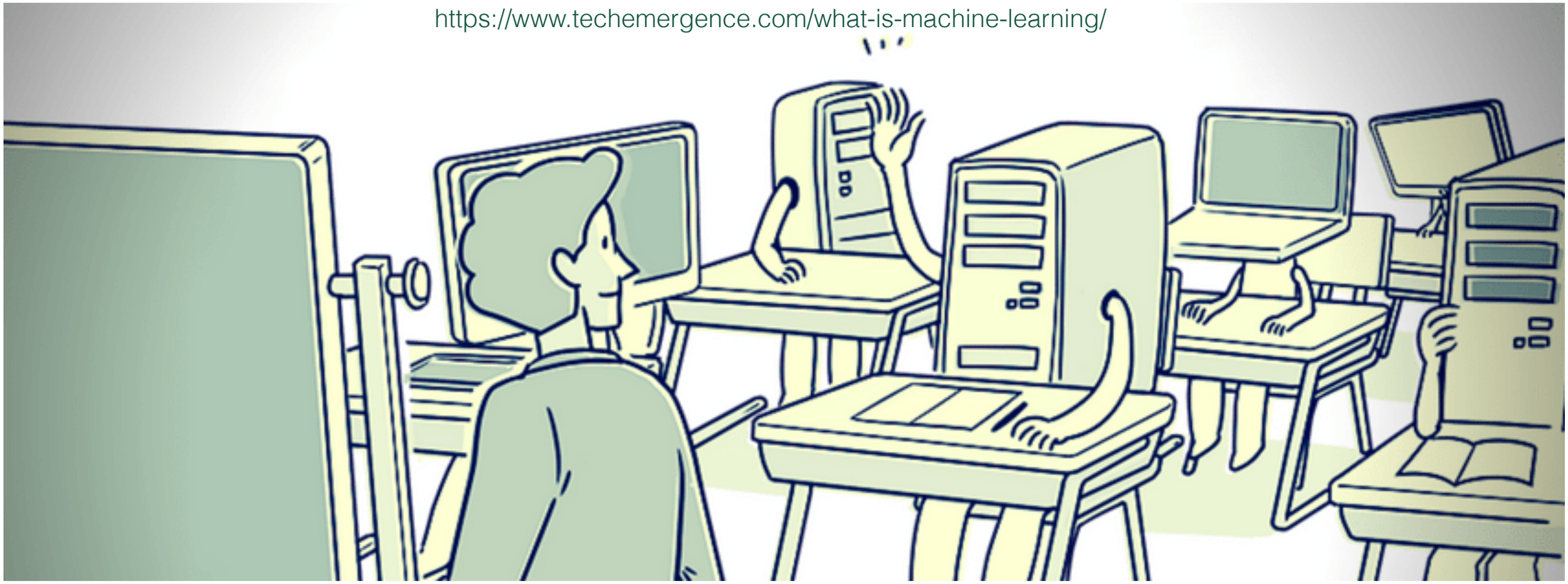
Introduction to Machine Learning

UNIVERSITY OF OREGON

Lecture 2

https://www.techemergence.com/what-is-machine-learning/



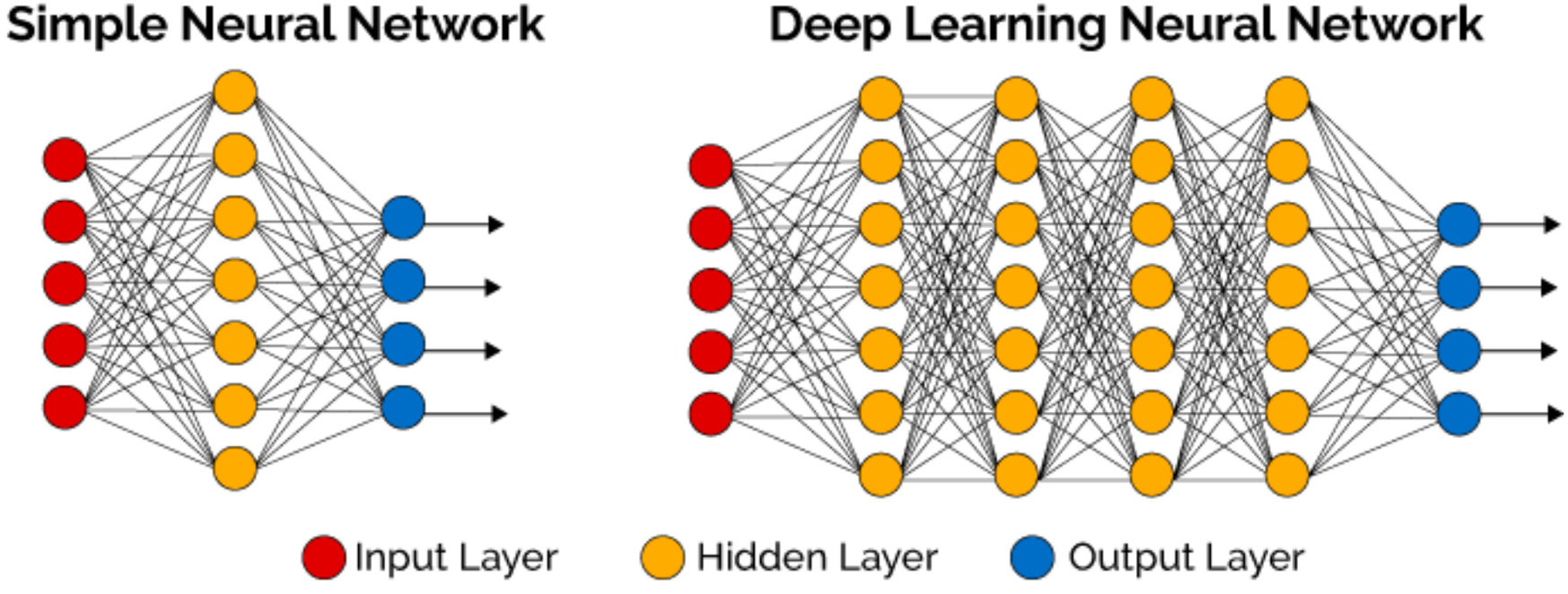
CTEQ Summer School 2019
University of Pittsburgh
July 2019

Bryan Ostdiek

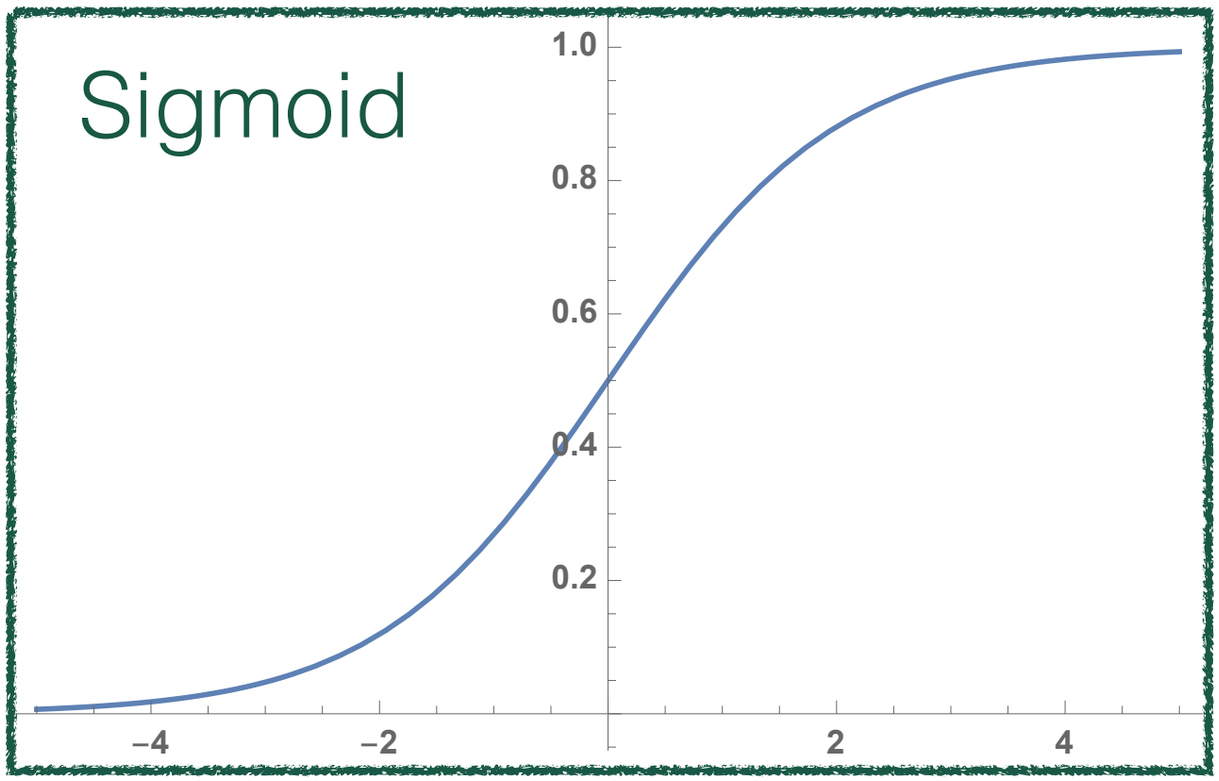
Disappearing Gradient

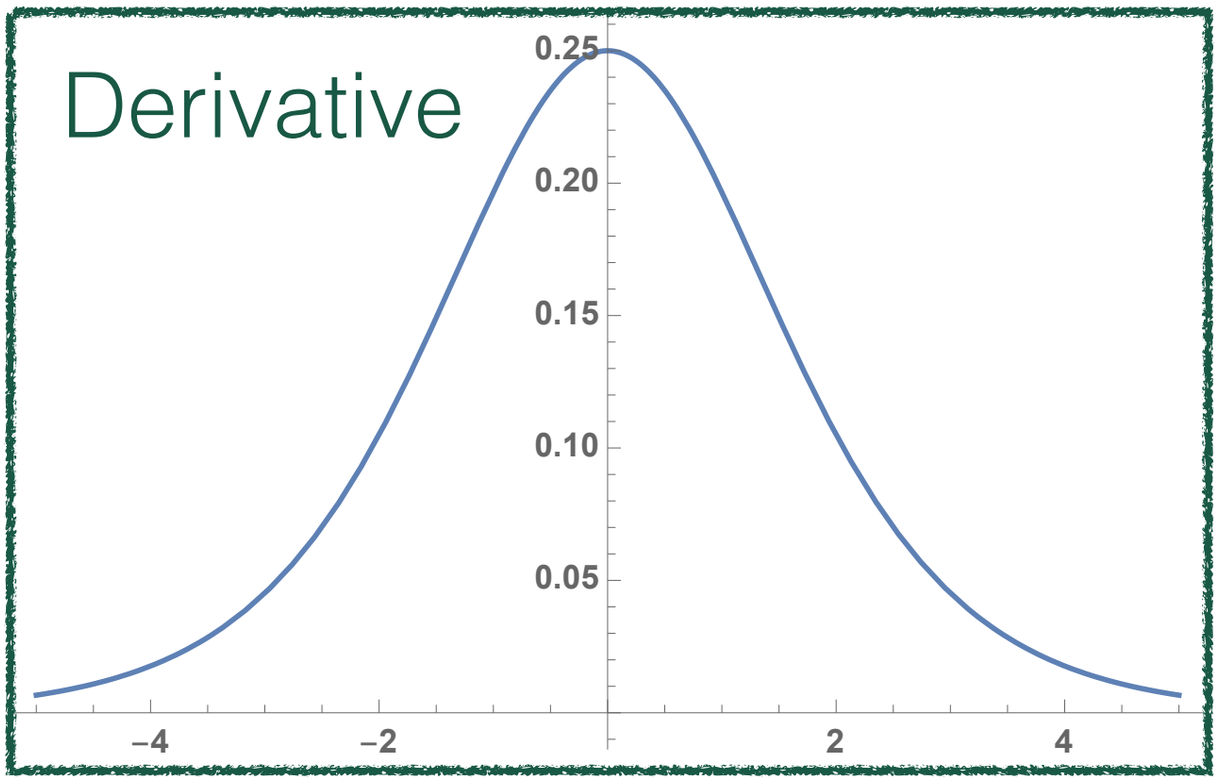
Deep learning uses raw inputs, with many hidden layers to let the machine come up with its own features.

How deep can we go before running into problems?



