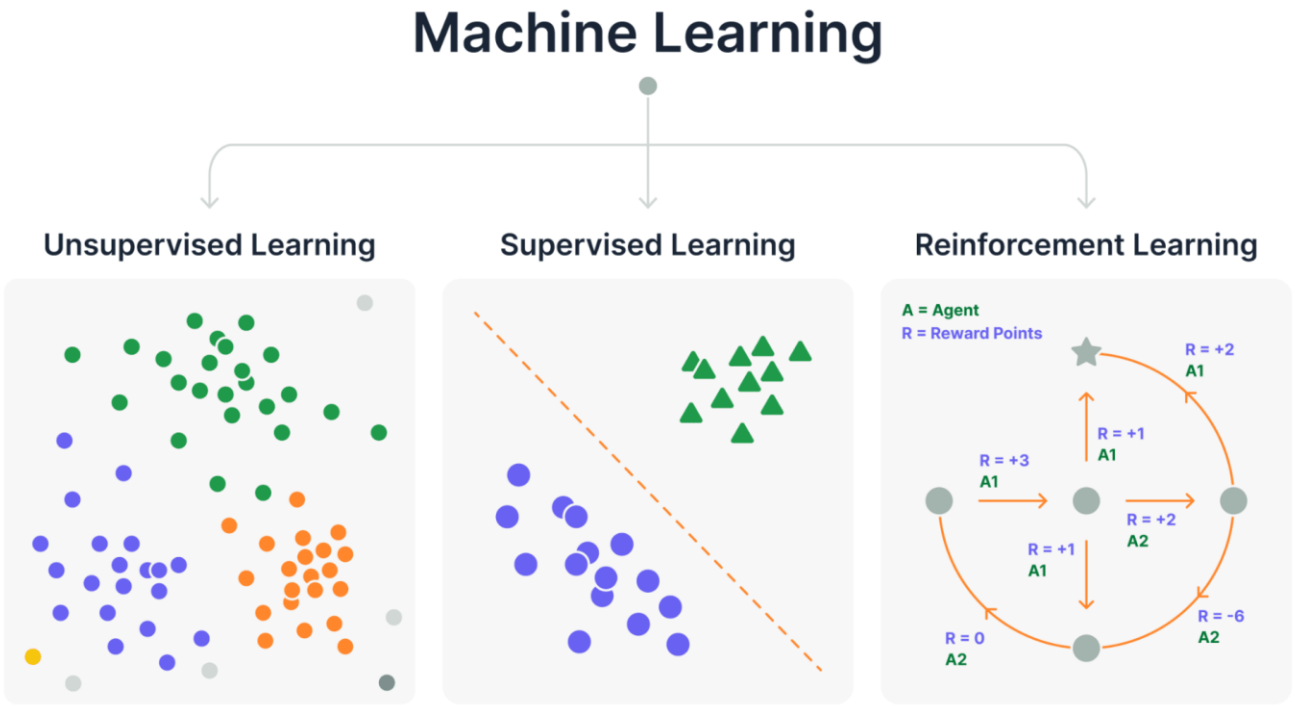


AI: Broad goal of mimicking human intelligence.

ML: Achieving AI through data-driven learning.

DL: Achieving ML using neural networks with multiple layers.

Overview


Machine Learning
Unsupervised Learning
Supervised Learning
Reinforcement Learning
A = Agent
R = Reward Points
R = +2
A1
R = +1
A1
R = +3
A1
R = +2
A2
R = +1
A1
R = -6
A2
R = 0
A2

What part of ML we going through

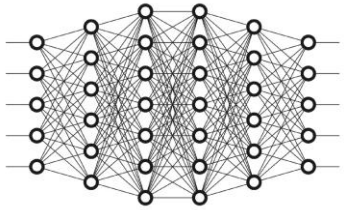
- Using machine learning (ML) models (mainly deep learning and Gen AI) in:

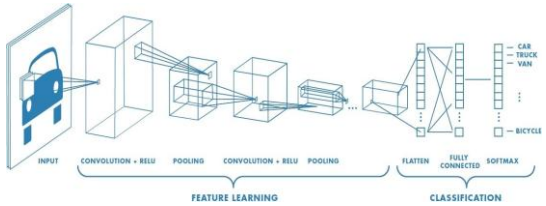
What part of ML we not going through

Deep Neural Network
(Multi layer perceptron)

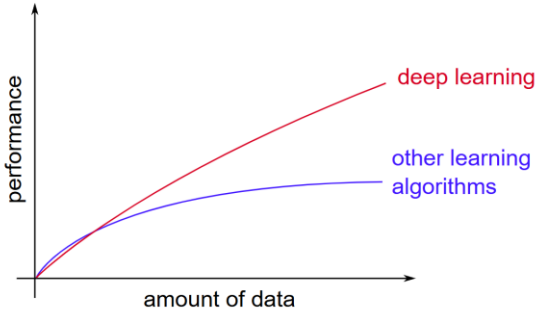
Why Deep Learning Now?

1. Better algorithm and understanding





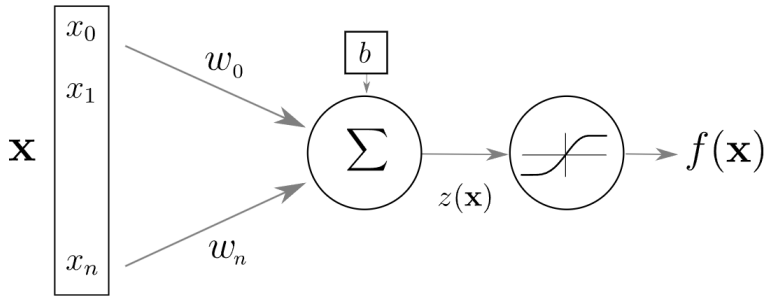
3. Data with labels



2. Computing power (GPU and TPU)

4. Open source tools and models

Simplest DNN

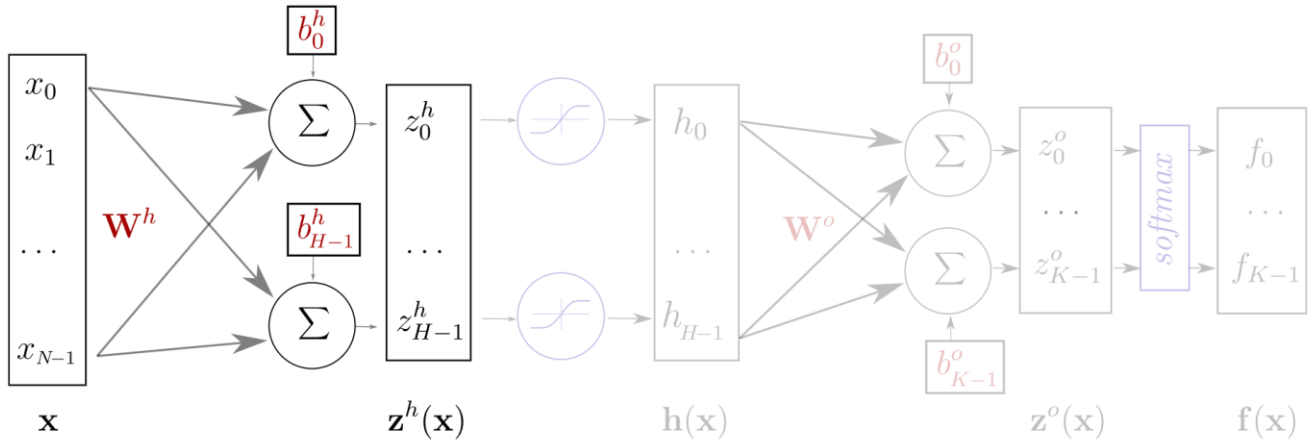


$$z(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b$$

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

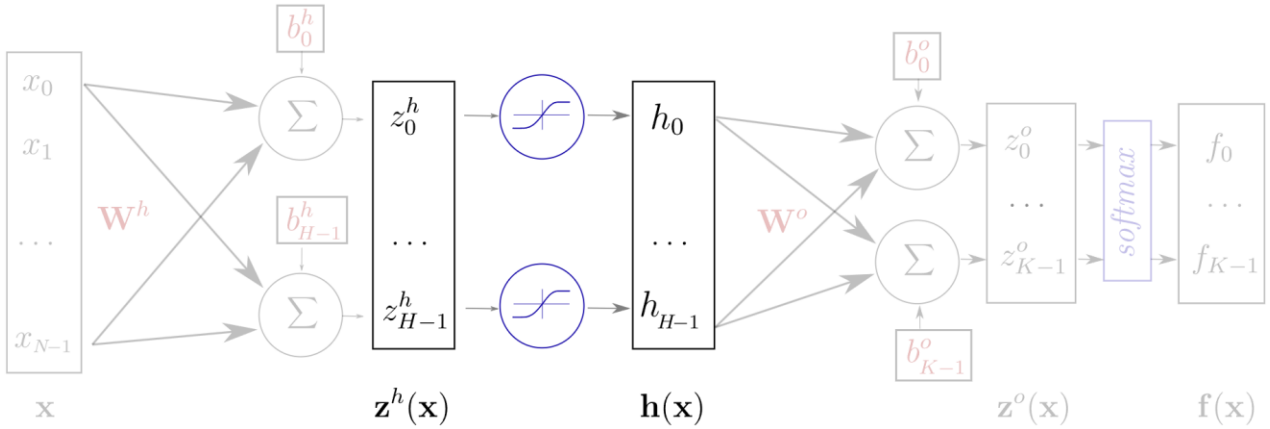
- $\mathbf{x}, f(\mathbf{x})$ input and output
- $z(\mathbf{x})$ pre-activation
- $\mathbf{w}, b$ weights and bias
- g activation function

Overview



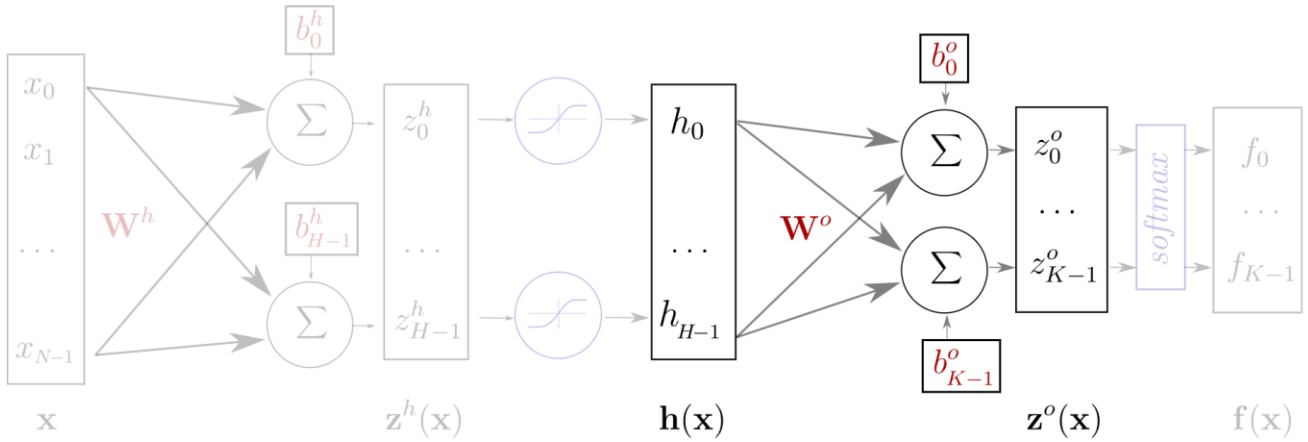
- $\mathbf{z}^h(\mathbf{x}) = \mathbf{W}^h\mathbf{x} + \mathbf{b}^h$
- $\mathbf{h}(\mathbf{x}) = g(\mathbf{z}^h(\mathbf{x})) = g(\mathbf{W}^h\mathbf{x} + \mathbf{b}^h)$
- $\mathbf{z}^o(\mathbf{x}) = \mathbf{W}^o\mathbf{h}(\mathbf{x}) + \mathbf{b}^o$
- $\mathbf{f}(\mathbf{x}) = softmax(\mathbf{z}^o) =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

Overview



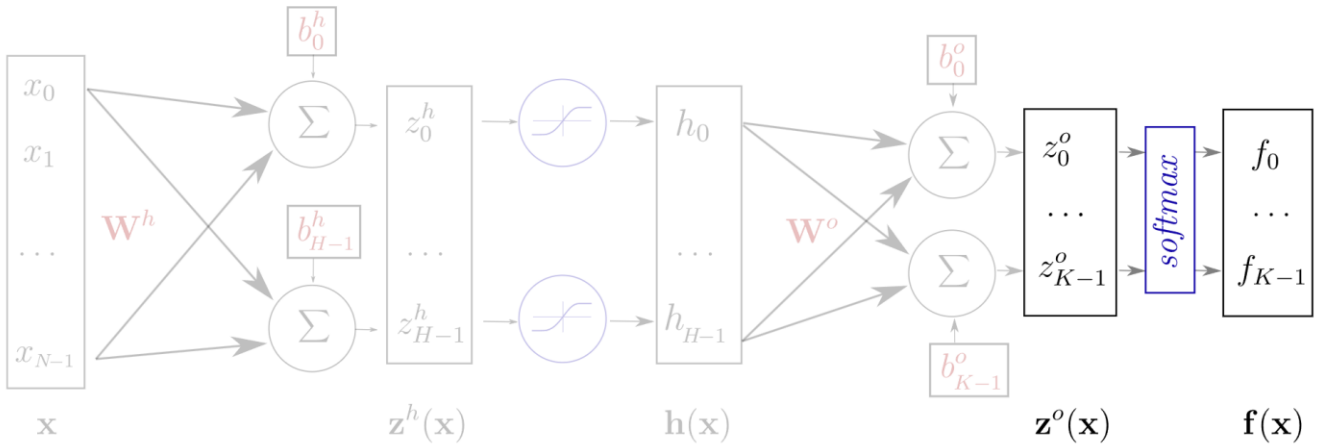
- $\mathbf{z}^h(\mathbf{x}) = \mathbf{W}^h\mathbf{x} + \mathbf{b}^h$
- $\mathbf{h}(\mathbf{x}) = g(\mathbf{z}^h(\mathbf{x})) = g(\mathbf{W}^h\mathbf{x} + \mathbf{b}^h)$
- $\mathbf{z}^o(\mathbf{x}) = \mathbf{W}^o\mathbf{h}(\mathbf{x}) + \mathbf{b}^o$
- $\mathbf{f}(\mathbf{x}) = softmax(\mathbf{z}^o) =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