Disappearing Gradient





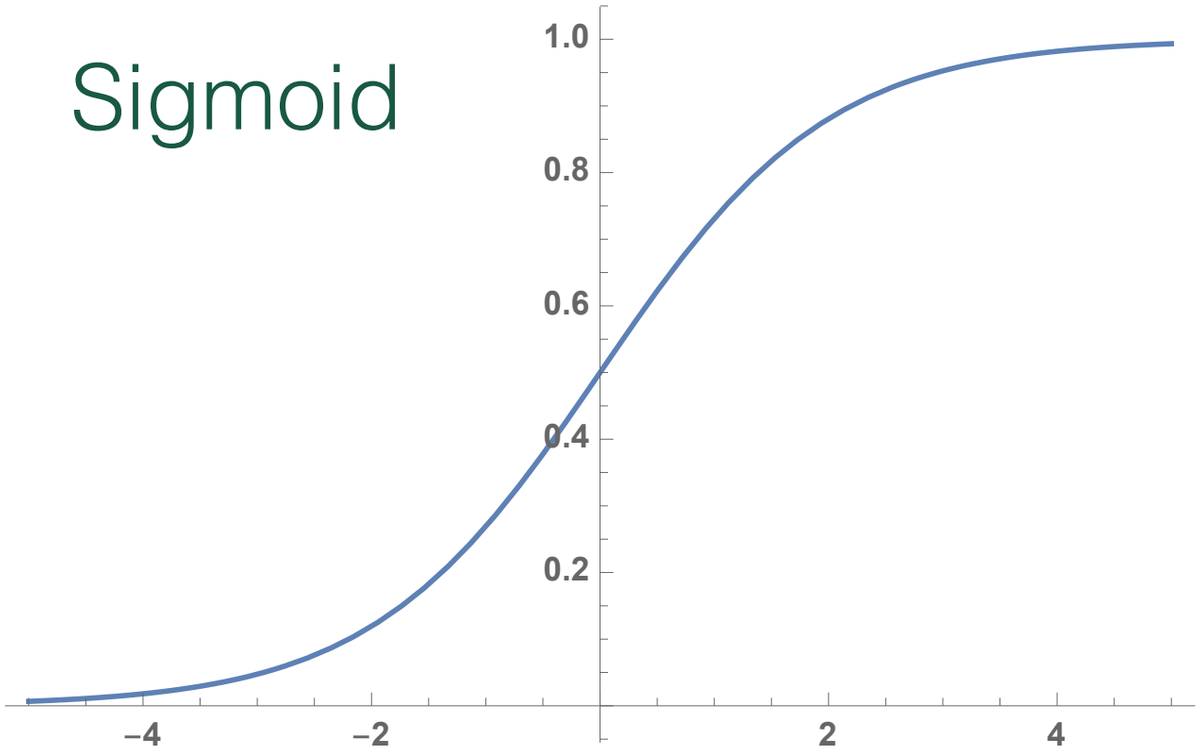
Chain rule for gradient of network involves multiple factors of the derivative multiplied together

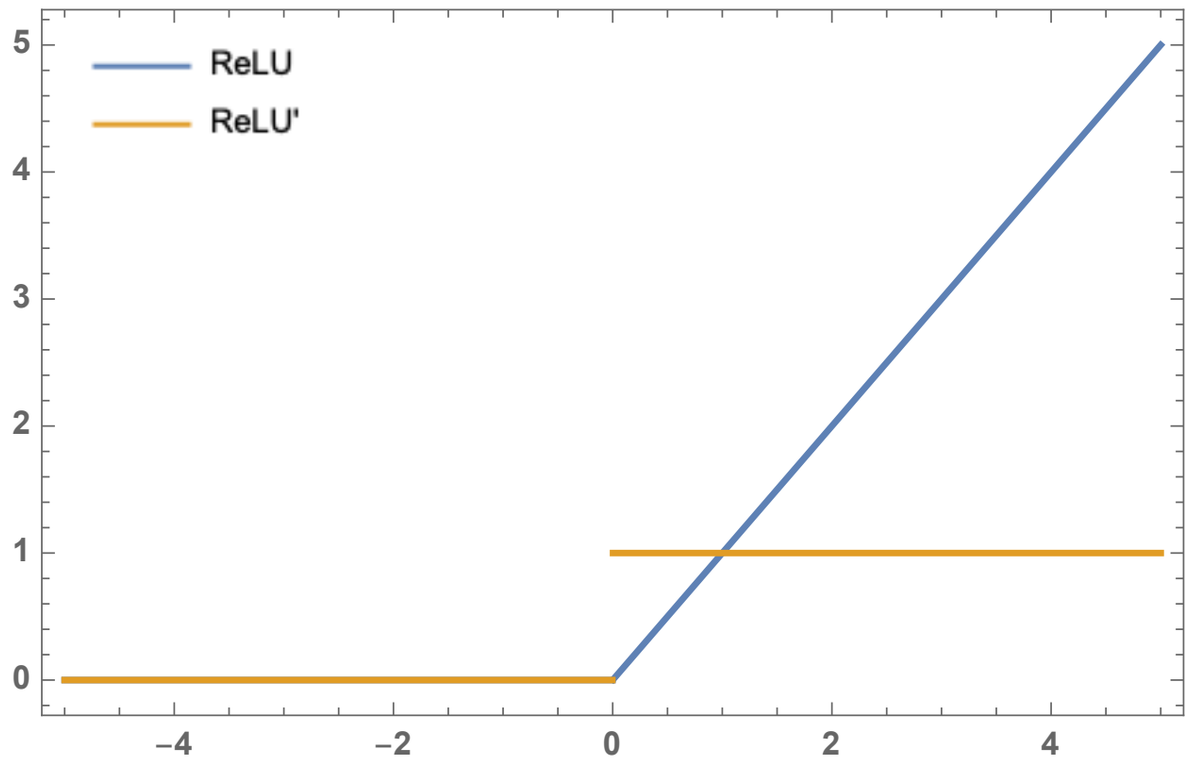
$$(0.25)^4 = 0.0039$$

Deep networks with Sigmoid activations have exponentially hard time training early layers

Disappearing Gradient

Sigmoid





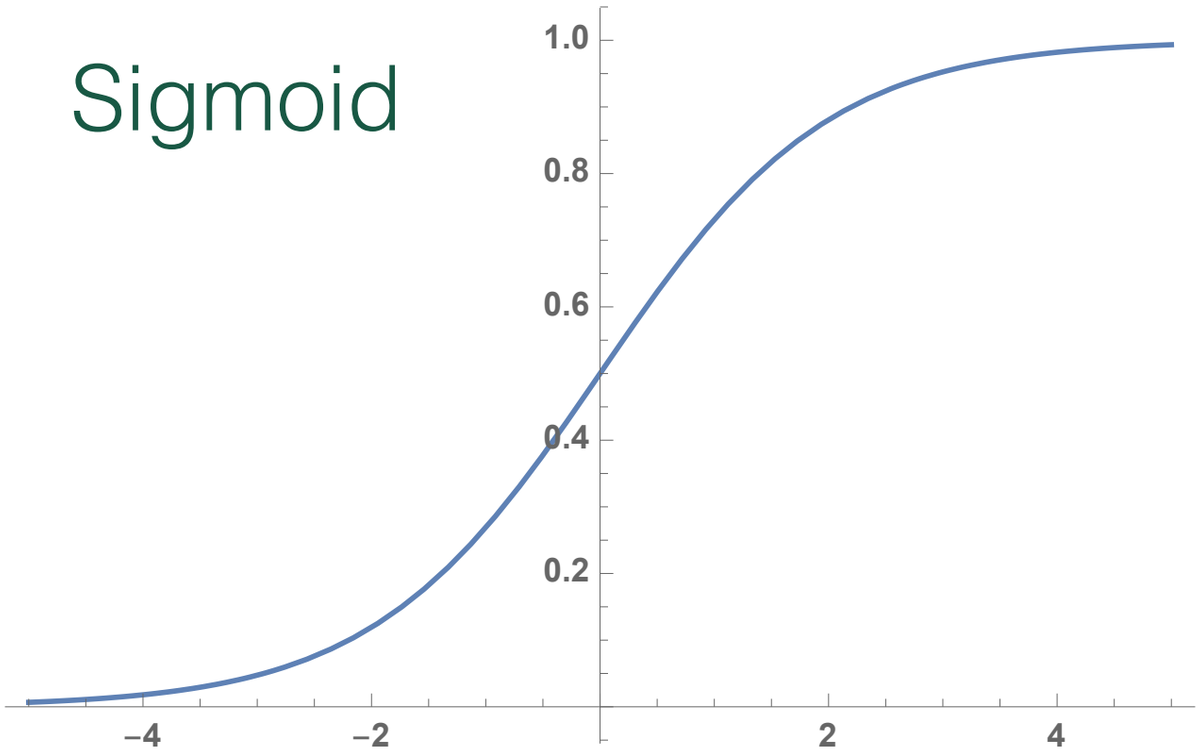
Using the Rectified Linear Unit (ReLU) solves this problem.
ReLU(x) = {0 if x <=0, x if x >0}

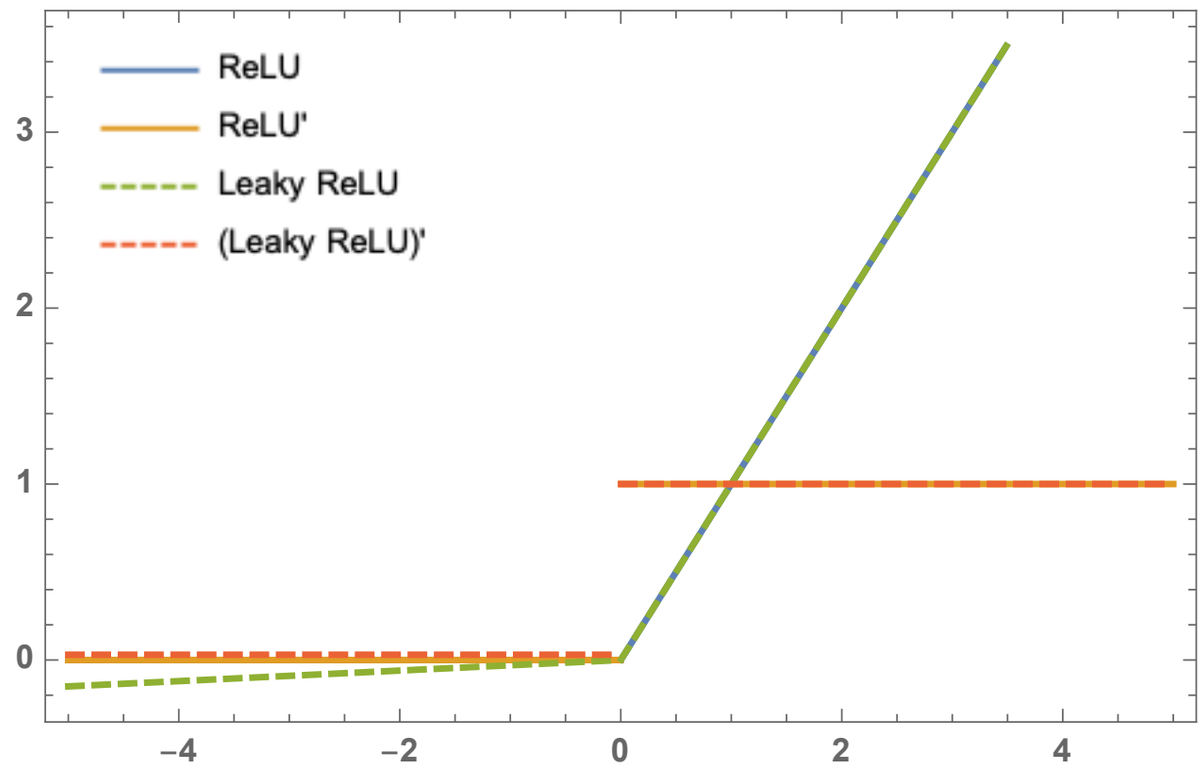
Still has nonlinearity which allows network to learn complicated patterns

Nodes can die (derivative always 0 so cannot update)

Disappearing Gradient

Sigmoid





Leaky Rectified Linear Unit (LeakyReLU) solves this problem.

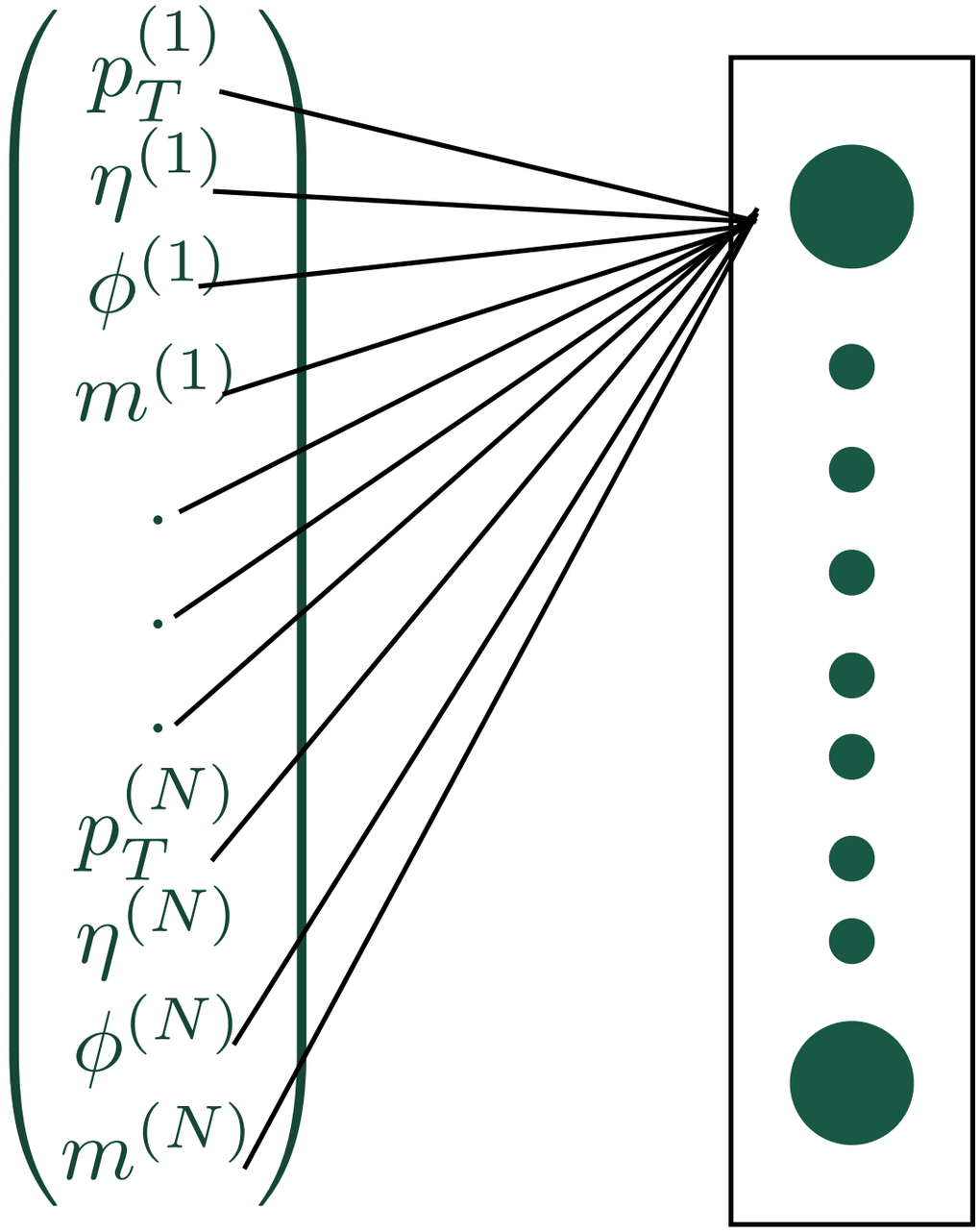
LeakyReLU(x) = {alpha*x if x <=0, x if x >0}

I have never had to use this in practice

Deep learning

- Deep learning trained on raw (4-vector) data out classified shallow network and BDT on high-level variables
- Give the network as much data as possible and let it learn what it wants
- Is 4-vectors the best way to represent the data?
- Are there more efficient forms?
- Most news coverage of deep learning is about image recognition, can we use this?

Deep learning



- Deep learning with 4-vectors performs well
- Each node is a linear combination of variables with different units?!?!
- Is this all the information available in the detectors?

How much information is in a jet?

[1704.08249]

- **M-body phase space.** For M-body phase space, we can define the coordinates of that phase space by $M-1$ transverse momentum fractions z_i, for $i = 1, \ldots, M-1$, and $2M-3$ pairwise angles θ_{ij} between particles i and j. The remaining

$$\binom{M}{2} - (2M-3) = \frac{1}{2}(M-2)(M-3)\,,$$

pairwise angles are then uniquely determined by the geometry of points in a plane.[6] To determine all of these phase space variables, we extend the set of N-subjettinesses that were measured in the 2- and 3-body case. In this case, the $3M-4$ observables we measure are:

$$\left\{\tau_1^{(0.5)}, \tau_1^{(1)}, \tau_1^{(2)}, \tau_2^{(0.5)}, \tau_2^{(1)}, \tau_2^{(2)}, \ldots, \tau_{M-2}^{(0.5)}, \tau_{M-2}^{(1)}, \tau_{M-2}^{(2)}, \tau_{M-1}^{(1)}, \tau_{M-1}^{(2)}\right\}. \qquad (2.8)$$

Note that there are $3(M-2)+2 = 3M-4$ observables, and these will span the space of phase space variables for generic momenta configurations, when all particles have non-zero energy and are a finite angle from one another.

$$\tau_N^{(\beta)} = \frac{1}{p_{TJ}} \sum_{i \in \text{Jet}} p_{Ti} \min\left\{R_{1i}^{\beta}, R_{2i}^{\beta}, \ldots, R_{Ni}^{\beta}\right\}.$$

How much information is in a jet?

[1704.08249]

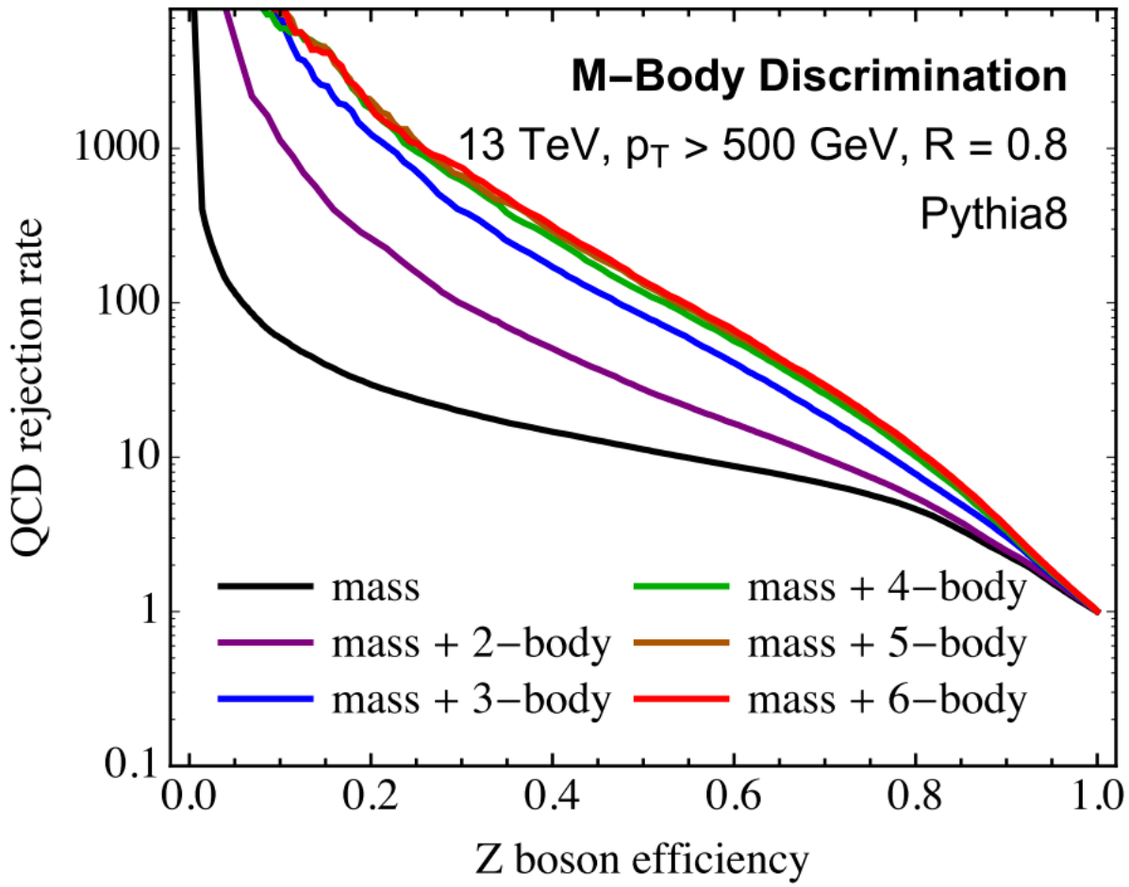


Figure 1. Z boson jet efficiency vs. QCD jet rejection rate plot as generated by the deep neural network. Details of the event simulation, jet finding, and machine learning are described in section 3. The different curves correspond to the mass plus collections of observables that uniquely define M-body phase space. Discrimination power is seen to saturate when 4-body phase space is resolved.

Including all of the 4-vectors doesn't improve the classification

Boosted Z

Represent a jet as an image

[1501.05968]

Representing data: Images, early works took all of the pixels, then unrolled them into a line

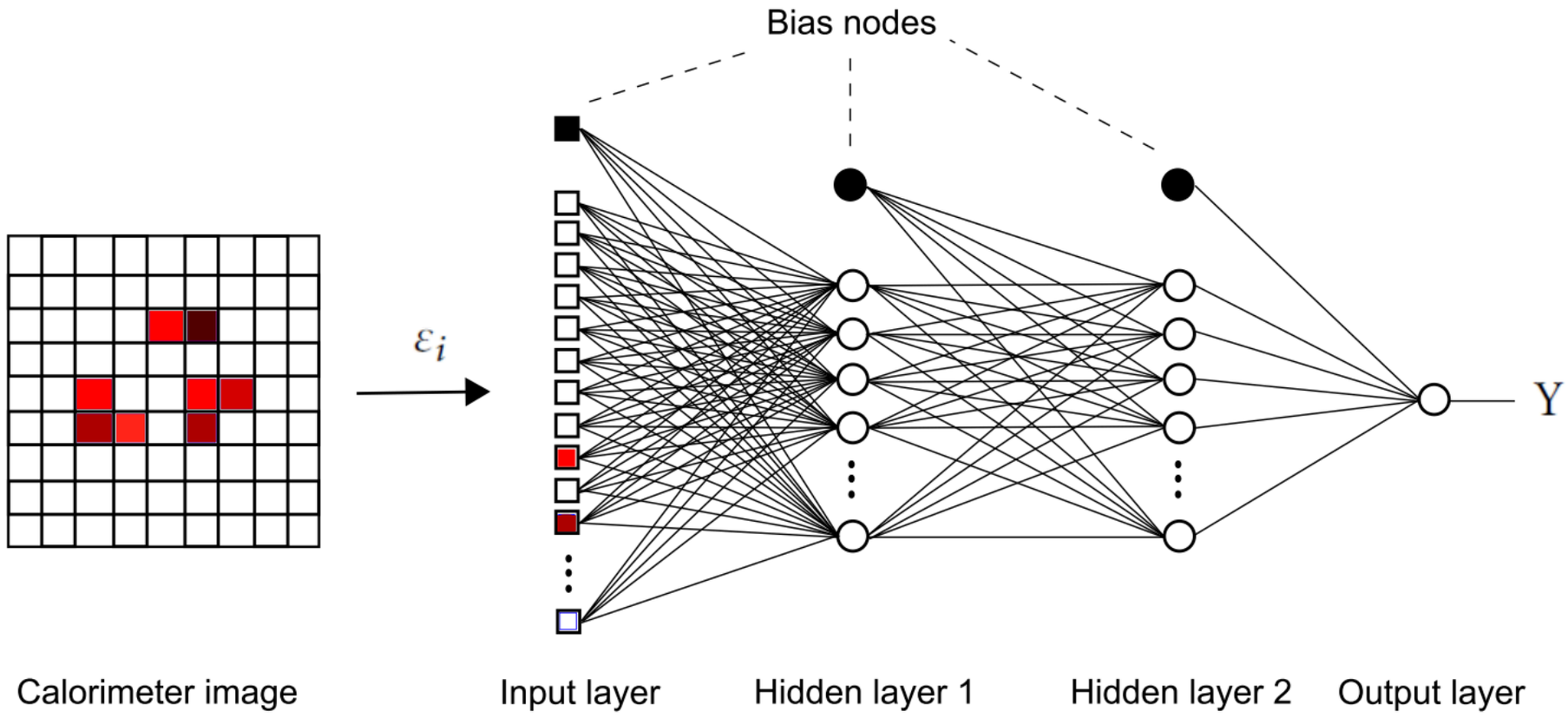
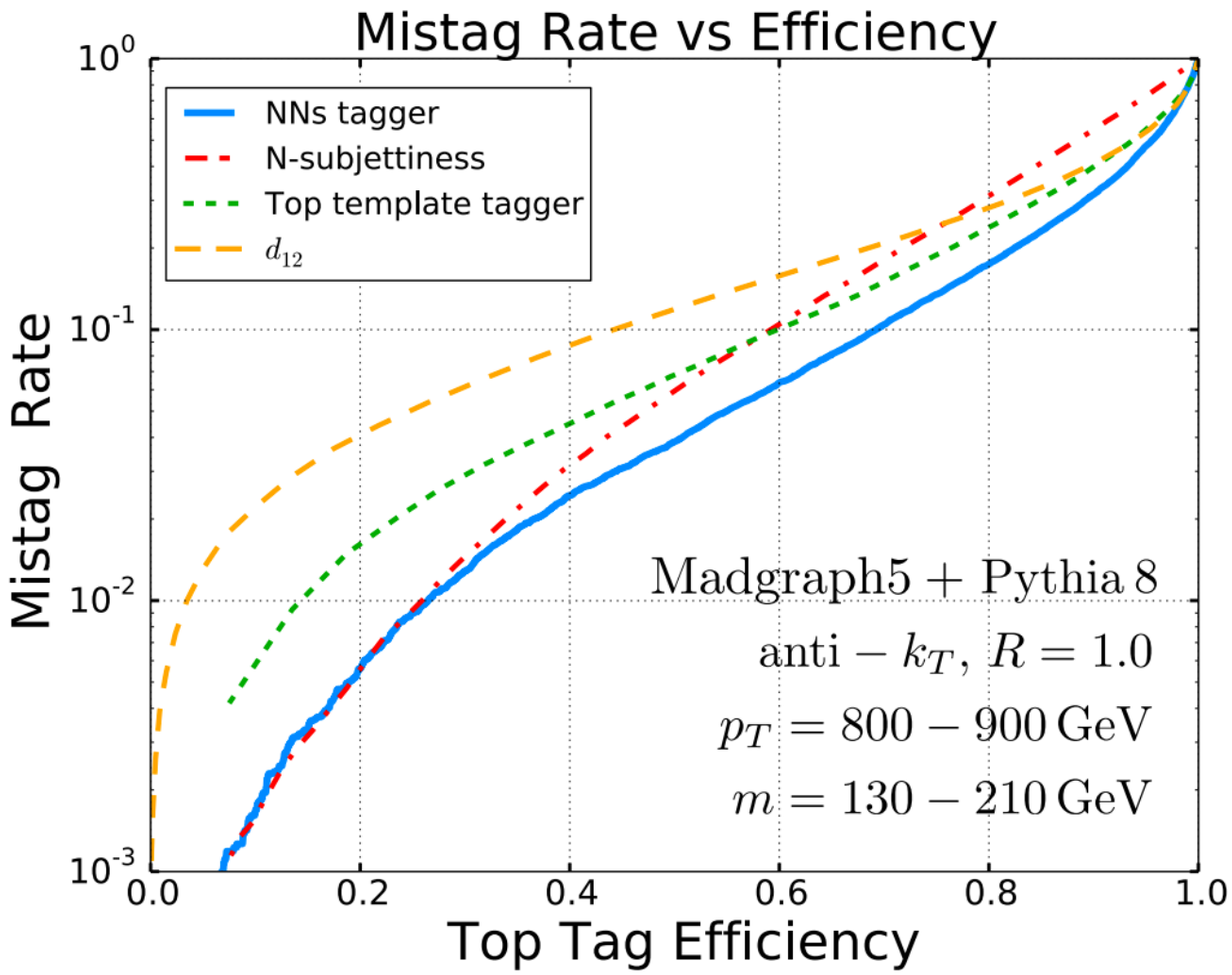