Overview



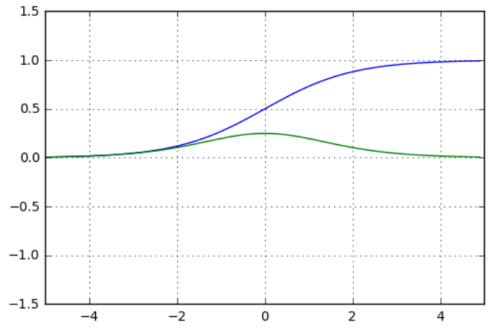
- $\mathbf{z}^h(\mathbf{x}) = \mathbf{W}^h\mathbf{x} + \mathbf{b}^h$
- $\mathbf{h}(\mathbf{x}) = g(\mathbf{z}^h(\mathbf{x})) = g(\mathbf{W}^h\mathbf{x} + \mathbf{b}^h)$
- $\mathbf{z}^o(\mathbf{x}) = \mathbf{W}^o\mathbf{h}(\mathbf{x}) + \mathbf{b}^o$
- $\mathbf{f}(\mathbf{x}) = softmax(\mathbf{z}^o) =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

Overview



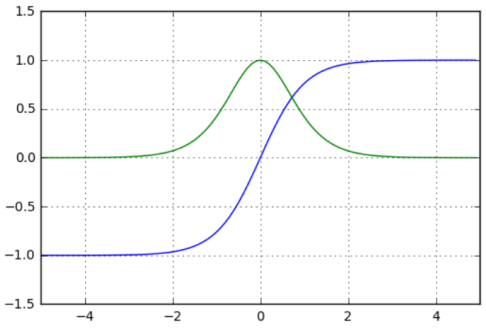
- $\mathbf{z}^h(\mathbf{x}) = \mathbf{W}^h\mathbf{x} + \mathbf{b}^h$
- $\mathbf{h}(\mathbf{x}) = g(\mathbf{z}^h(\mathbf{x})) = g(\mathbf{W}^h\mathbf{x} + \mathbf{b}^h)$
- $\mathbf{z}^o(\mathbf{x}) = \mathbf{W}^o\mathbf{h}(\mathbf{x}) + \mathbf{b}^o$
- $\mathbf{f}(\mathbf{x}) = softmax(\mathbf{z}^o) =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

What about ML not included



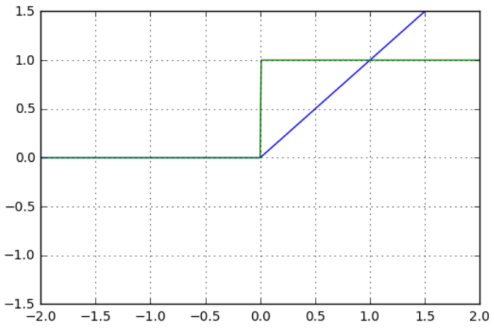
$$\text{sigm}(x) = \frac{1}{1+e^{-x}}$$

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



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

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



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

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

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

What about ML not included

Find parameters $\theta = (\mathbf{W}^h; \mathbf{b}^h; \mathbf{W}^o; \mathbf{b}^o)$ that minimize the **negative log likelihood** (or cross entropy)

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

example $y^s = 3$

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

What about ML not included

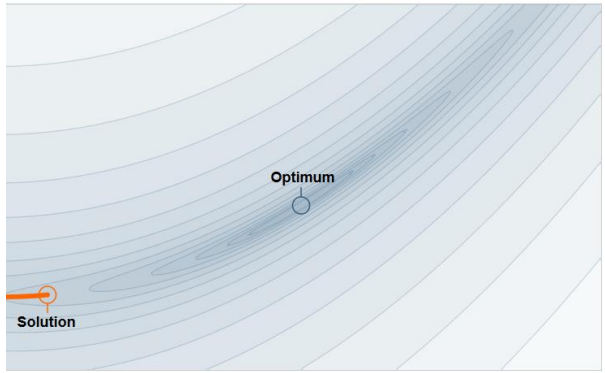
Initialize θ randomly

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)

What about ML not included

- Very sensitive:
  - Too high $\rightarrow$ early plateau or even divergence
  - Too low $\rightarrow$ slow convergence
  - Try a large value first: $\eta = 0.1$ or even $\eta = 1$
  - Divide by 10 and retry in case of divergence
- Large constant LR prevents final convergence
  - multiply η_t by $\beta < 1$ after each update
  - or monitor validation loss and divide η_t by 2 or 10 when no progress
  - See [ReduceLROnPlateau] in Keras



We often think of Momentum as a means of dampening oscillations and speeding up the iterations, leading to faster convergence. But it has other interesting behavior. It allows a larger range of step-sizes to be used, and creates its own oscillations. What is going on?

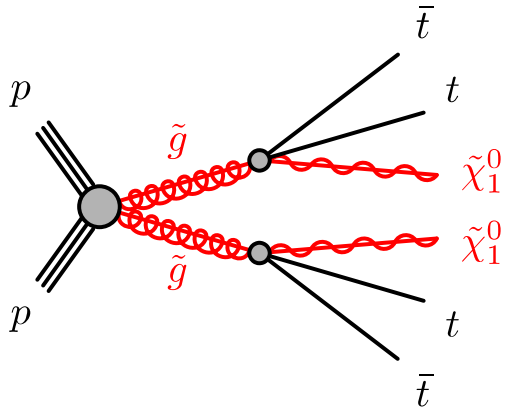
What about ML not included

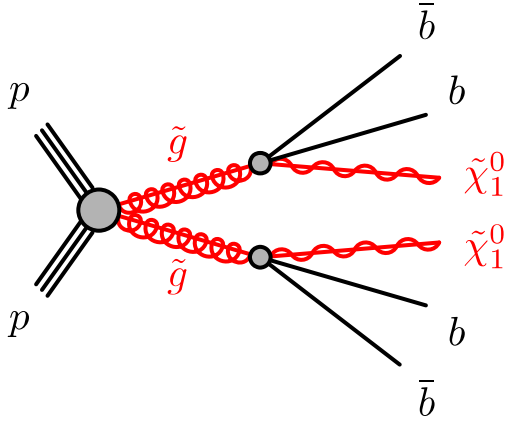
metrics

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)~~

- Potentially high cross-section for gluino pair production
- Target final states with large amount of E_T^{miss} and several *b*-jets
- Main background is **semi-leptonic $t\bar{t}$**

Definition of Key Variables

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

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

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) :

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

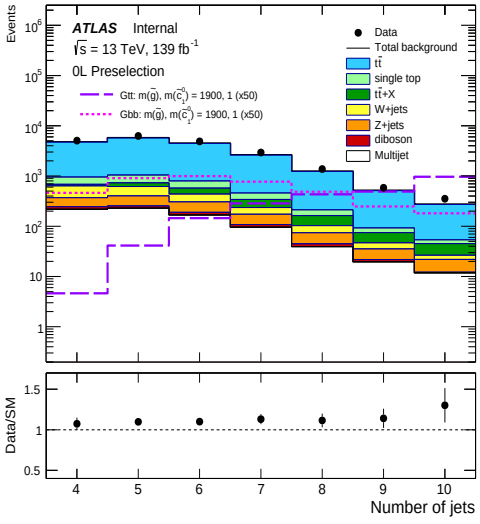
$m_{T,min}^{b-jets}$ min transverse mass between E_T^{miss} and three leading b-jets:

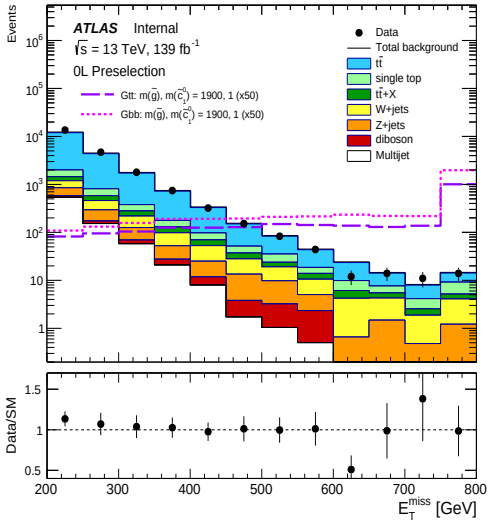
$$m_{T,min}^{b-jets} = \min_{i \le 3}\left(\sqrt{2 p_T^{b-jet_i} E_T^{miss} \left(1 - \cos\Delta\phi\left(\vec{p}_T^{miss}, \vec{p}_T^{b-jet_i}\right)\right)}\right)$$

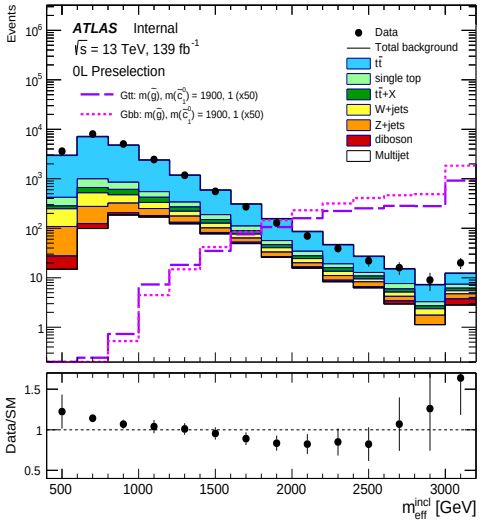
$M_J^{\Sigma,4}$ sum of the mass of re-clustered jets (higher for Gtt signal):

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

Preselection Distributions





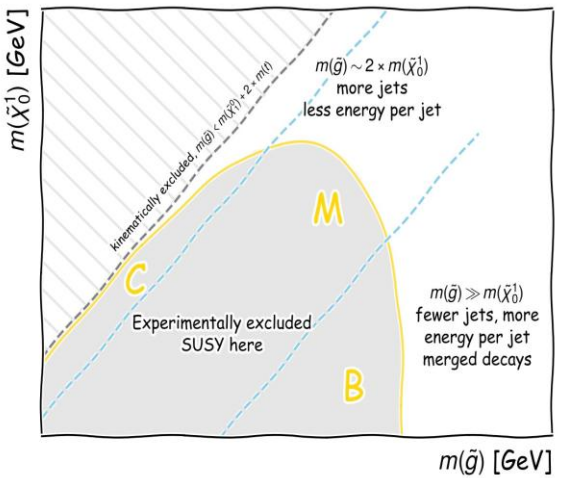


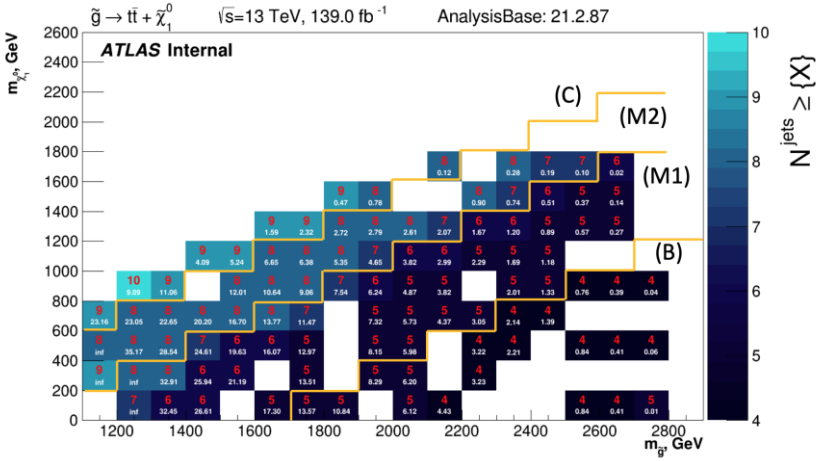
$m_{T,min}^{b-jets}$ min transverse mass between E_T^{miss} and three leading b-jets:

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

Cut-and-count

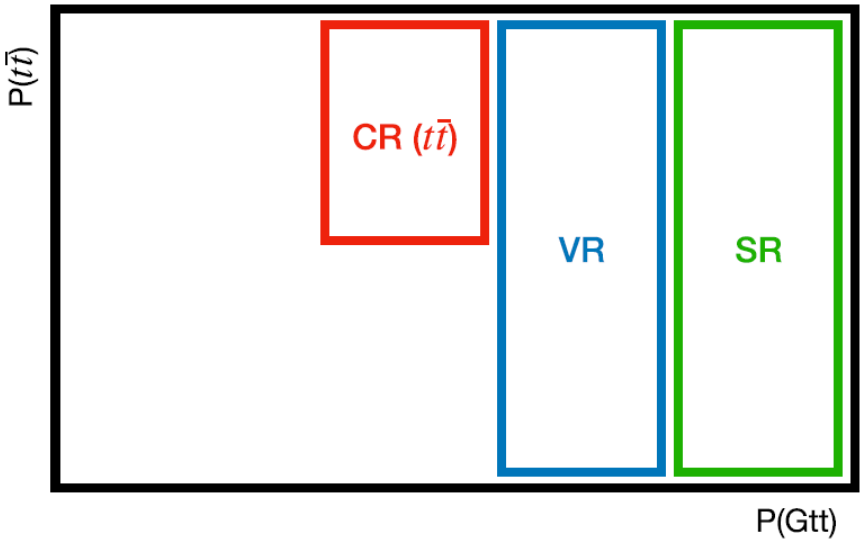
- Define **signal regions** (SRs) targeting different regio parametrized by the mass splitting between the gluino anc
  - optimized to maximize the significance of a discove
- Non-overlapping **control regions** (CRs):
  - low signal contamination
  - high ttbar purity
  - used to derive scale factors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