Figure 1. Graphical representation of the Artificial Neural Network (ANN).

Represent a jet as an image

[1501.05968]



- Claims to be better than N-subjettiness, but didn't use whole basis
- Used different signal, so can't do direct comparison to last study
- Didn't use the methods traditionally used for image recognition

Represent a jet as an image

Image classification usually uses ***Convolutional Layers*** as opposed to fully connected (dense) layers

data = [1, 2, 4, 3, 2, 5, 1] filter or kernel = [1, 0, -1]

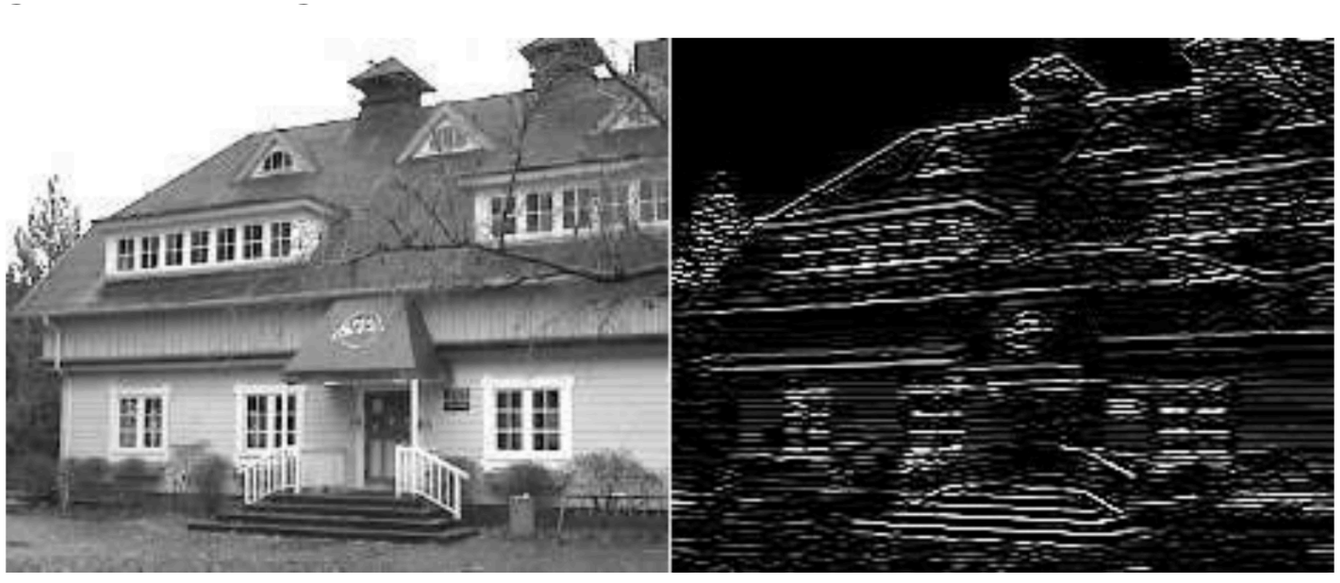
convolved data = [-3, -1, 2, -2, 1]

Builds in translational invariance
Less parameters to fit

https://leonardoaraujosantos.gitbooks.io/artificial-inteligence/content/more_images/Convolution_schematic.gif

Convolutional filters

$$\begin{pmatrix} -1 & -1 & -1 \\ 2 & 2 & 2 \\ -1 & -1 & -1 \end{pmatrix}$$



Horizontal line detector

$$\begin{pmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{pmatrix}$$

Edge detector

http://aishack.in/tutorials/image-convolution-examples/

Jet Images with CNNs

[1511.05190]

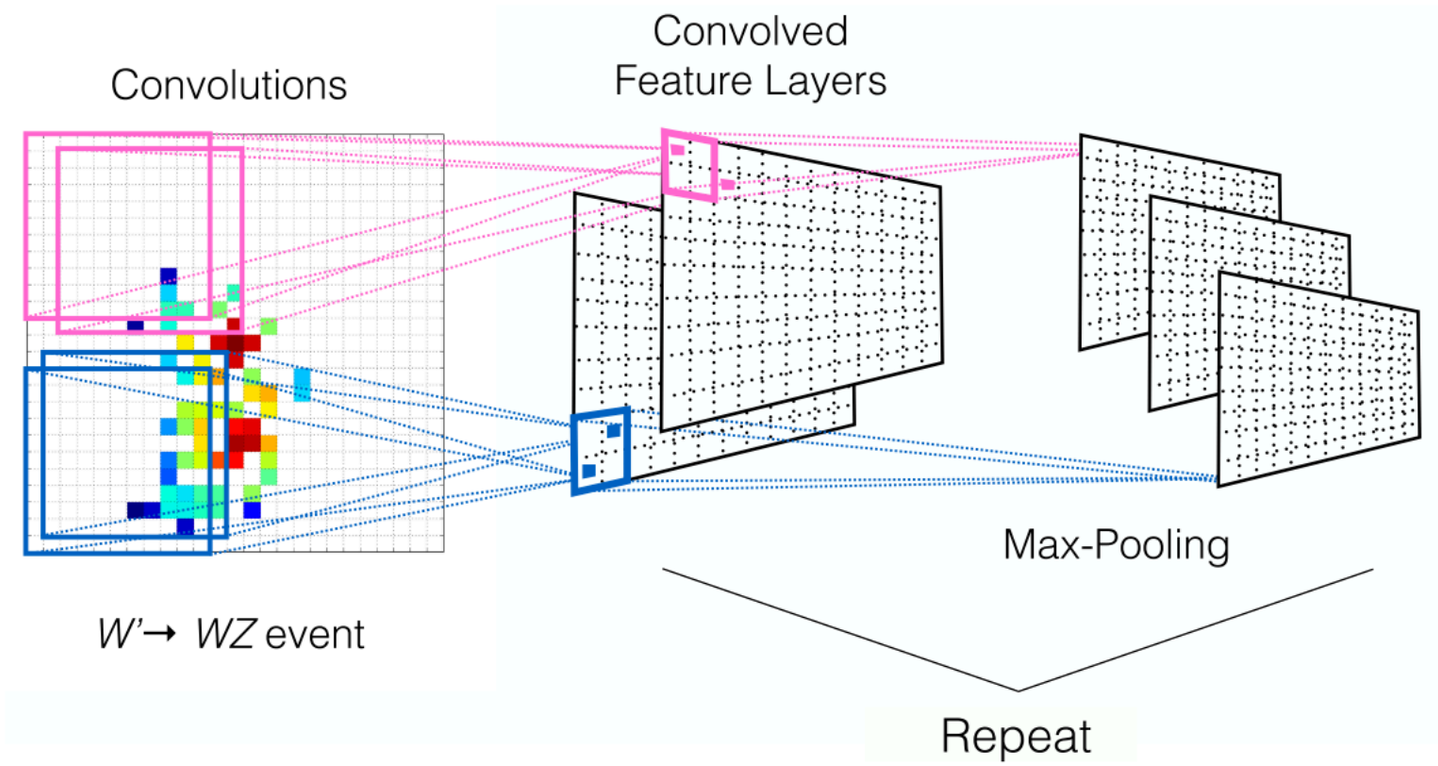


Figure 5: The convolution neural network concept as applied to jet-images.

Boosted W

Jet Images with CNNs

[1511.05190]

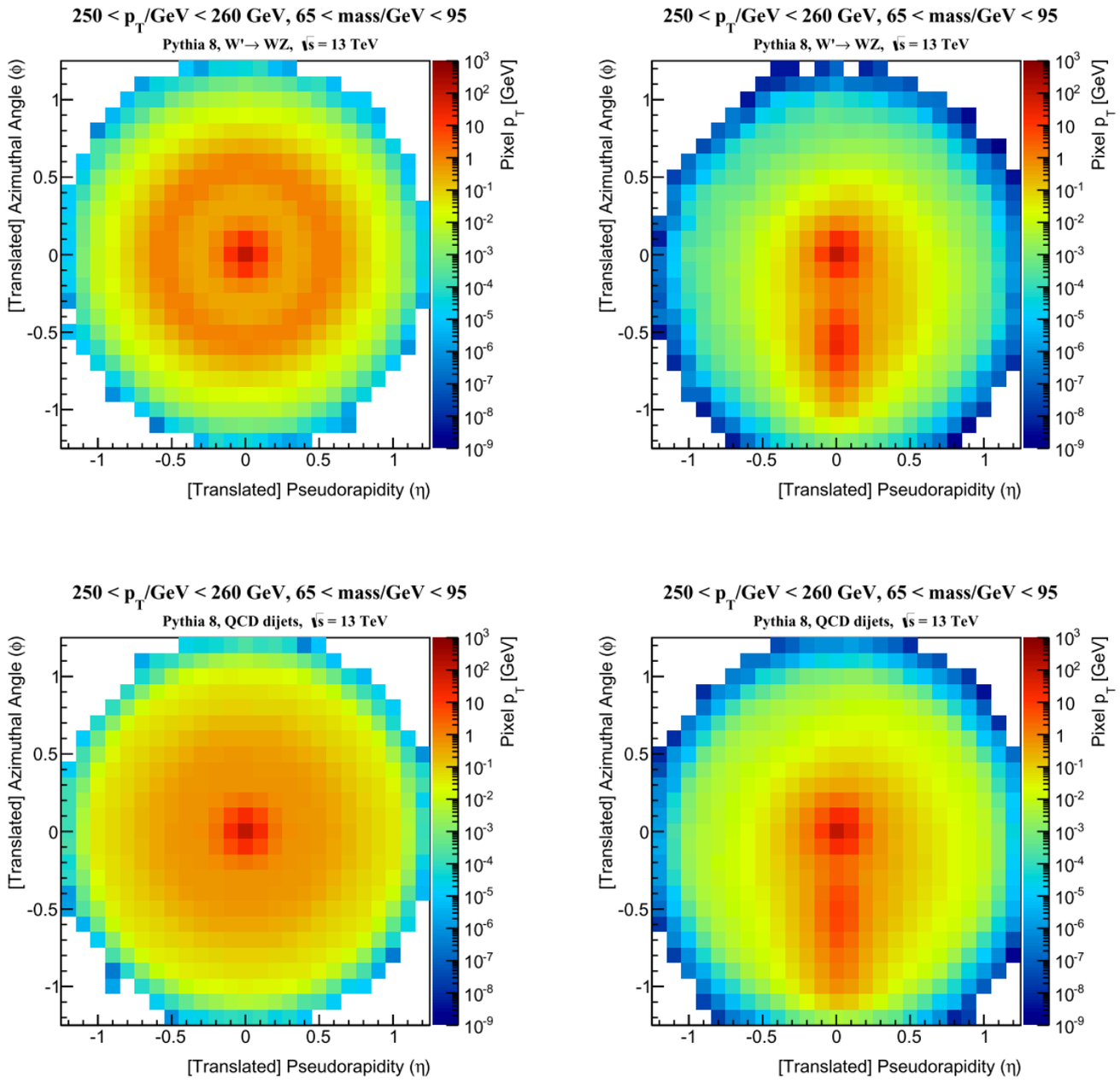


Figure 2: The average jet image for signal W jets (top) and background QCD jets (bottom) before (left) and after (right) applying the rotation, re-pixelation, and inversion steps of the pre-processing. The average is taken over images of jets with 240 GeV $< p_T <$ 260 GeV and 65 GeV $<$ mass $<$ 95 GeV.

Boosted W

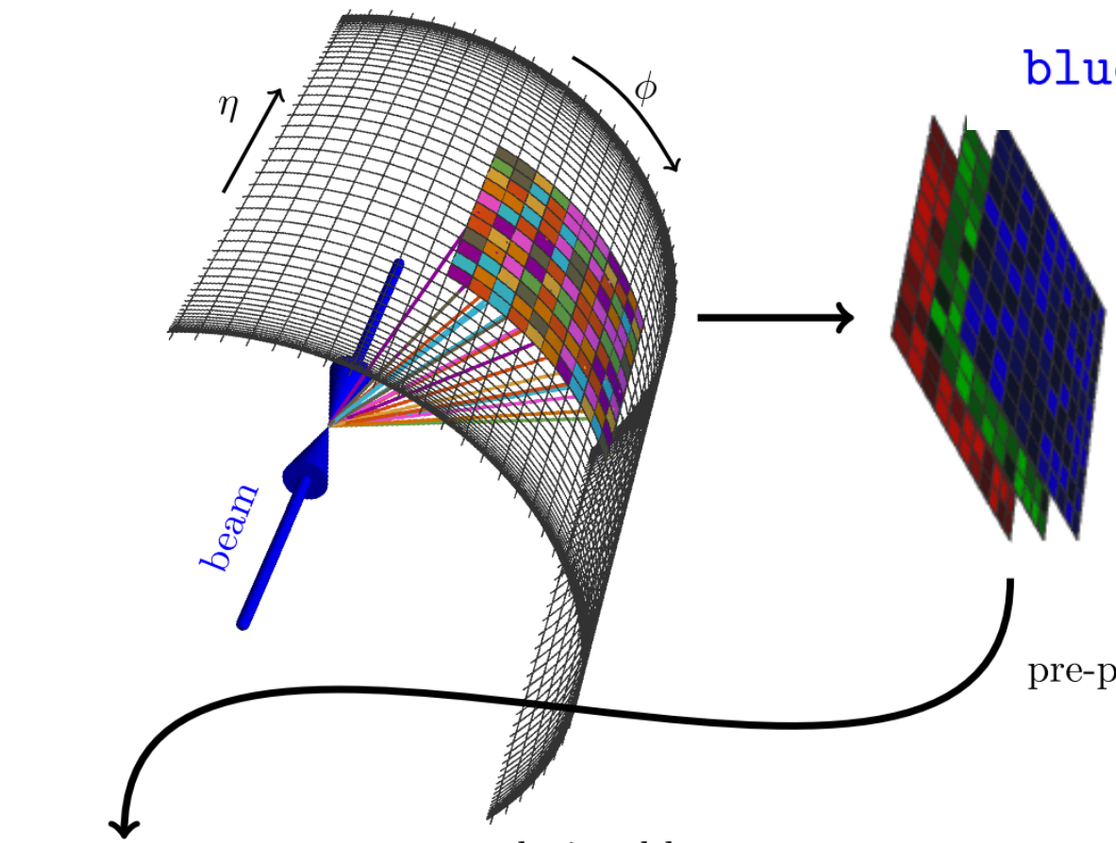
Jet Images with CNNs round 2

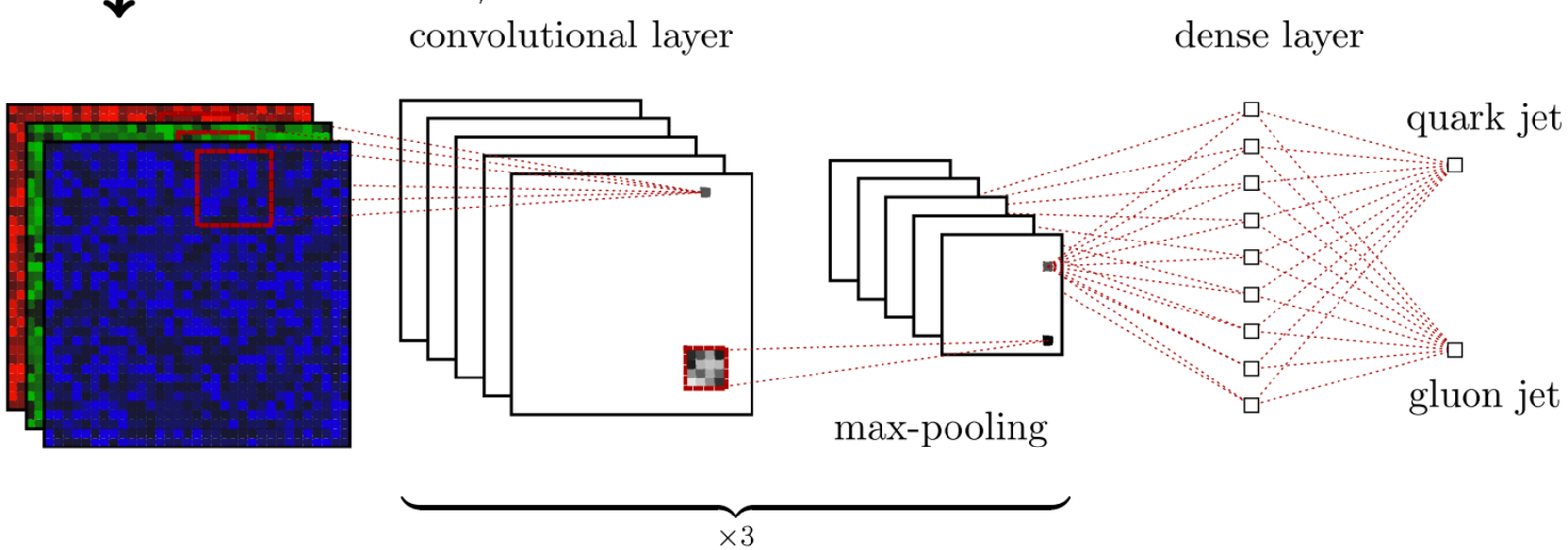
[1612.01551]

`red` = transverse momenta of charged particles

`green` = the transverse momenta of neutral particles

`blue` = charged particle multiplicity





Quark vs. Gluon

Jet Images with CNNs round 2

[1612.01551]

3.1 Pre-processing

The following series of data-motivated pre-processing steps were applied to the jet images:

1. **Center:** Center the jet image by translating in (η, ϕ) so that the total p_T-weighted centroid pixel is at $(\eta, \phi) = (0, 0)$. This operation corresponds to rotating and boosting along the beam direction to center the jet.

2. **Crop:** Crop to a 33×33 pixel region centered at $(\eta, \phi) = (0, 0)$, which captures the region with $\eta, \phi \in (-R, R)$ for $R = 0.4$.

3. **Normalize:** Scale the pixel intensities such that $\sum_{ij} I_{ij} = 1$ in the image, where i and j index over the pixels.

4. **Zero-center:** Subtract the mean μ_{ij} of the normalized training images from each image, transforming each pixel intensity as $I_{ij} \to I_{ij} - \mu_{ij}$.

5. **Standardize:** Divide each pixel value by the standard deviation σ_{ij} of that pixel value in the normalized training dataset, $I_{ij} \to I_{ij}/(\sigma_{ij} + r)$. A value of $r = 10^{-5}$ was used to suppress noise.

Quark vs. Gluon

Jet Images with CNNs round 2

[1612.01551]

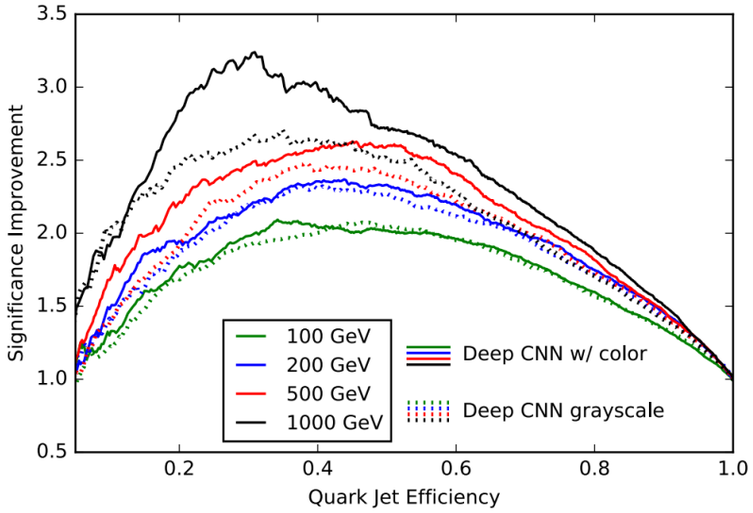


Figure 6: SIC curve of deep convolutional network performance on Pythia jets with color (solid) and without color (dotted). The introduction of color becomes more helpful at higher energies, with the largest improvement on the 1000 GeV jets.

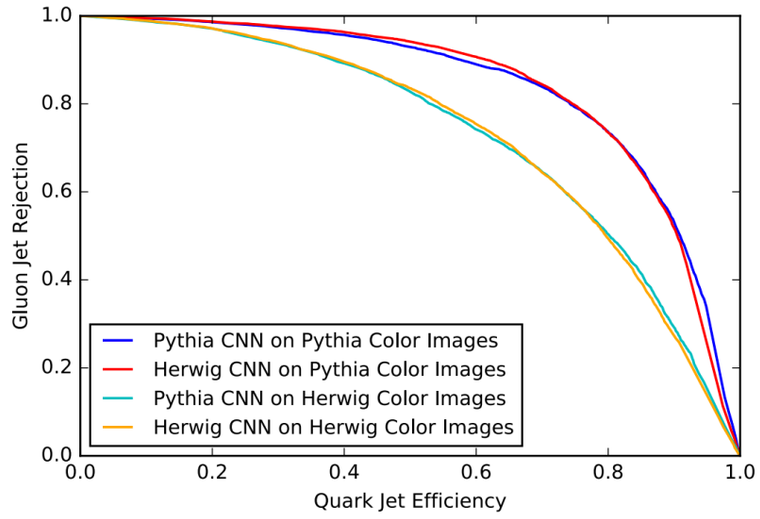


Figure 8: ROC curves for the Pythia- and Herwig-trained CNNs applied to 200 GeV samples generated with both of the generators. Remarkably, the network performance seems robust to which samples are used for training.

Can start to learn physics from the results of machine learning!

Quark vs. Gluon

Jet Representations

4-vectors of all the hadrons in a jet is an easy representation of the data, but not necessarily efficient

Using N-subjettiness, can get good classification with more intuition for what the machine is using in it's decisions

Images can also be used and seem to get very good separation

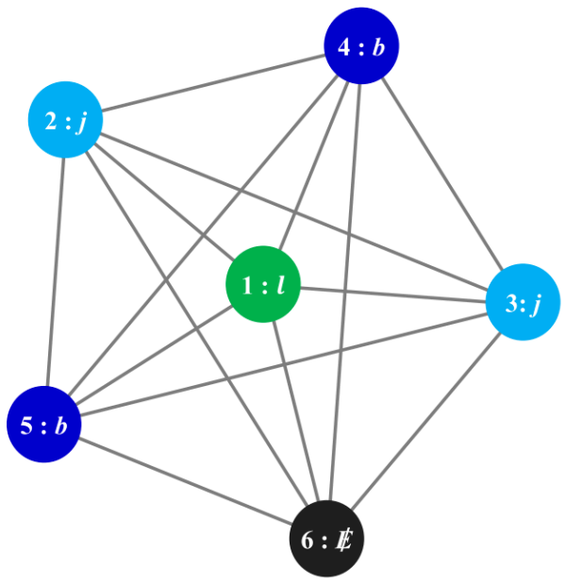
Each study used a different test signal, so can't directly compare performance

Are there other representations of the data which could be easier for a machine to learn from?