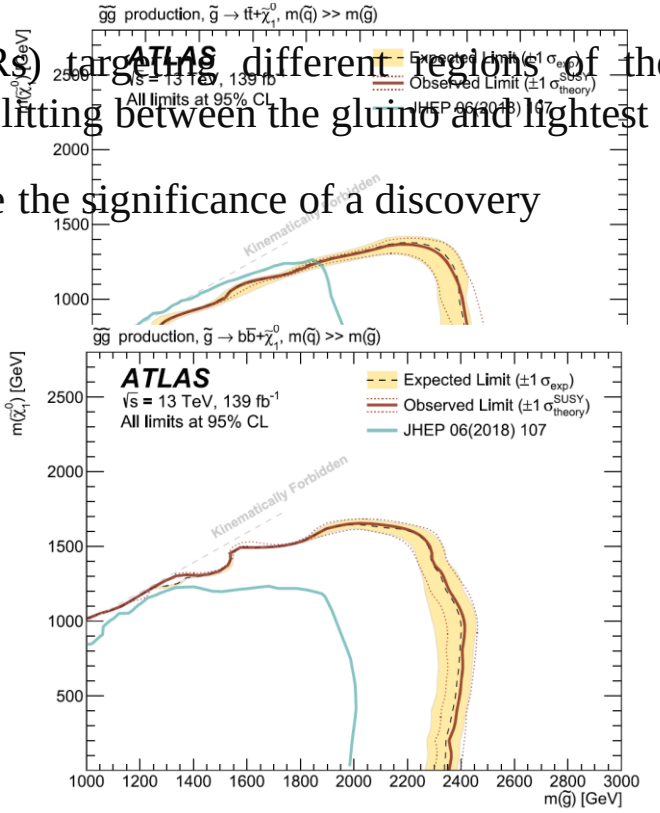
What about ML not included

- 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



What about ML not included

- 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



Convolutional Neural Network

What about ML not included

[Krizhevsky 2012]

[Ciresan et al. 2013]

[Faster R-CNN - Ren 2015]

[NVIDIA dev blog]