Representing a jet as a tree/graph

[1702.00748], [1711.02633], [1711.09059]
[1807.09088], [1810.05165], [1902.08570]

Can build in permutation invariance, other symmetries



x

	Photon	lepton charge	b-jet or light jet	MET	p_T (TeV)	E (TeV)	m (TeV)	
$x_1 = ($	0	-1	0	0	0.0229	0.0289	0.0000	$)$
$x_2 = ($	0	0	-1	0	0.2637	0.3304	0.0373	$)$
$x_3 = ($	0	0	-1	0	0.1003	0.1888	0.0091	$)$
$x_4 = ($	0	0	1	0	0.0980	0.1146	0.0133	$)$
$x_5 = ($	0	0	1	0	0.0689	0.0773	0.0062	$)$
$x_6 = ($	0	0	0	1	0.2107	0.2107	0.0000	$)$

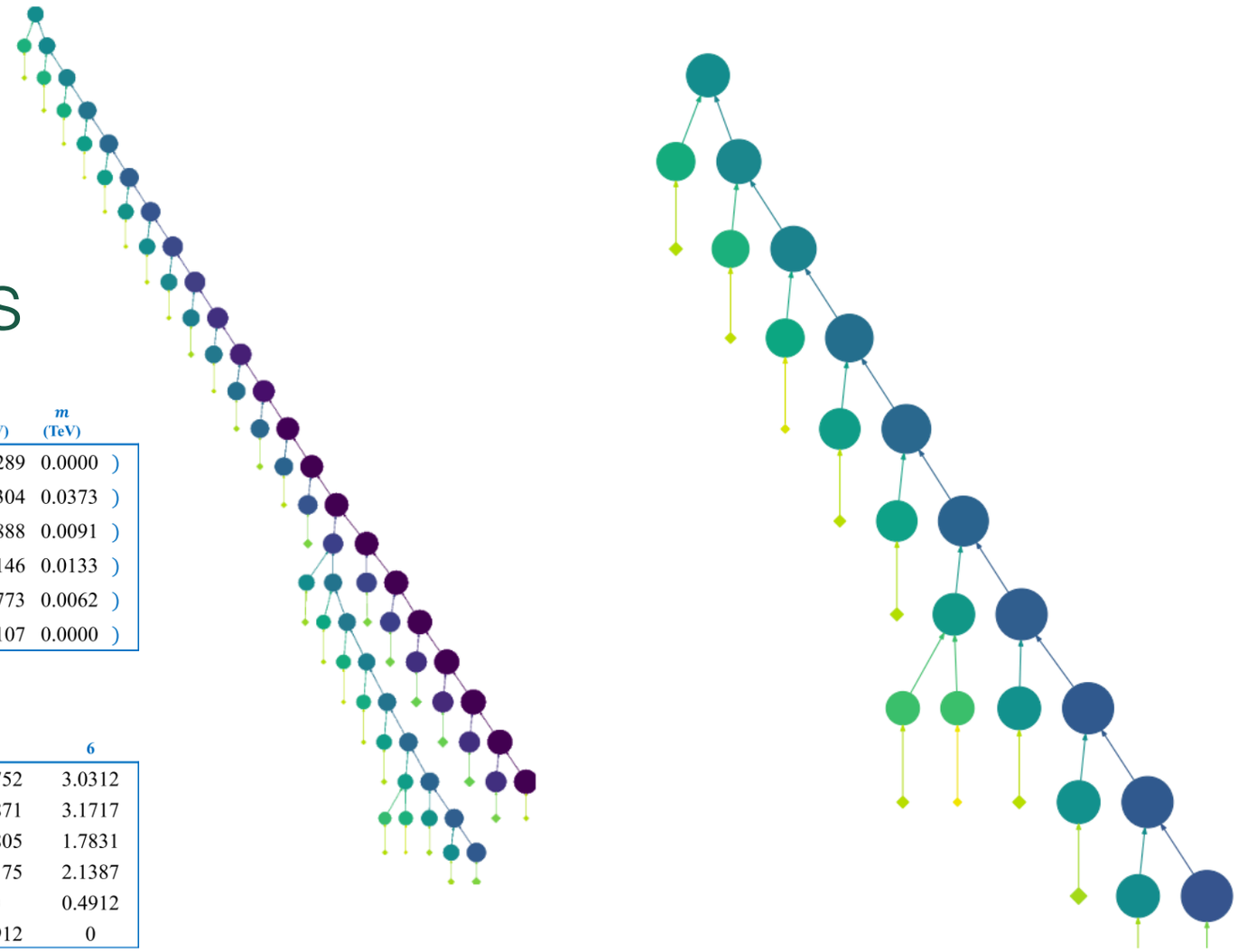
d

	1	2	3	4	5	6
1	0	1.3971	2.5649	1.2801	3.2752	3.0312
2	1.3971	0	1.9019	1.6688	3.0871	3.1717
3	2.5649	1.9019	0	3.4440	1.5805	1.7831
4	1.2801	1.6688	3.4440	0	2.2175	2.1387
5	3.2752	3.0871	1.5805	2.2175	0	0.4912
6	3.0312	3.1717	1.7831	2.1387	0.4912	0

[1807.09088]

Gluon (1 TeV)

Quark (1 TeV)



[1711.02633]

Building physics into the Machine

Adding a Lorentz Layer [1707.08966]

Start with 4-vectors (not components of 4-vectors)

$$(k_{\mu,i}) = \begin{pmatrix} k_{0,1} & k_{0,2} & \cdots & k_{0,N} \\ k_{1,1} & k_{1,2} & \cdots & k_{1,N} \\ k_{2,1} & k_{2,2} & \cdots & k_{2,N} \\ k_{3,1} & k_{3,2} & \cdots & k_{3,N} \end{pmatrix}$$

Learn the linear combinations to use

$$k_{\mu,i} \xrightarrow{\text{CoLa}} \widetilde{k}_{\mu,j} = k_{\mu,i}\, C_{ij}$$

Compute Lorentz invariant quantities

$$\tilde{k}_j \xrightarrow{\text{LoLa}} \hat{k}_j = \begin{pmatrix} m^2(\tilde{k}_j) \\ p_T(\tilde{k}_j) \\ w^{(E)}_{jm}\, E(\tilde{k}_m) \\ w^{(d)}_{jm}\, d^2_{jm} \end{pmatrix}$$

End with 2 fully connected layers

Building physics into the Machine

Adding a Lorentz Layer [1707.08966]

Compute Lorentz invariant quantities

$$\tilde{k}_j \xrightarrow{\text{LoLa}} \hat{k}_j = \begin{pmatrix} m^2(\tilde{k}_j) \\ p_T(\tilde{k}_j) \\ w_{jm}^{(E)}\, E(\tilde{k}_m) \\ w_{jm}^{(d)}\, d_{jm}^2 \end{pmatrix}$$

To make this layer have learnable features, don't encode the metric, but leave it as weights to be updated

$$g = \mathrm{diag}(\quad 0.99 \pm 0.02, \\ -1.01 \pm 0.01, -1.01 \pm 0.02, -0.99 \pm 0.02)$$

The network learns to use the Minkowski metric!

Building physics into the Machine (2)

[1812.09722]

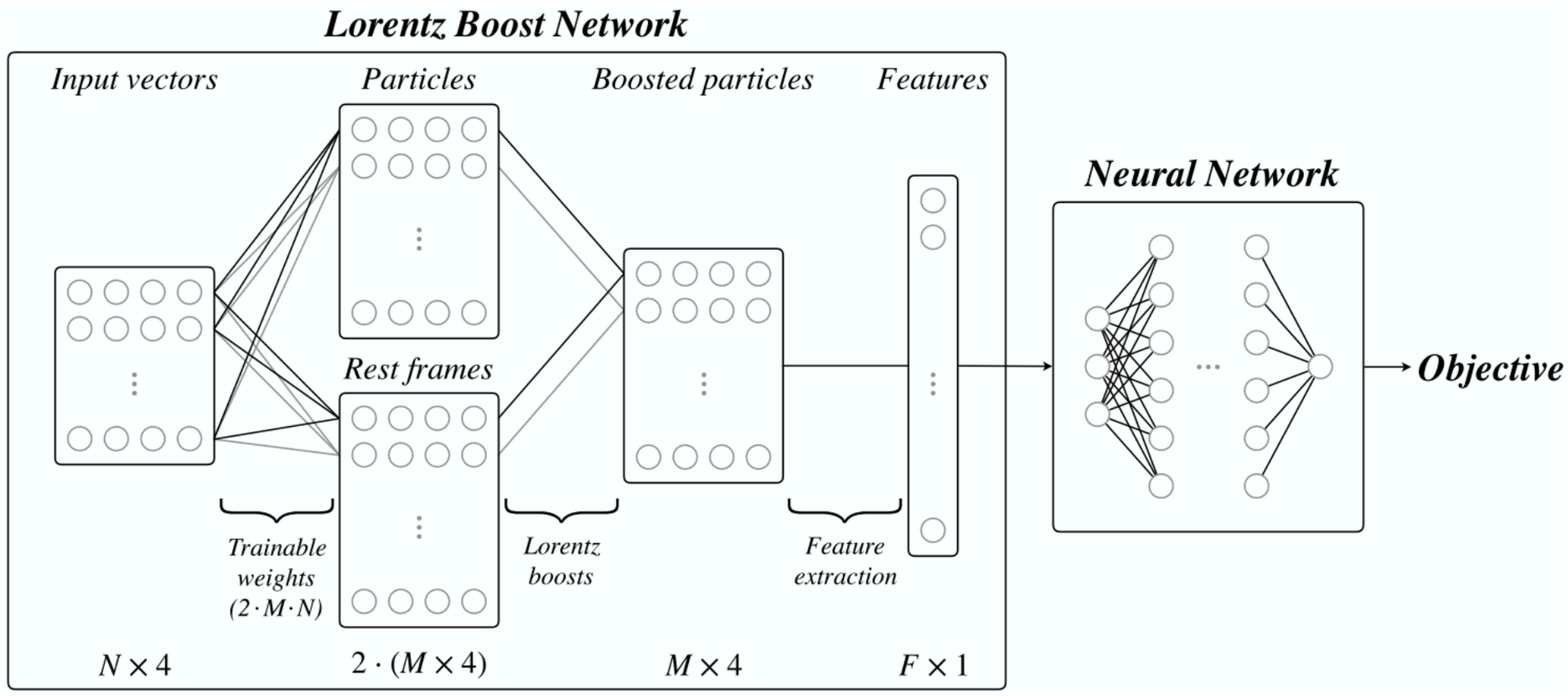
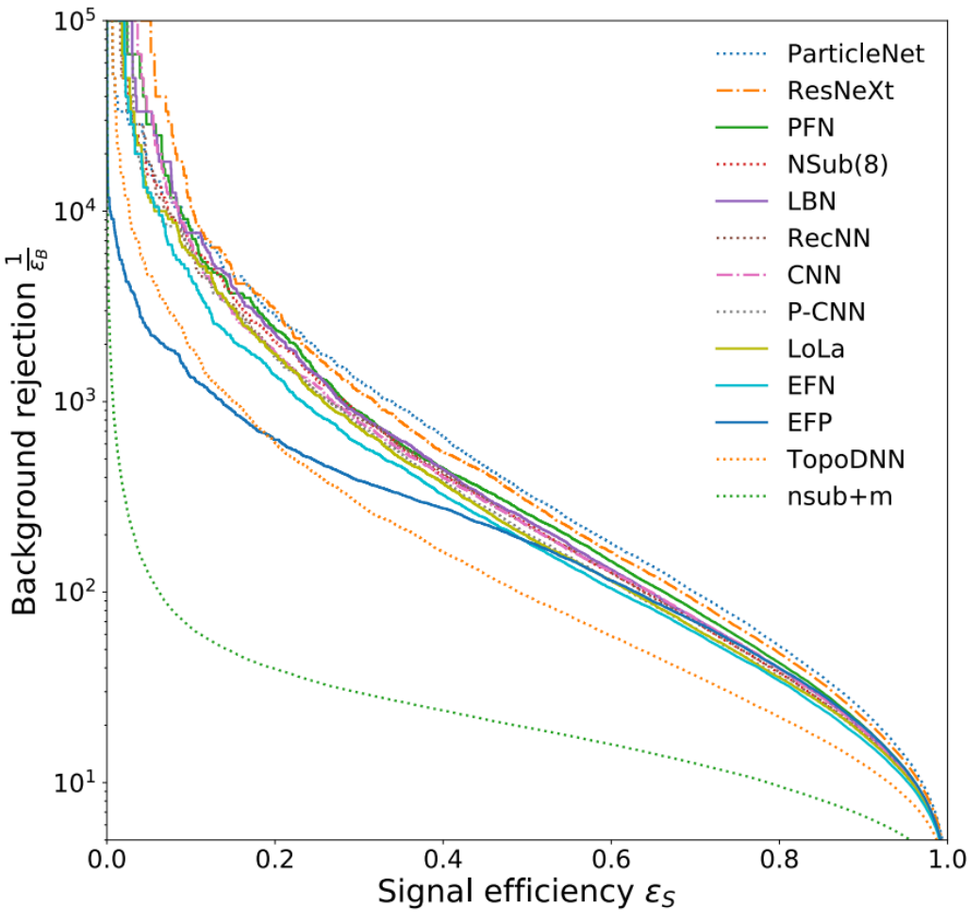


Figure 1: The two-stage deep neural network architecture consists of the Lorentz Boost Network (LBN) and a subsequent deep neural network (NN). In the LBN, the input four-vectors (E, p_x, p_y, p_z) are combined in two independent ways before each of the combined particles is boosted into its particular rest frame, which is formed from a different particle combination. The boosted particles are characterized by variables which can be, e.g., invariant masses, transverse momenta, pseudorapidities, and angular distances between them. These features serve as input to the second network designed to accomplish a particular analysis task.

A Fair Comparison

[1902.09914]: Try to find what representations of the data work best. Use the **same** training data and the ***same*** test data for all methods



	AUC	Accuracy	$1/\epsilon_B$ ($\epsilon_S = 0.3$)	#Parameters
CNN [16]	0.981	0.930	780	610k
ResNeXt [32]	0.984	0.936	1140	1.46M
TopoDNN [18]	0.972	0.916	290	59k
Multi-body N-subjettiness 6 [24]	0.979	0.922	856	57k
Multi-body N-subjettiness 8 [24]	0.981	0.929	860	58k
RecNN	0.981	0.929	810	13k
P-CNN	0.980	0.930	760	348k
ParticleNet [45]	0.985	0.938	1280	498k
LBN [19]	0.981	0.931	860	705k
LoLa [22]	0.980	0.929	730	127k
Energy Flow Polynomials [21]	0.980	0.932	380	1k
Energy Flow Network [23]	0.979	0.927	600	82k
Particle Flow Network [23]	0.982	0.932	880	82k
GoaT (see text)	0.985	0.939	1440	25k

Table 1: Single-number performance metrics for all algorithms evaluated on the test sample. We quote the area under the ROC curve (AUC), the accuracy, and the background rejection at a signal efficiency of 30%. The number of trainable parameters of the model is given as well. Performance metrics for the GoaT meta-tagger are based on a subset of events.

Big Transition

- Discussed many different representations of data for deep learning on jets
- Everything was still “supervised” classification on a labeled dataset
- Must train on Monte Carlo simulated data and then apply to real LHC data
- Are there ways to train directly on real data to avoid any mis-modeling effects?

Autoencoders

[1808.08979] and [1808.08992]

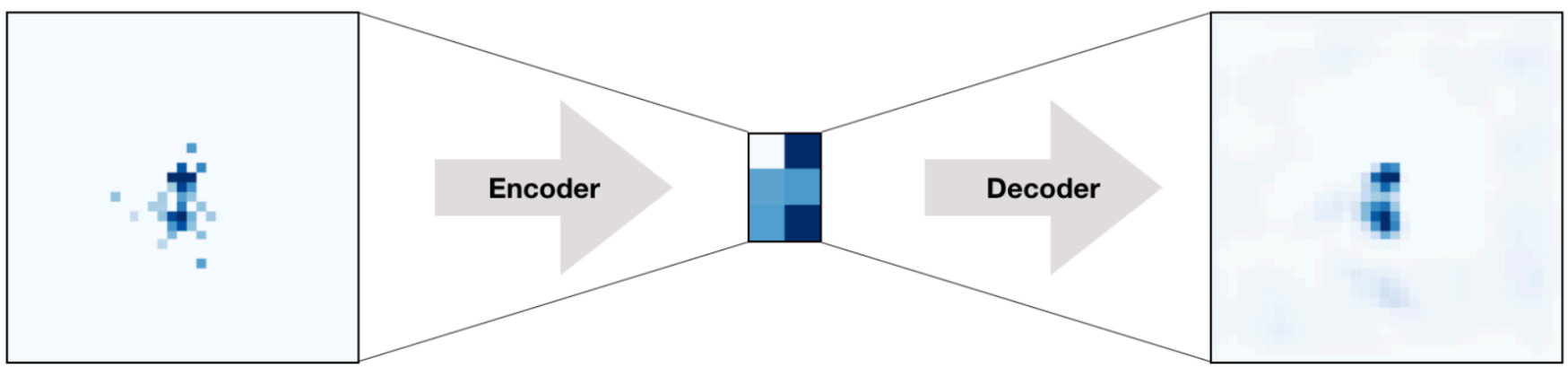


Figure 1: The schematic diagram of an autoencoder. The input is mapped into a low(er) dimensional representation, in this case 6-dim, and then decoded.

Minimize the reconstruction error

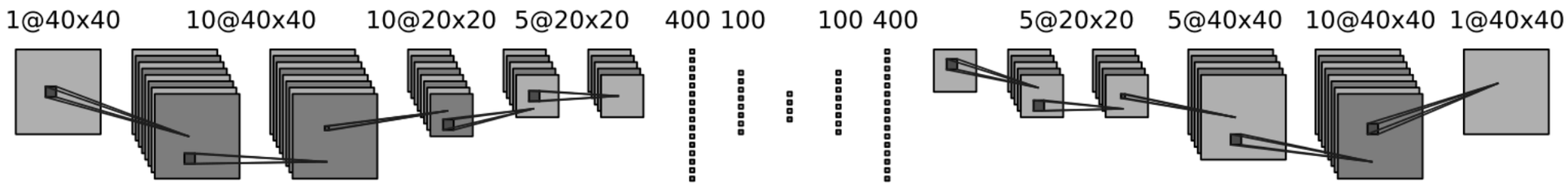


Figure 2: Architecture of the image-based autoencoder network. The 40×40 images are average-pooled to 20×20 images before entering the bottleneck. The dense units are first reduced from 400 to 100, the minimum size at the bottleneck is variable.

Autoencoders

[1808.08979] and [1808.08992]

$$L_{\text{auto}} = \sum_{\text{1600 pixels}} \left(k_T^{\text{norm,in}} - k_T^{\text{auto}} \right)^2$$

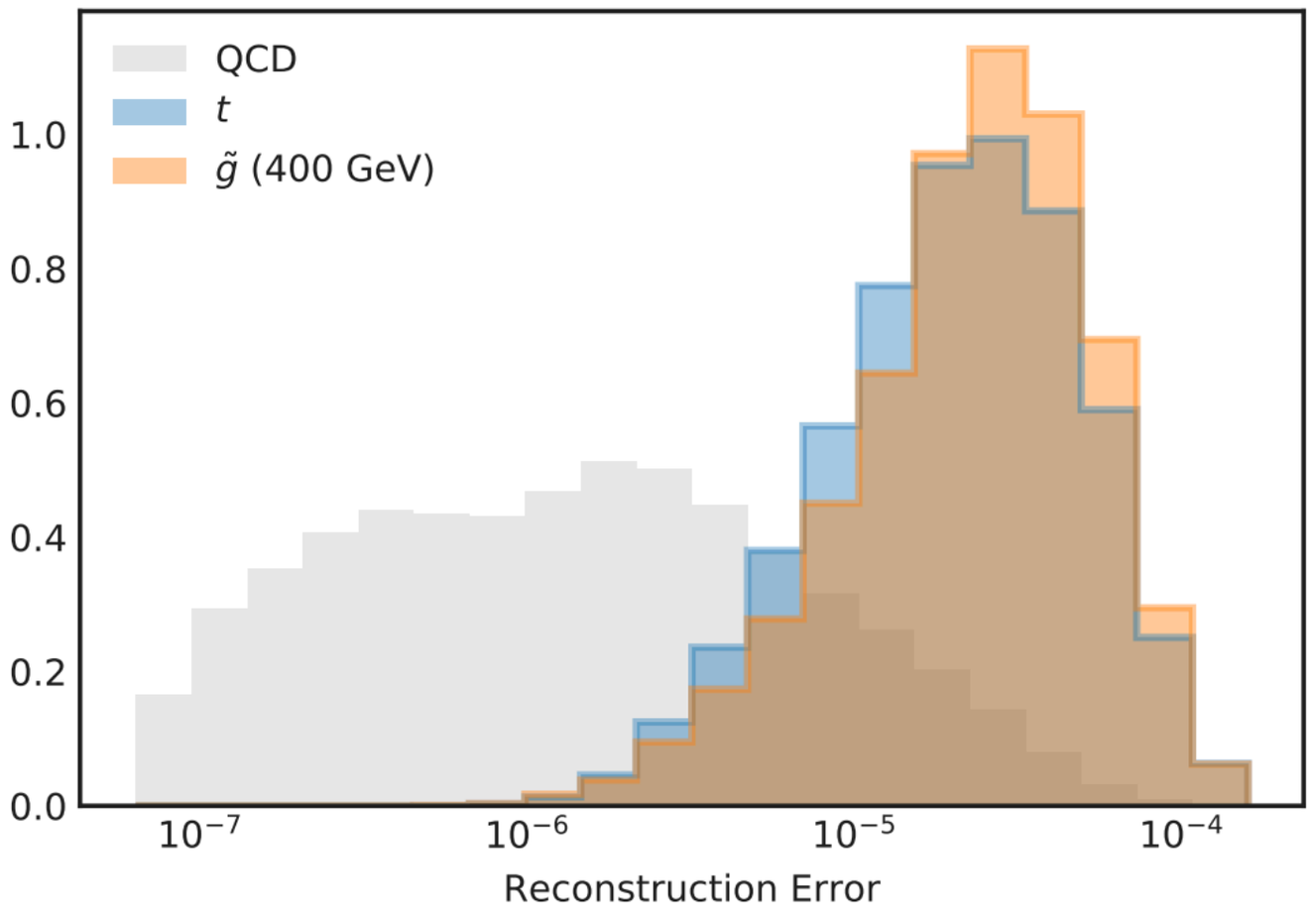


Figure 2: Distribution of reconstruction error computed with a CNN autoencoder on test samples of QCD background (gray) and two signals: tops (blue) and 400 GeV gluinos (orange).

Weak supervision / CWoLa

[1702.00414], [1706.09451], [1708.02949], [1801.10158], [1805.02664], [1902.02634]

Theorem 1 *Given mixed samples M_1 and M_2 defined in terms of pure samples S and B with signal fractions $f_1 > f_2$, an optimal classifier trained to distinguish M_1 from M_2 is also optimal for distinguishing S from B.*

Weak supervision / CWoLa

[1702.00414], [1706.09451], [1708.02949], [1801.10158], [1805.02664], [1902.02634]

Theorem 1 *Given mixed samples M_1 and M_2 defined in terms of pure samples S and B with signal fractions $f_1 > f_2$, an optimal classifier trained to distinguish M_1 from M_2 is also optimal for distinguishing S from B.*

Proof. The optimal classifier to distinguish examples drawn from p_{M_1} and p_{M_2} is the likelihood ratio $L_{M_1/M_2}(\vec{x}) = p_{M_1}(\vec{x})/p_{M_2}(\vec{x})$. Similarly, the optimal classifier to distinguish examples drawn from p_S and p_B is the likelihood ratio $L_{S/B}(\vec{x}) = p_S(\vec{x})/p_B(\vec{x})$. Where p_B has support, we can relate these two likelihood ratios algebraically:

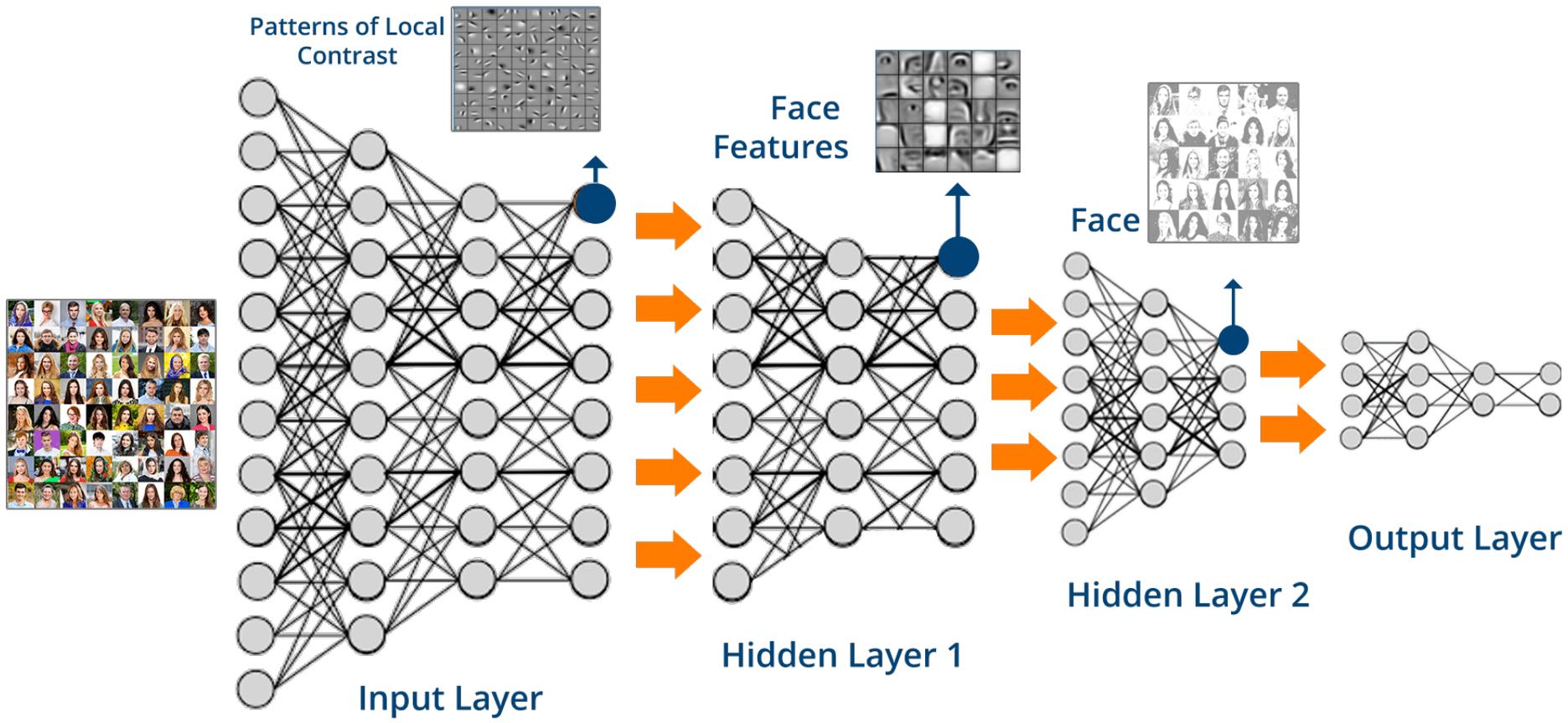
$$L_{M_1/M_2} = \frac{p_{M_1}}{p_{M_2}} = \frac{f_1\, p_S + (1-f_1)\, p_B}{f_2\, p_S + (1-f_2)\, p_B} = \frac{f_1\, L_{S/B} + (1-f_1)}{f_2\, L_{S/B} + (1-f_2)},$$

which is a monotonically increasing rescaling of the likelihood $L_{S/B}$ as long as $f_1 > f_2$, since $\partial_{L_{S/B}} L_{M_1/M_2} = (f_1 - f_2)/(f_2 L_{S/B} - f_2 + 1)^2 > 0$. If $f_1 < f_2$, then one obtains the reversed classifier. Therefore, $L_{S/B}$ and L_{M_1/M_2} define the same classifier. □