What about ML not included

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)
```

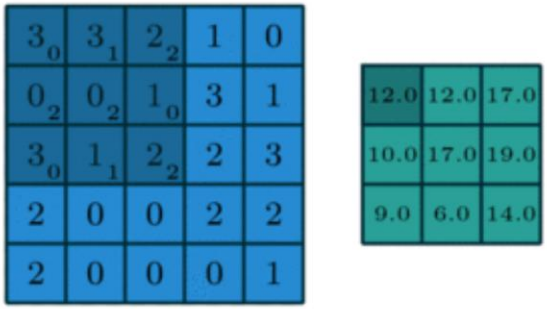
How many parameters in the Dense layer?

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

Spatial organization of the input is destroyed by `Flatten`

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

What about ML not included

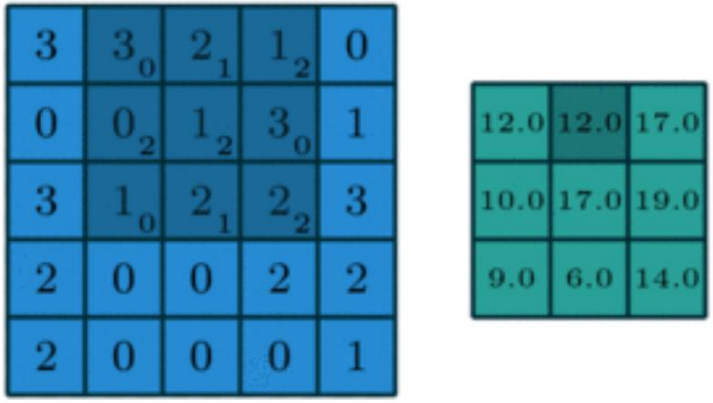


- x is a 3×3 chunk (dark area) of the image *(blue array)*
- Each output neuron is parametrized with the 3×3 weight matrix $\mathbf{w}$ *(small numbers)*

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

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

What about ML not included

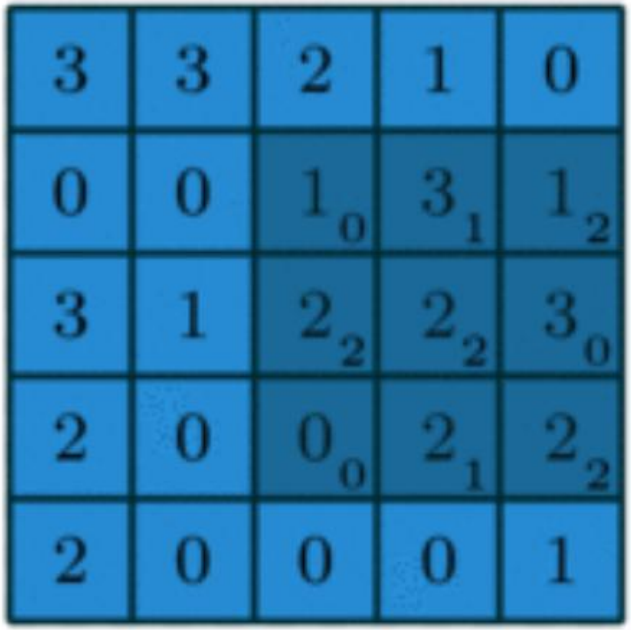


2D-convolutions (actually 2D cross-correlation):

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

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

What about ML not included

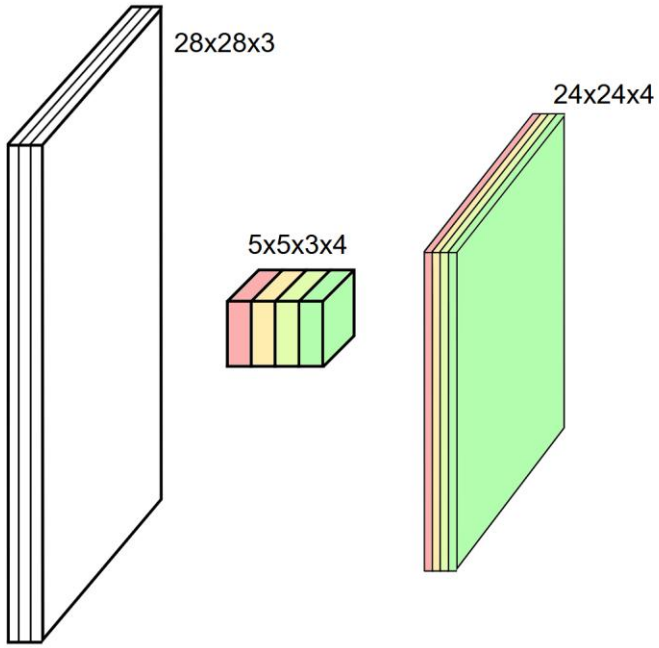


What about ML not included

Comparison to Fully connected

- Parameter sharing: reduce overfitting
- Make use of spatial structure: **strong prior** for vision!

What about ML not included

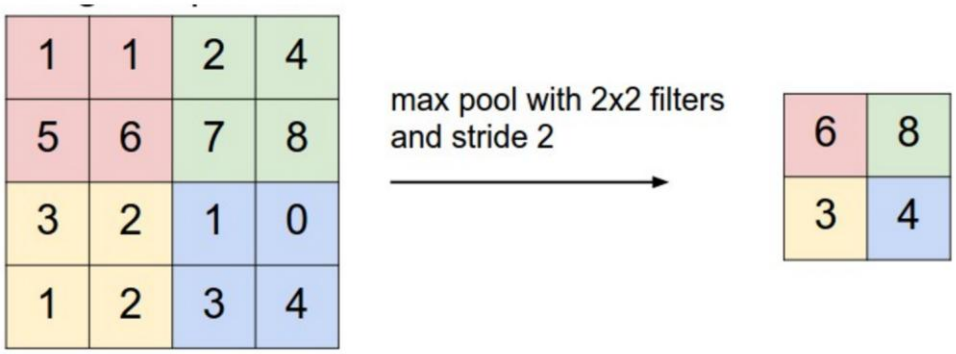


- Kernel size aka receptive field (usually 1, 3, 5, 7, 11)
- Output dimension: `length - kernel_size + 1`

What about ML not included

Pooling

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



What about ML not included

Classic ConvNet Architecture

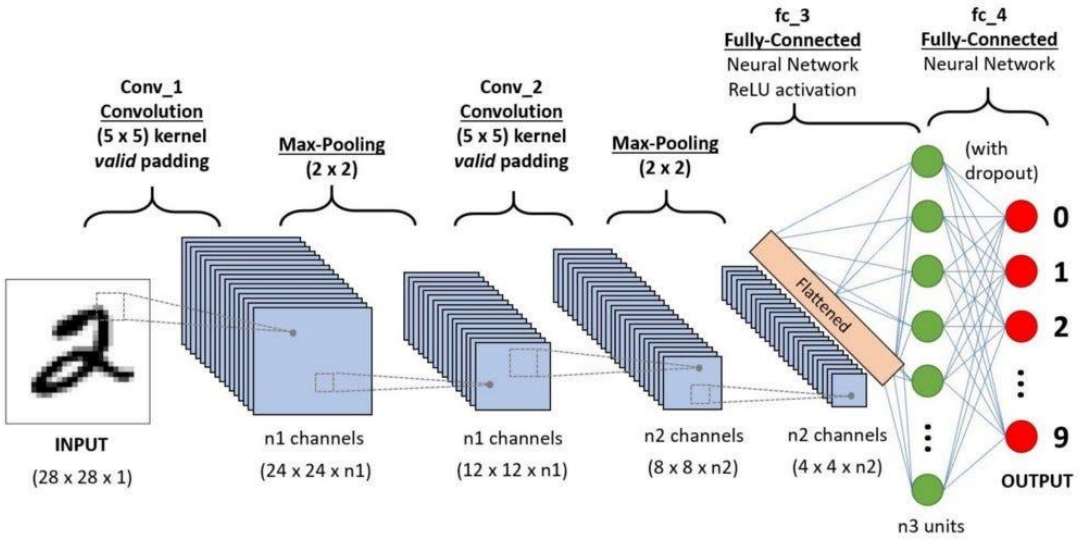
Input

Conv blocks

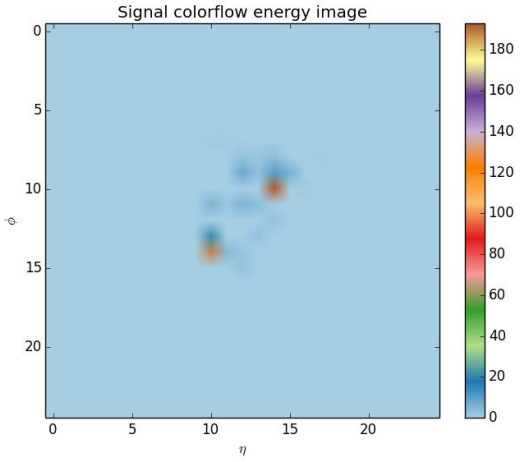
- Convolution + activation (relu)
- Convolution + activation (relu)
- ...
- Maxpooling 2x2

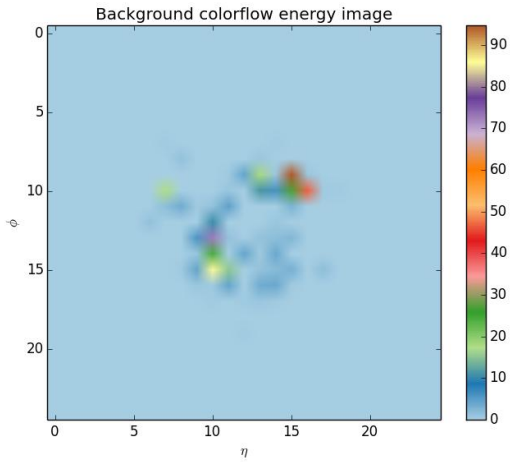
Output

- Fully connected layers
- Softmax

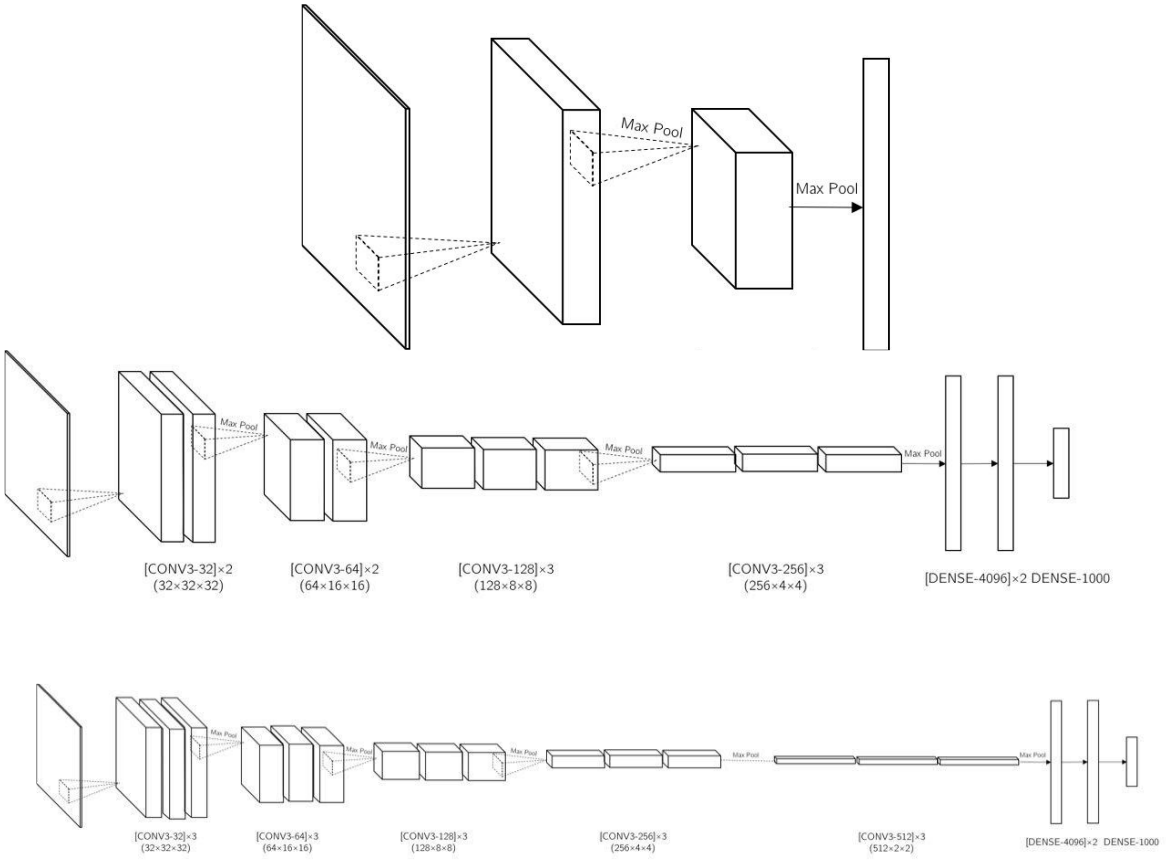


What about ML not included

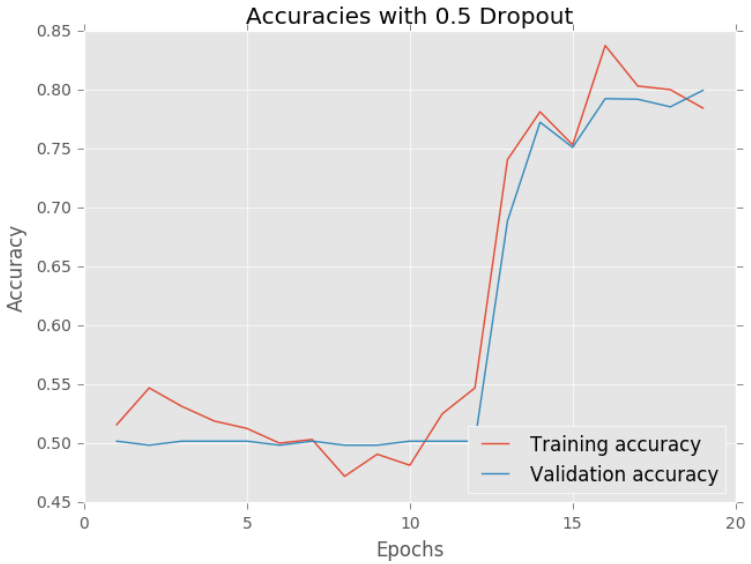


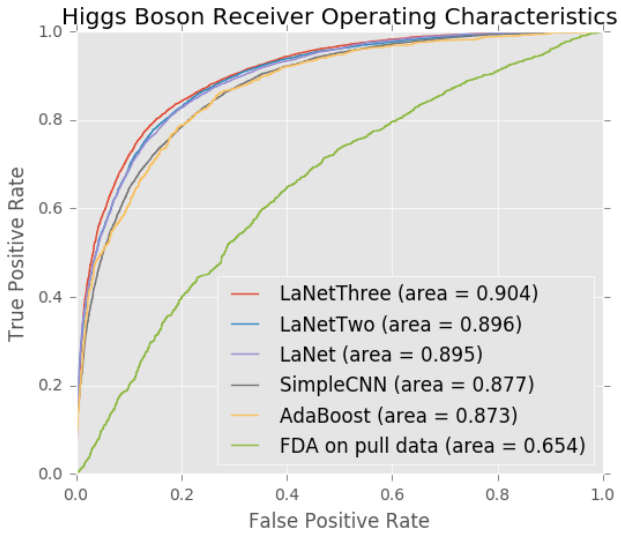


What about ML not included



What about ML not included

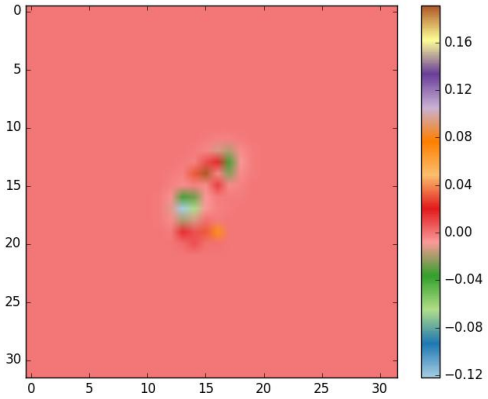




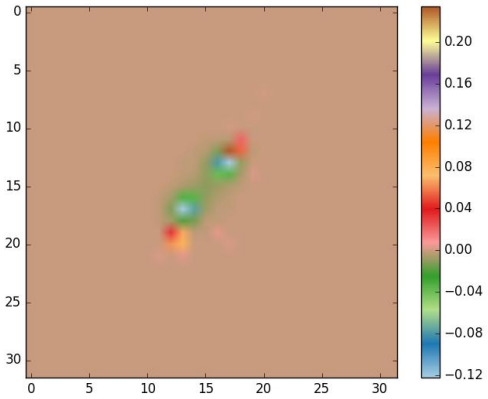
Model	AUC	Accuracy
FDA	0.654	-
AdaBoost	0.873	0.798
SimpleModel	0.877	0.774
LaNet	0.895	0.812
LaNetTwo	0.896	**0.825**
LaNetThree	**0.904**	0.820

What about ML not included

metrics



background



Higgs

Quantum Machine Learning

What about ML not included

Introduction

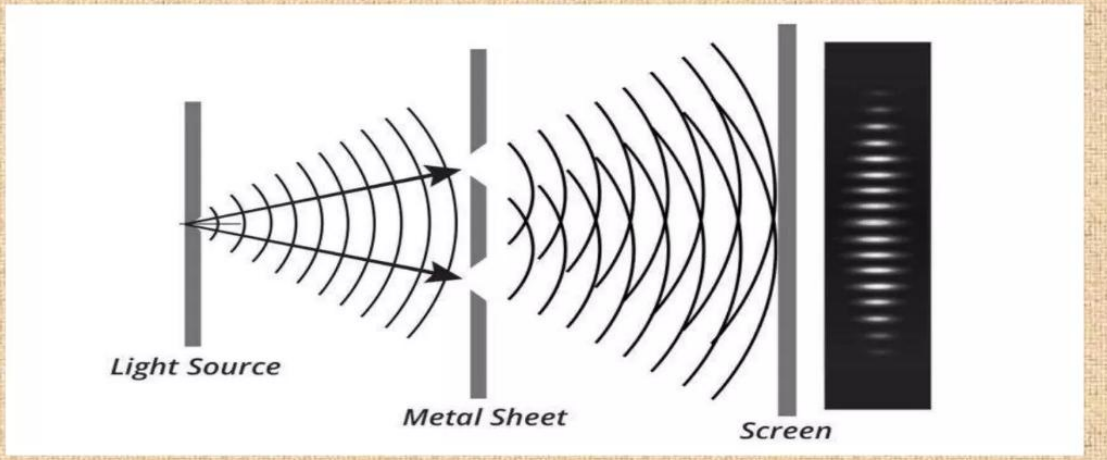
- Quantum computing is an area of computer science that uses the principles of quantum theory. Quantum theory explains the behavior of energy and material on the atomic and subatomic levels.
- Quantum computing uses subatomic particles, such as electrons or photons. Quantum bits, or qubits, allow these particles to exist in more than one state (i.e., 1 and 0) at the same time.
- A qubit can be in state $\frac{1}{\sqrt{2}}|0\rangle$ and $\frac{1}{\sqrt{2}}|1\rangle$ or (unlike a classical bit) in a linear combination of both states. The name of this phenomenon is superposition.

What about ML not included

Graphical Representation of QUBIT

Quantum Superposition

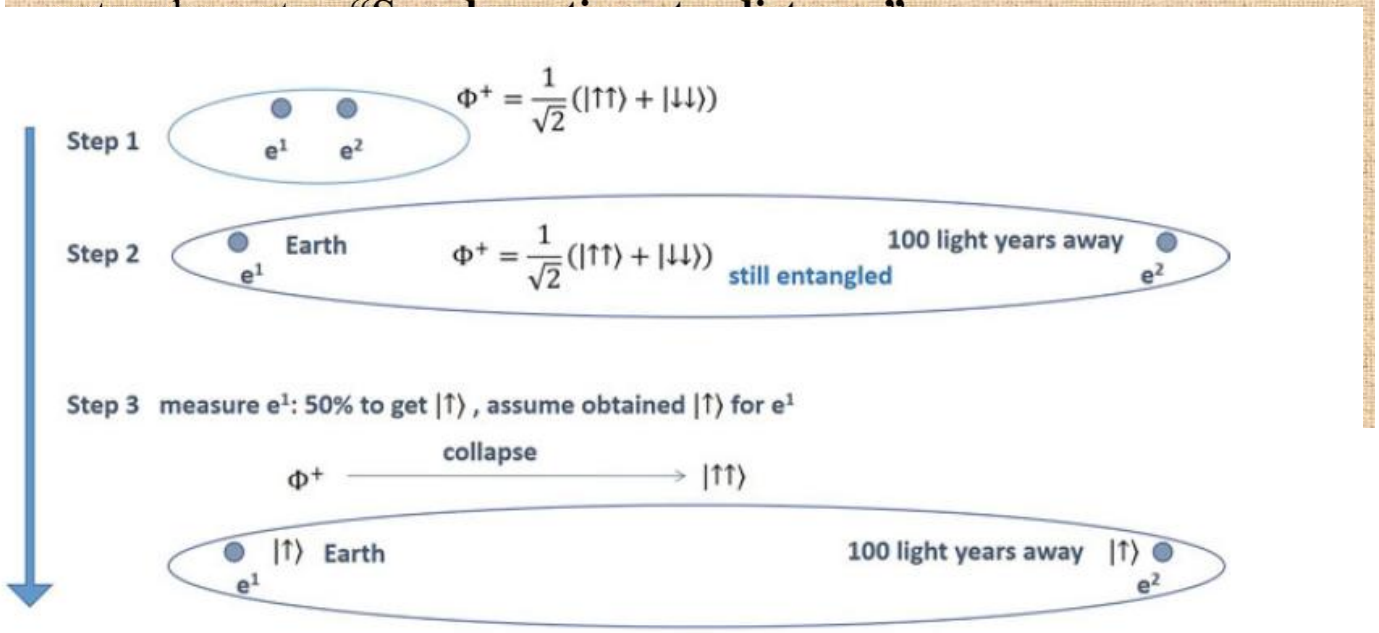
- The quantum system is capable of being in several different states at the same time.
- Example – Young's Double Slit Experiment



What about ML not included

Quantum Entanglement

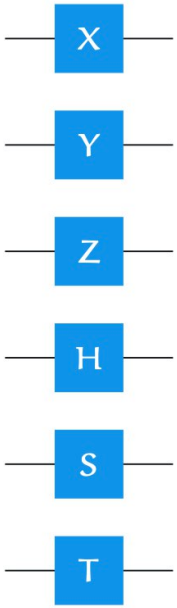
- It is an extremely strong correlation that exists between quantum particles
- Two or more quantum particles can be inextricably linked in perfect unison, even when placed at opposite ends of the universe.
- This seemingly impossible connection inspired Einstein to describe



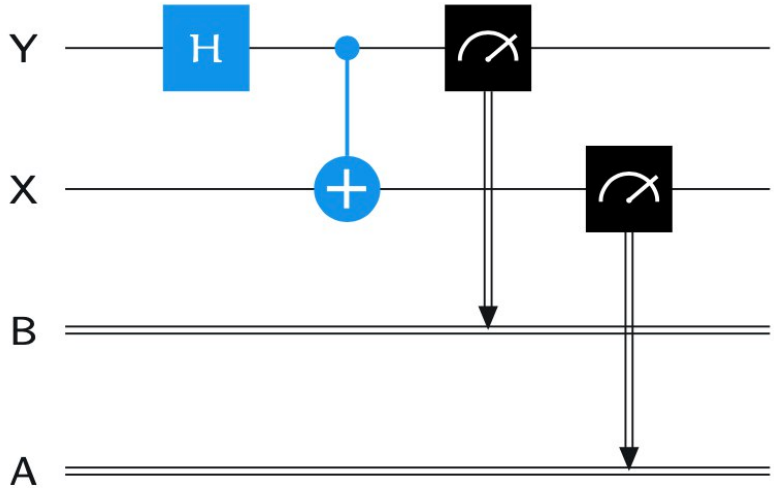
What about ML not included

Example

Single-qubit gates



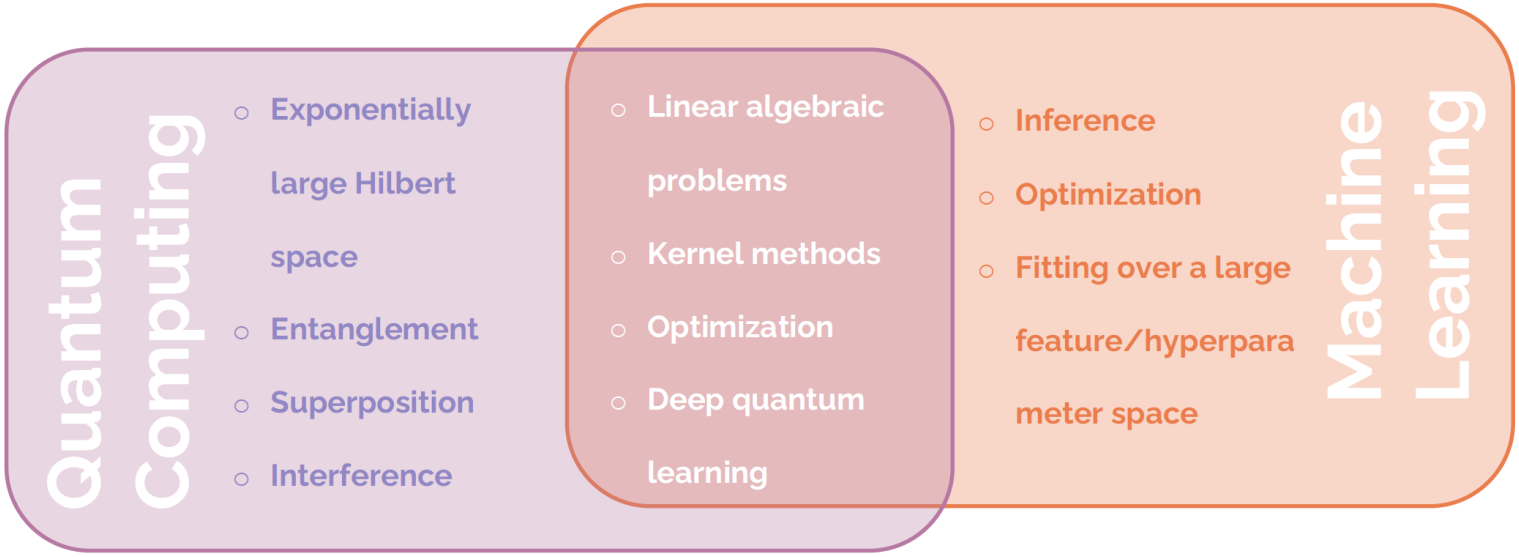
Example



What about ML not included

Quantum Machine Learning

The main goal of Quantum Machine Learning (QML) is to speed things up by applying what we know from quantum computing to machine learning

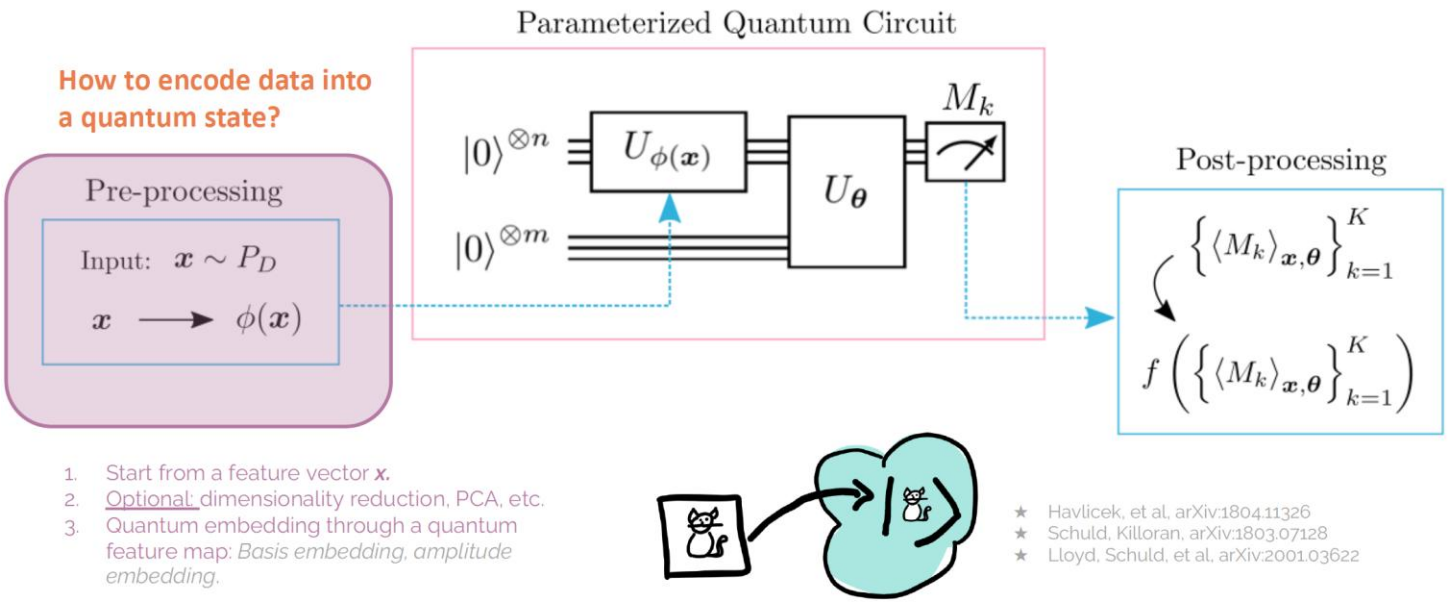


QCs can naturally solve certain problems with complex relations between inputs that can be incredibly hard for traditional, or "classical", computers. This suggest that learning models made on QC may be dramatically powerful for select applications, potentially boasting faster computation, better generalization on less data, or both.

What about ML not included

Parameterized Quantum Circuits as ML Models

Benedetti, arXiv:1906.07682



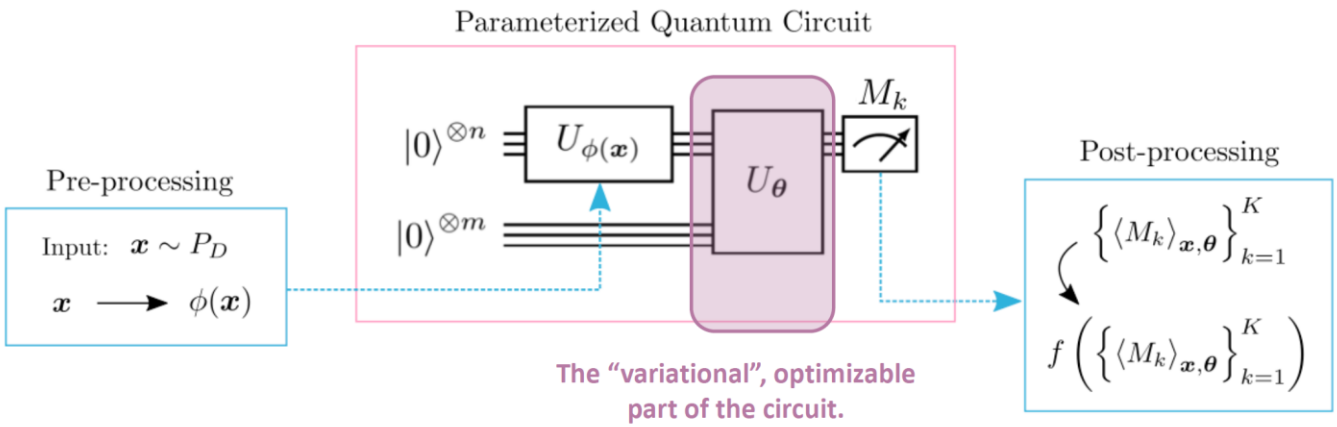
1. Start from a feature vector ***x***.
2. Optional: dimensionality reduction, PCA, etc.
3. Quantum embedding through a quantum feature map: *Basis embedding, amplitude embedding.*

★ Havlicek, et al, arXiv:1804.11326
★ Schuld, Killoran, arXiv:1803.07128
★ Lloyd, Schuld, et al, arXiv:2001.03622

What about ML not included

Parameterized Quantum Circuits as ML Models

Benedetti, arXiv:1906.07682

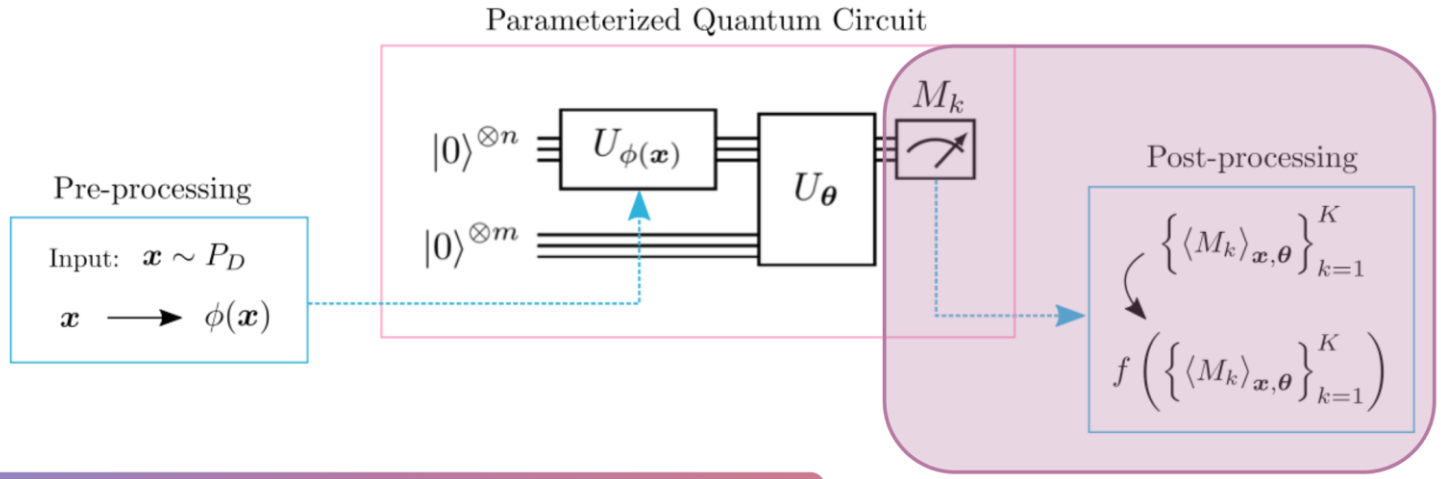


The "guess" or trial function is the unitary U parameterized by a set of free parameters θ that will be updated during training.

What about ML not included

Parameterized Quantum Circuits as ML Models

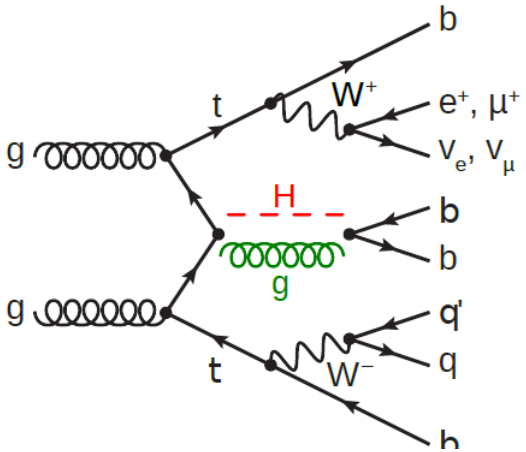
Benedetti, arXiv:1906.07682



Quantum information is turned back into classical information by evaluating the expectation value of an observable, or measurement.

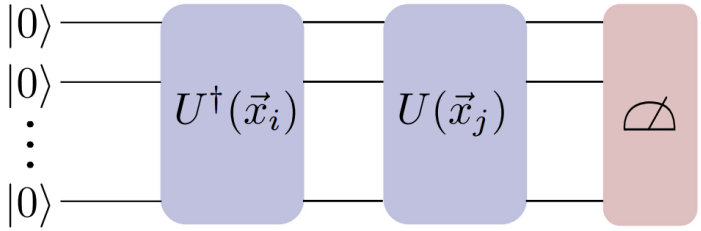
The measurement output is then used to construct a decision function, a probability distribution, a boundary, etc.

What about ML not included



In the pre-selection step, we require the electrons and muons to pass selection criteria of the transverse momentum p_T, pseudorapidity η and isolation with respect to jets. Namely, the object selection cuts are: $p_T > 30$ GeV, $|\eta| < 2.1$ and iso > 0.1 for the electrons, $p_T > 26$ GeV, $|\eta| < 2.1$ and iso > 0.1 for the muons and $p_T > 30$ GeV, $|\eta| < 2.4$ for the jets. After this initial object selection, we further consider events with at least one lepton and at least 4 jets from which at least two are b-tagged. Thus, the following event selection criteria are used: $n^{\text{jet}} \geq 4$, $n^{\text{b-tag}} \geq 2$ and $n^{\text{leptons}} = 1$. From the leading-order description of the process depicted in Fig. 1 we identify that the nominal expectation consists of 4 b-tagged jets and 2 jets of any flavour, hence 6 jets in total. For our analysis, we keep the 7 most energetic jets of each event, allowing one extra jet to take into account initial or final state radiation.

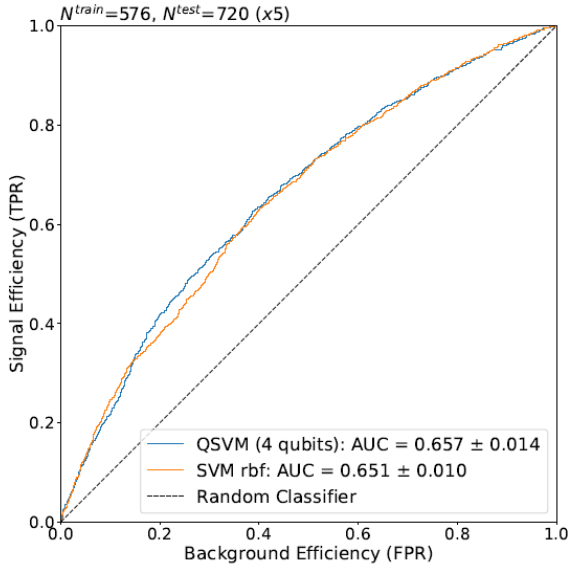
What about ML not included



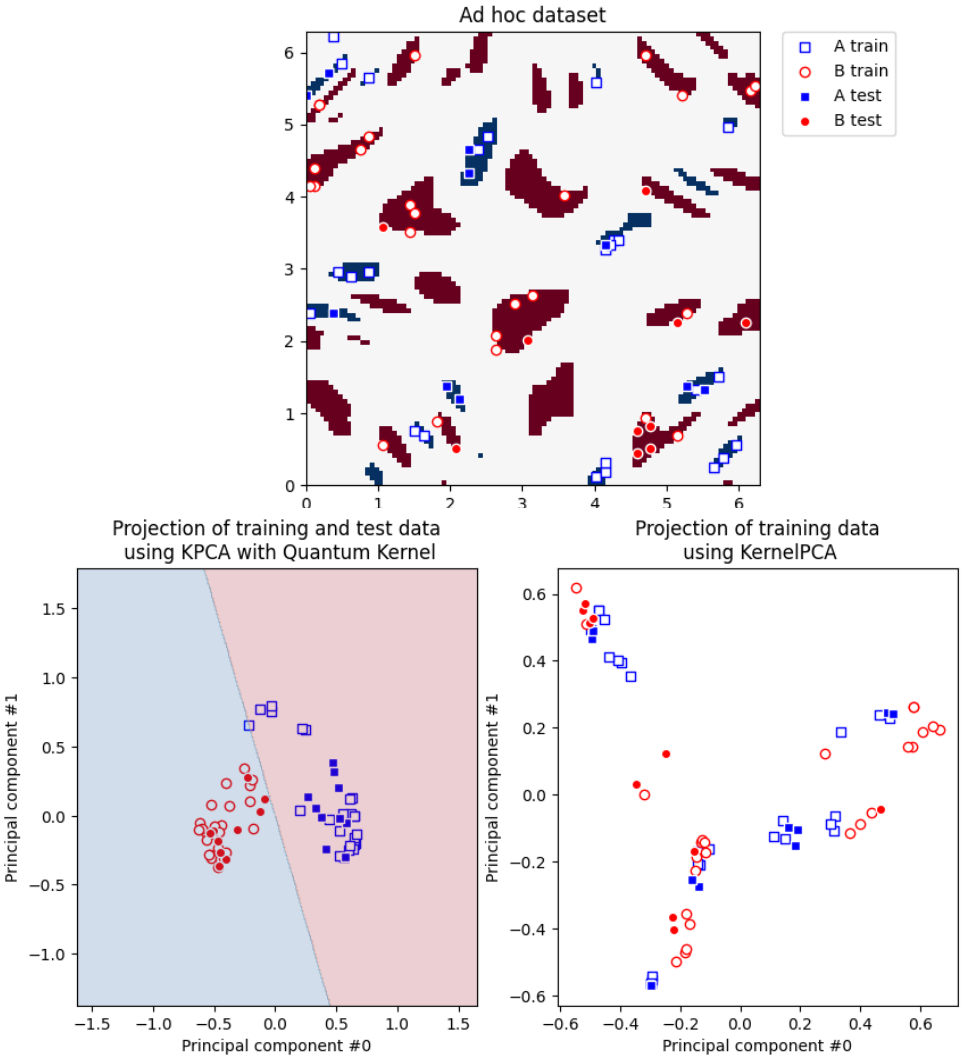
$$\Rightarrow K_{ij} = |\langle 0|U^\dagger(\vec{x}_i)U(\vec{x}_j)|0\rangle|^2$$

maximize $L(c_1 \ldots$

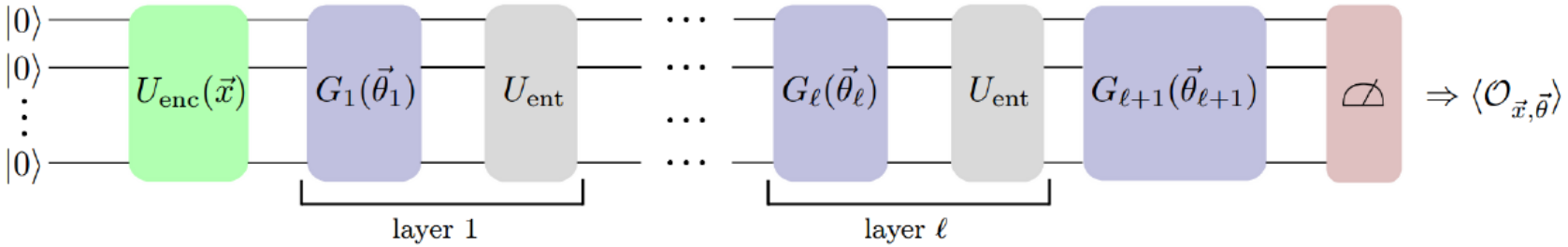
subject to $\sum_{i=1}^{n} c_i y_i$

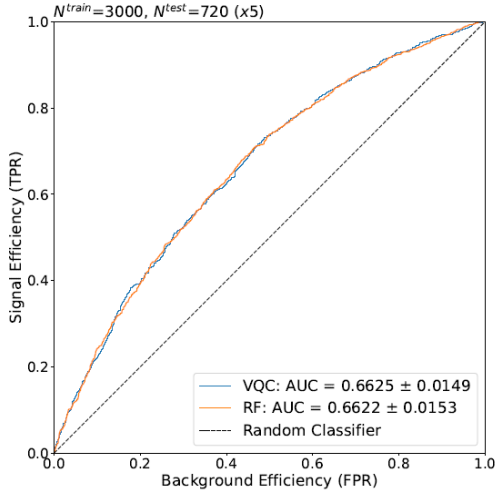


What about ML not included



What about ML not included





What about ML not included

What about ML not included

Generative AI

What about ML not included

🌍 The Problem: Data Bottleneck"There’s not enough human-made data in the world to train future AI systems."— Elon Musk"The next era of AI will be powered by AI-generated data."— Jensen Huang, NVIDIA CEO 🔁 The Solution: Generative AIGenAI can create synthetic data that mimics real-world distributions.Enables training at scale when labeled or diverse human data is limited.Powers simulations in physics, medicine, autonomous driving, and more.Critical for fine-tuning LLMs, robotics, and domain-specific applications. 🔗 Bridging the GapHuman-labeled data is costly, slow, and limited.GenAI offers scalable, controllable, and privacy-safe data.A new loop: Train AI → Use AI to generate better data → Train better AI.

Synthetic biomedical data: Patient data is sensitive and hard to share. GenAI can generate synthetic patient records or medical images (e.g., MRI scans) that preserve statistical properties without risking privacy.

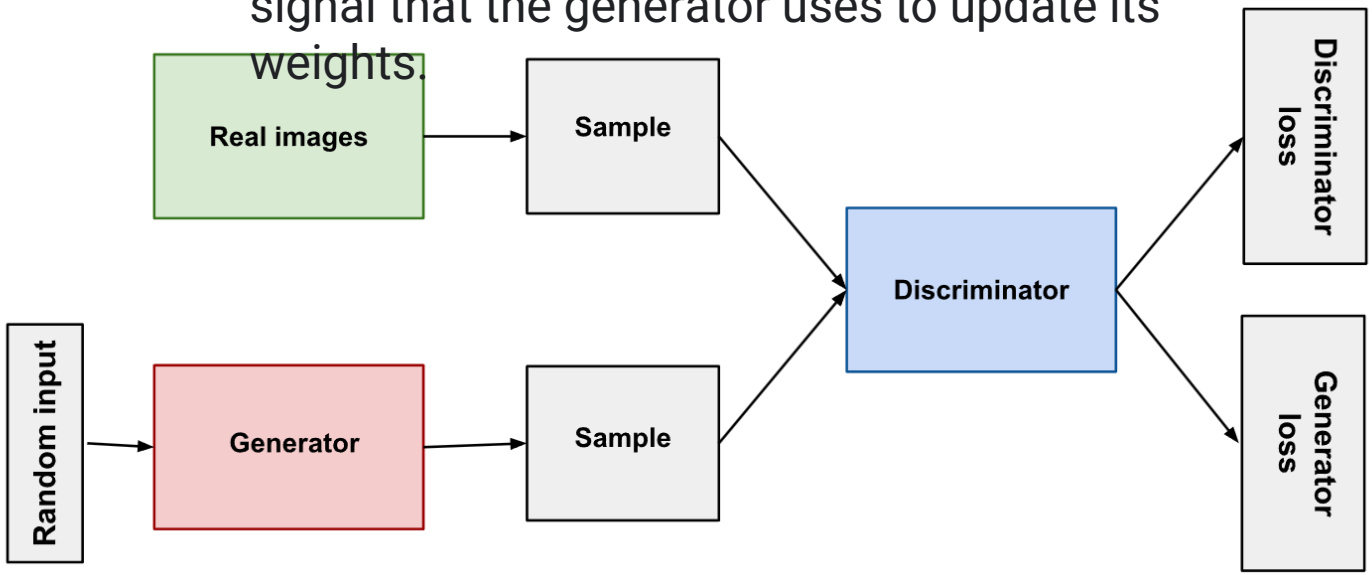
Simulation acceleration: Traditional physics simulations (e.g., in particle physics or cosmology) are **computationally expensive**. GenAI (like GANs or diffusion models) can generate **high-fidelity synthetic data** much faster.

Medical decision support: GenAI can simulate rare disease cases for training doctors or AI systems when **real cases are scarce**.

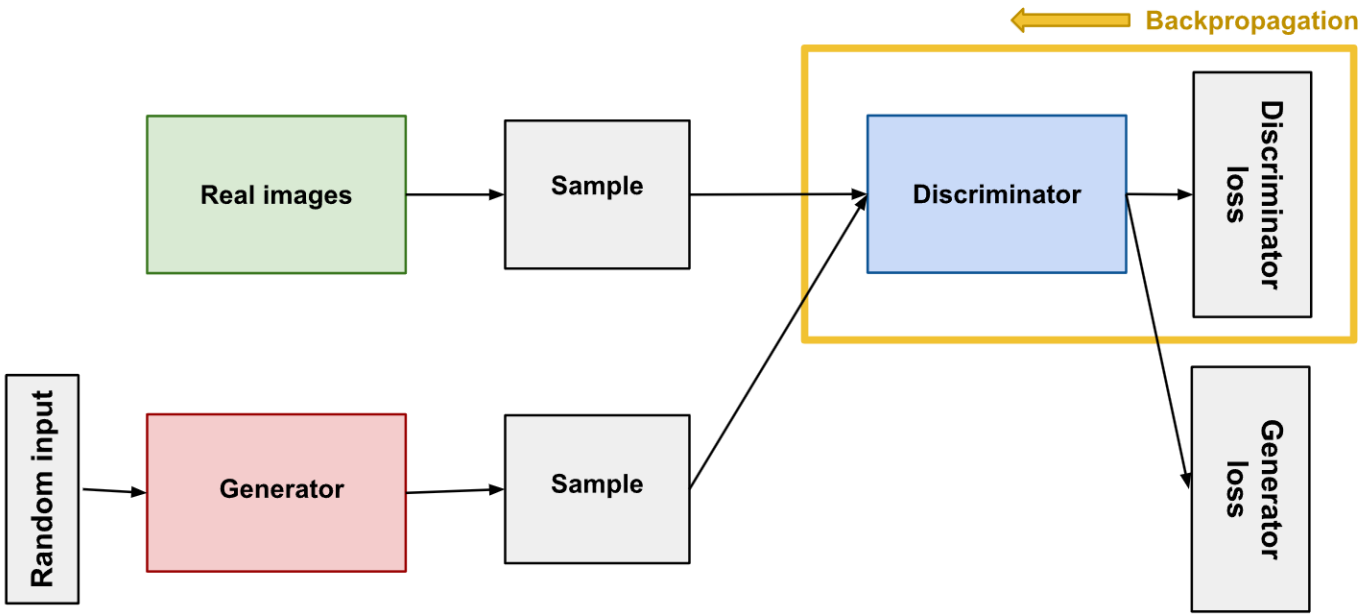
Edge case generation: Self-driving cars must handle rare and dangerous scenarios (e.g., kids running into the street). GenAI can create those **"what-if" simulations**.

What about ML not included

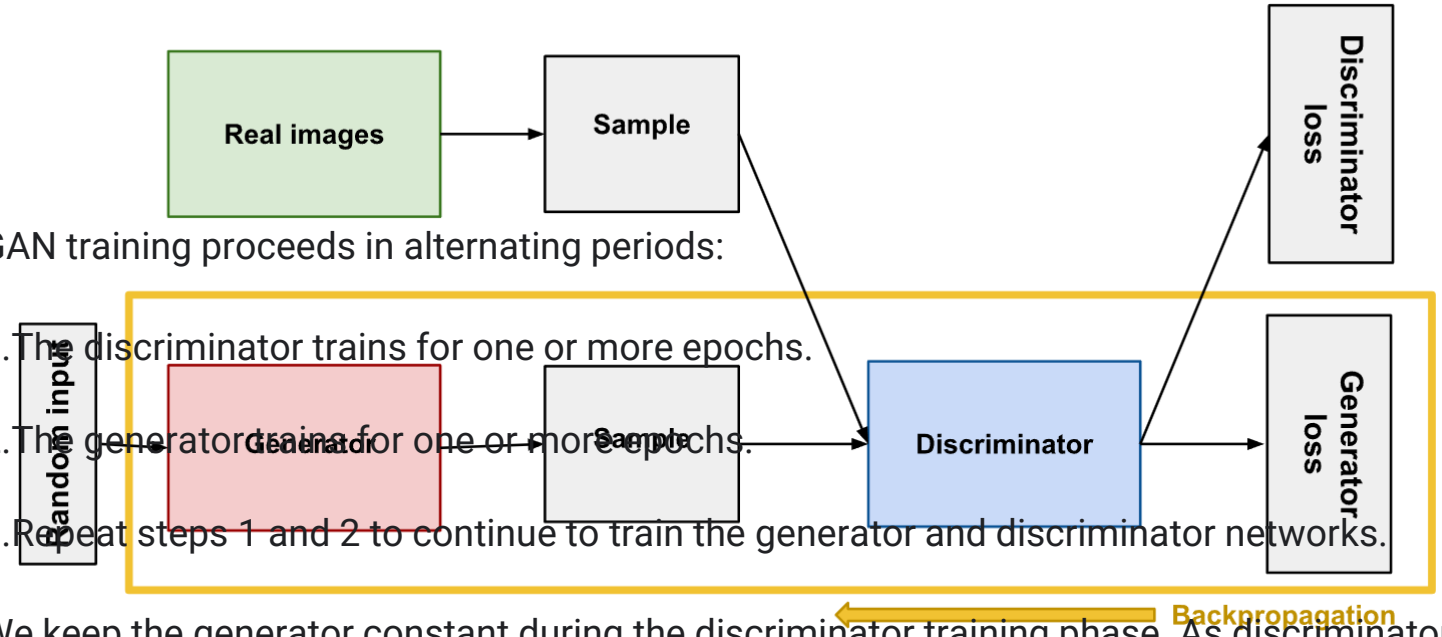
Both the generator and the discriminator are neural networks. The generator output is connected directly to the discriminator input. Through backpropagation, the discriminator's classification provides a signal that the generator uses to update its weights.



What about ML not included



What about ML not included



GAN training proceeds in alternating periods:

1. The discriminator trains for one or more epochs.
2. The generator trains for one or more epochs.
3. Repeat steps 1 and 2 to continue to train the generator and discriminator networks.

We keep the generator constant during the discriminator training phase. As discriminator training tries to figure out how to distinguish real data from fake, it has to learn how to recognize the generator's flaws. That's a different problem for a thoroughly trained generator than it is for an untrained generator that produces random output.

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.

What about ML not included

This progression poses a problem for convergence of the GAN as a whole: the discriminator feedback gets less meaningful over time. If the GAN continues training past the point when the discriminator is giving completely random feedback, then the generator starts to train on junk feedback, and its own quality may collapse.

For a GAN, convergence is often a fleeting, rather than stable, state.

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:

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

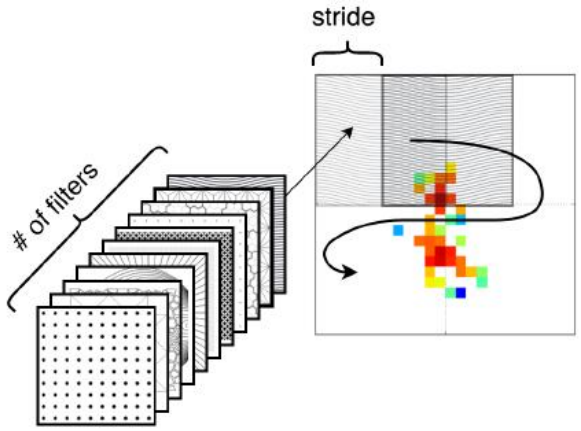
The GAN training objective is:

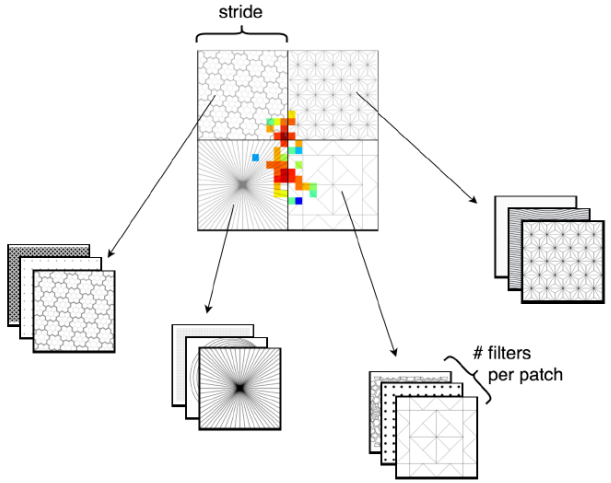
$$\min_G \max_D \mathcal{L}(D, G)$$

- **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 $\mathcal{L}$.

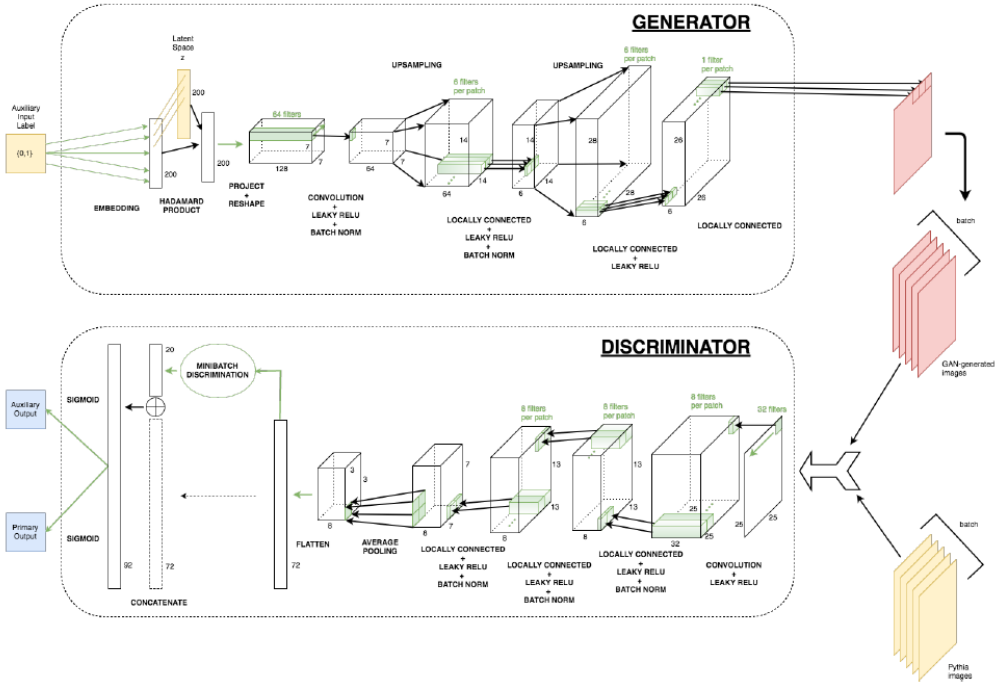
What about ML not included

What about ML not included



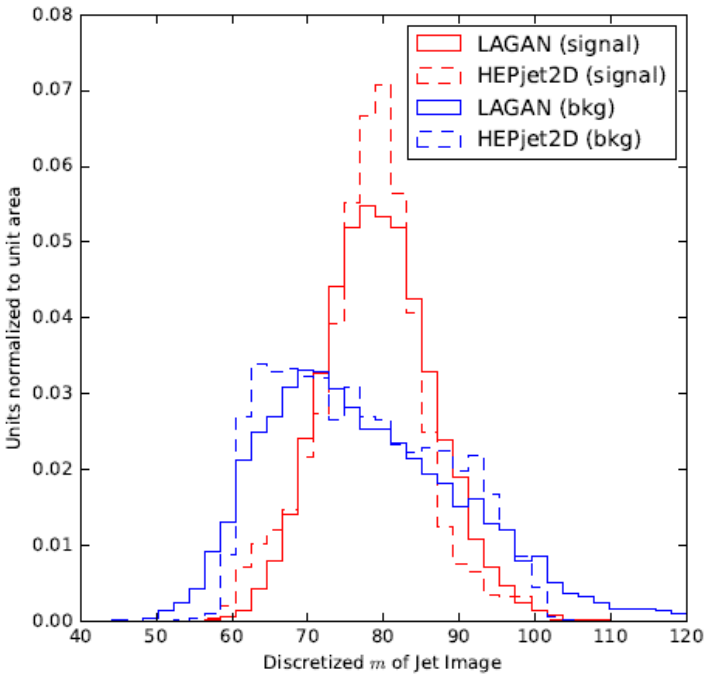


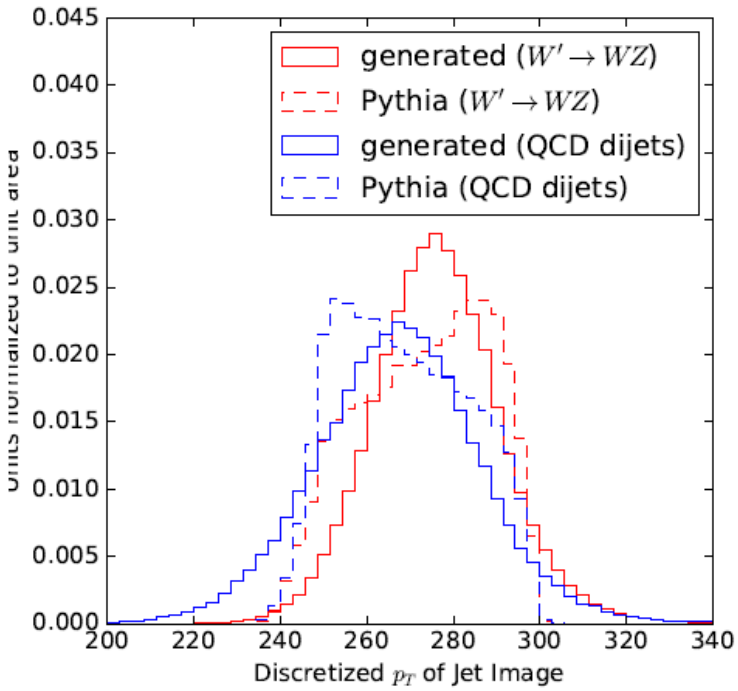
What about ML not included



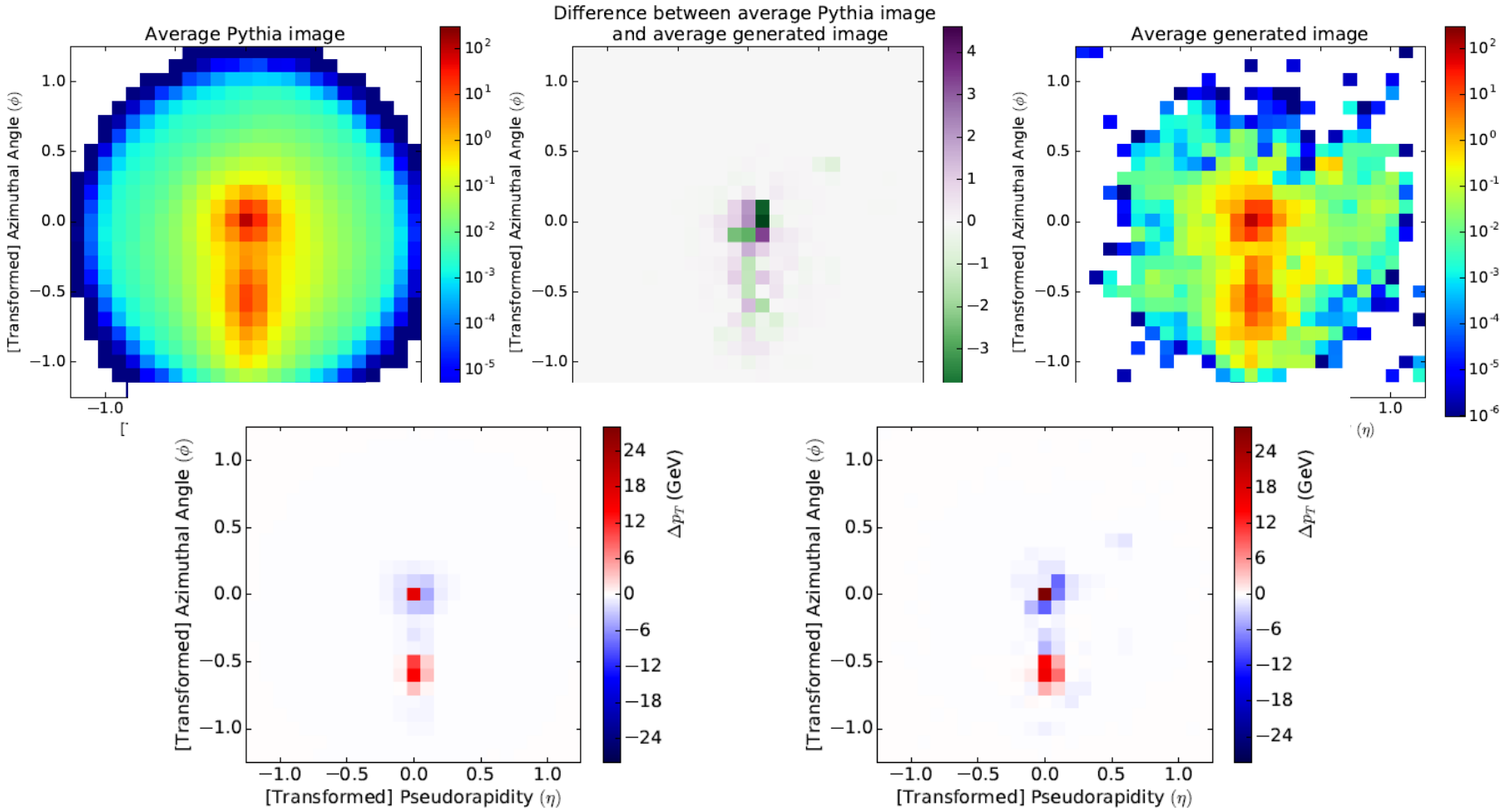
- **Locally Connected Layers** - or any attentional component where we can attend to location specific features - to be used in the generator and the discriminator
- **Rectified Linear Units** in the last layer to induce sparsity
- **Batch normalization**, as also recommended in [7], to help with weight initialization and gradient stability
- **Minibatch discrimination**[4], which experimentally was found to be crucial in modeling both the high dynamic range and the high levels of sparsity

What about ML not included





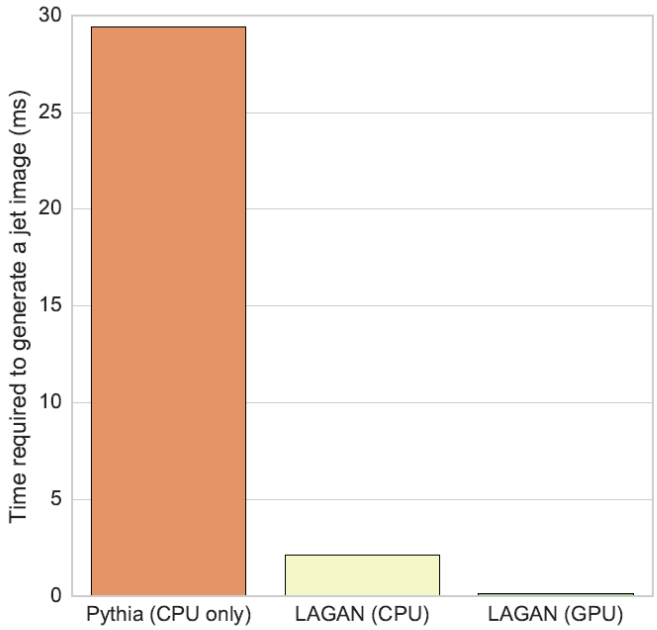
What about ML not included



What about ML not included

Table 1: Performance Comparison for LAGAN and Pythia-driven event generation

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



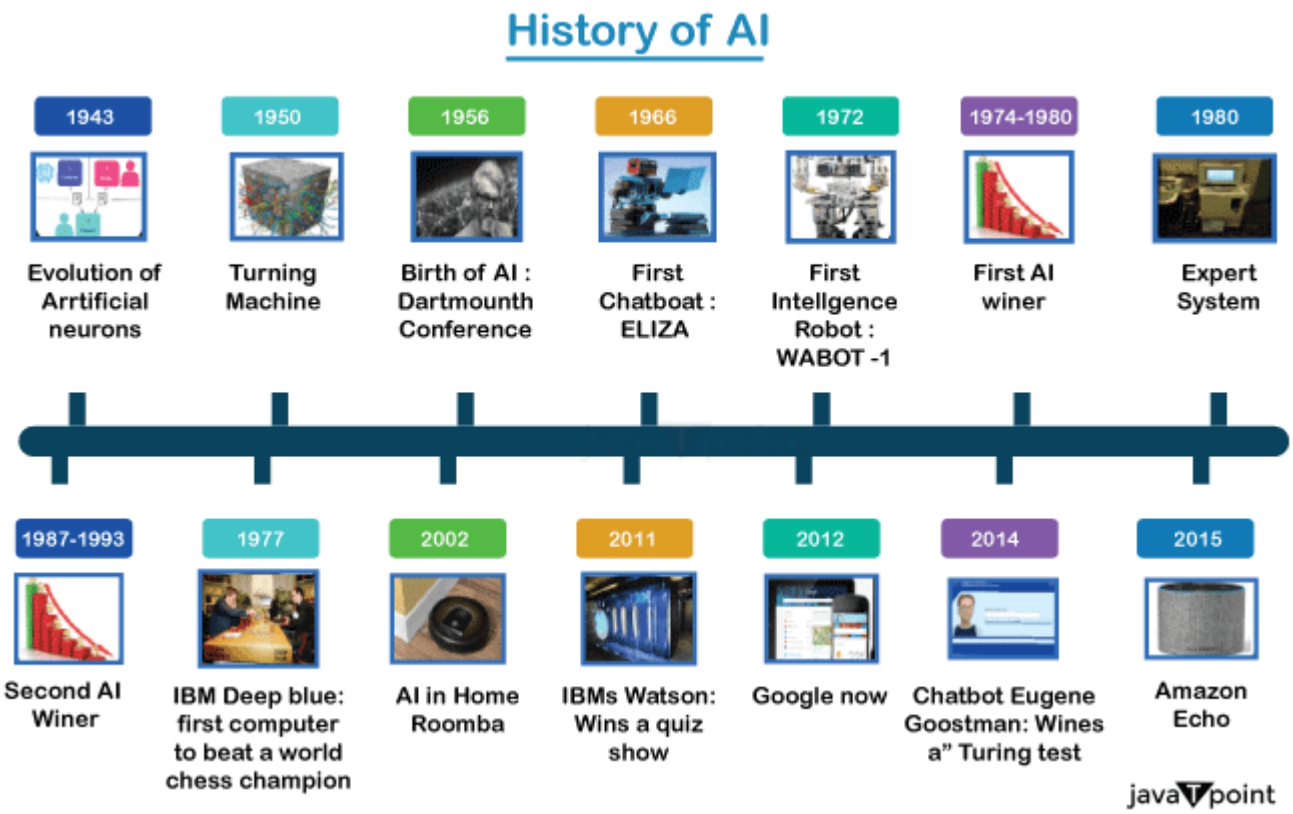
What about ML not included

Summary & ongoing

- The paper on top FCNC is finished and ready for submission.
- The paper on classical and quantum ML applications in jet discrimination is in progress (70% complete). The remaining tasks include running QSVM on a real IBM quantum server and training GNN with GraphSAGE.
- Your feedback is welcome and greatly appreciated.

Backup

What about ML not included



What about ML not included

📐 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.

What about ML not included

What about ML not included

What about ML not included

What about ML not included

What about ML not included

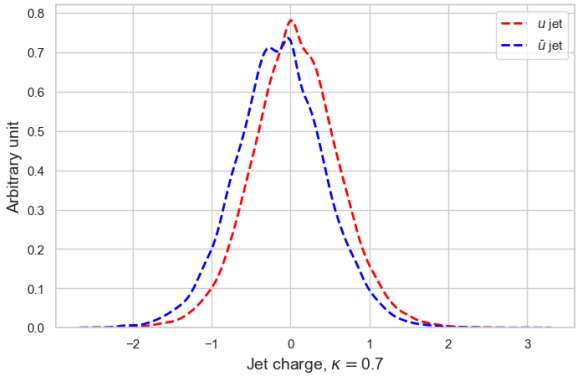
What about ML not included

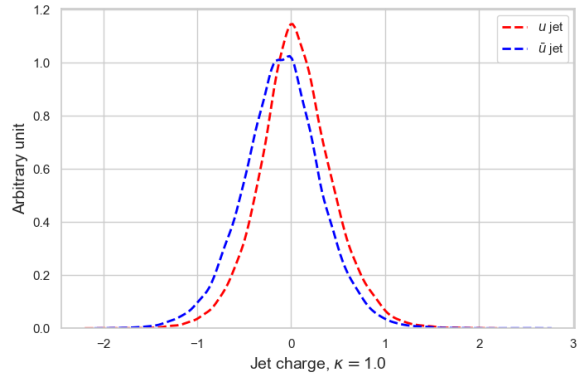
Application of Deep learning to Jet charge

- 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

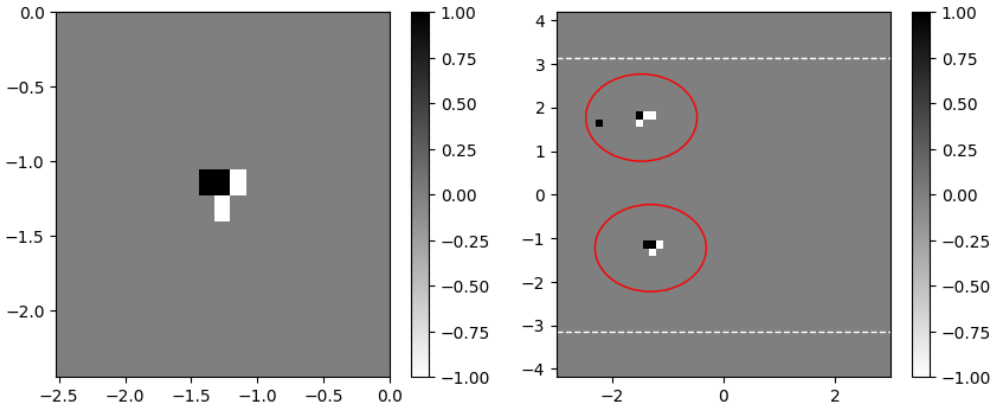
Convolutional NN and Graph NN

Convolutional Neural Networks (CNNs):

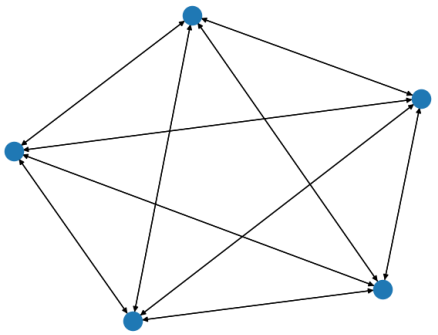
- CNNs process jet images by analyzing pixel charge distributions in **η-φ space**.
- They capture **spatial correlations** of energy deposits to classify jet charge.
- Can differentiate between quark-initiated and gluon-initiated jets using learned spatial features.

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

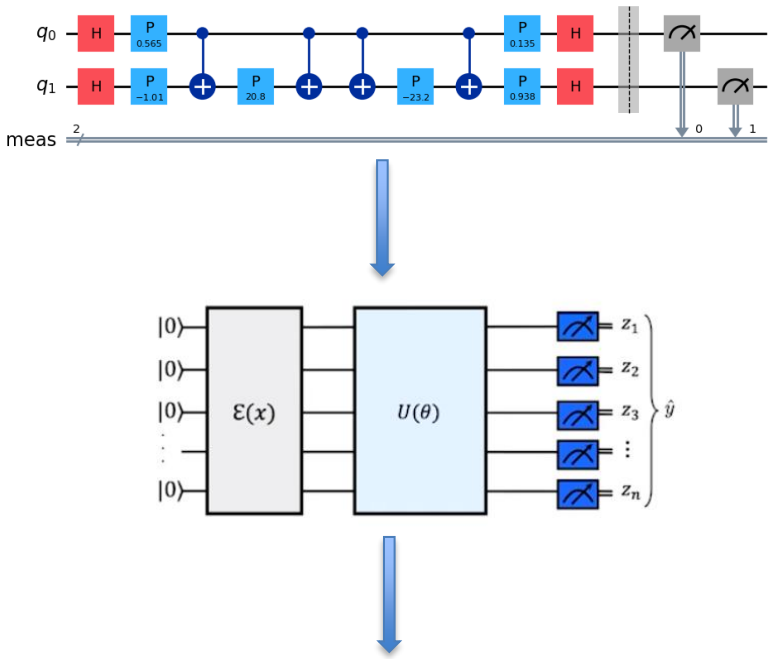


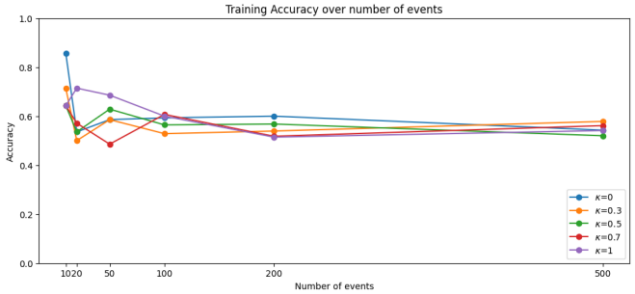
Graph representation of a leading jet

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



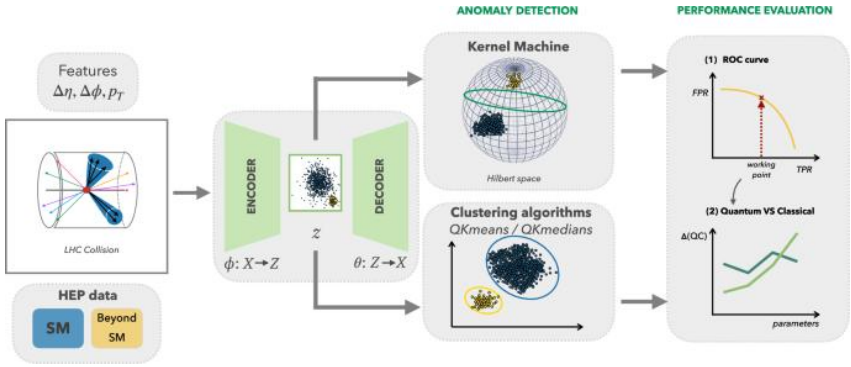


Accuracy for VQC in few events

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.**