Transfer Learning

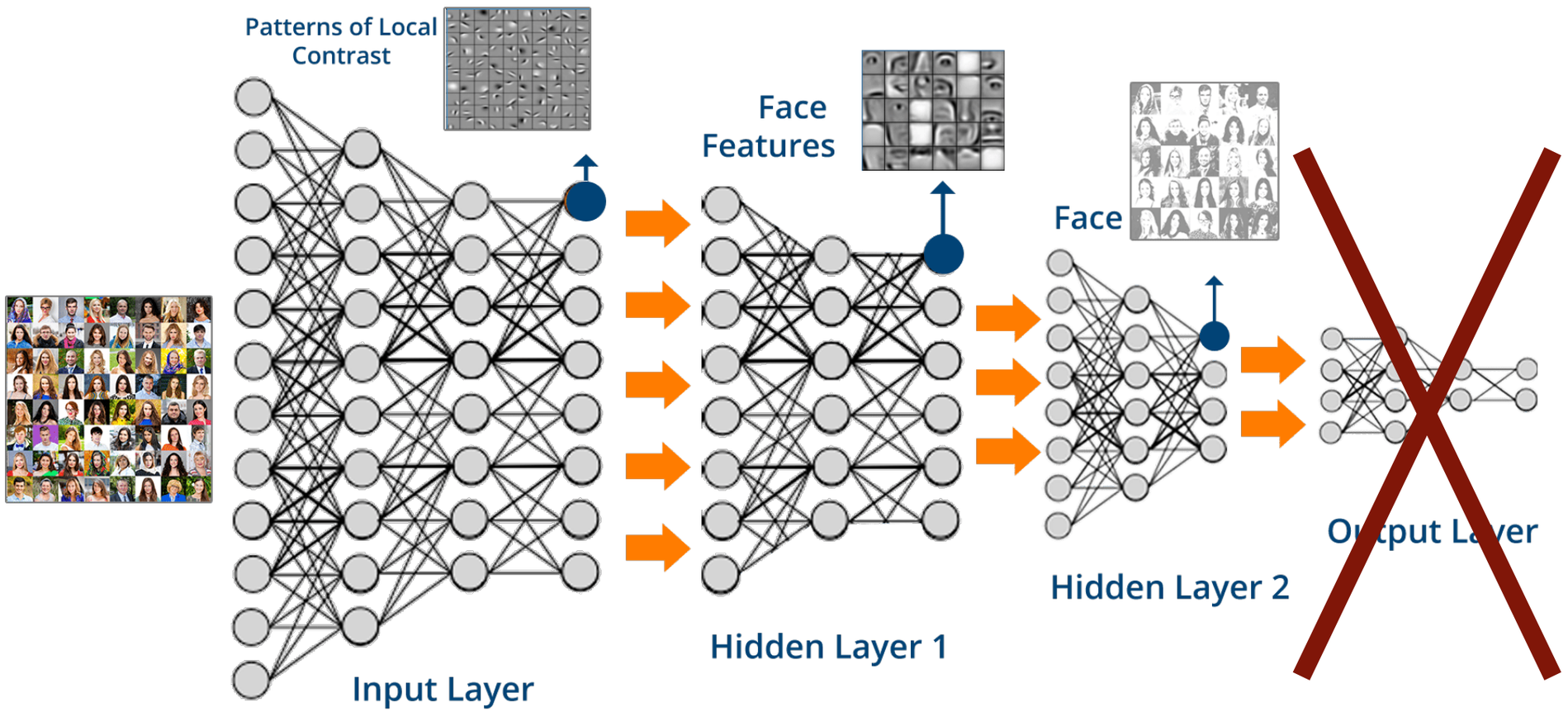
Pre-trained networks for image classification are readily available

https://keras.io/applications/



What if we want to do something with images, but not classification (using the same categories)?

Transfer Learning

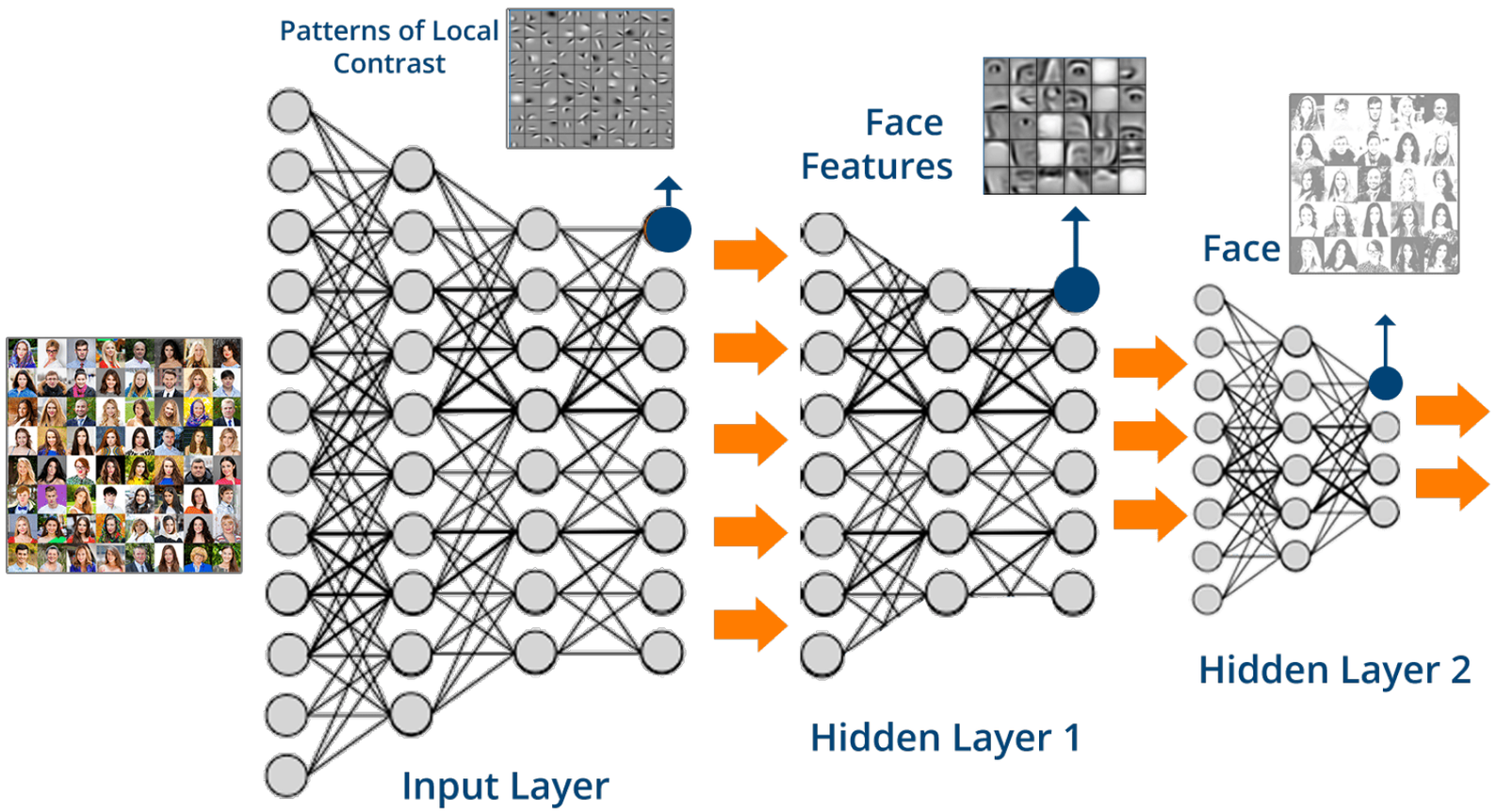


Keep the features

Get rid of the "classification part"

Freeze the layers so that the features don't change

Transfer Learning



"Preprocess" our data

New network

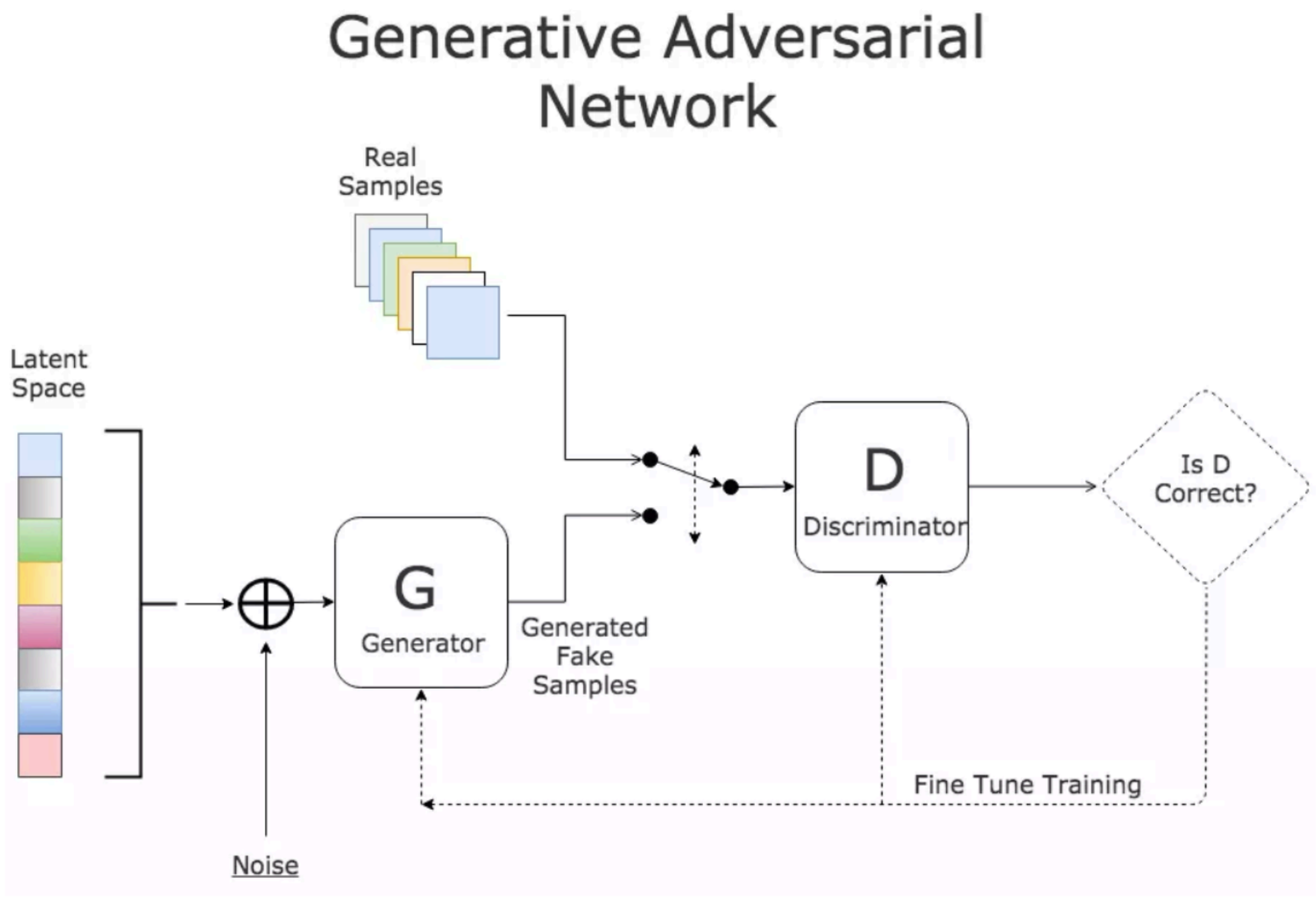
New network has small number of trainable weights, can train with limited resources

One more transition

Last section was about how to train on unlabeled data, but we still had classifying jets/events in our minds

Machine learning can be used for many other purposes in HEP

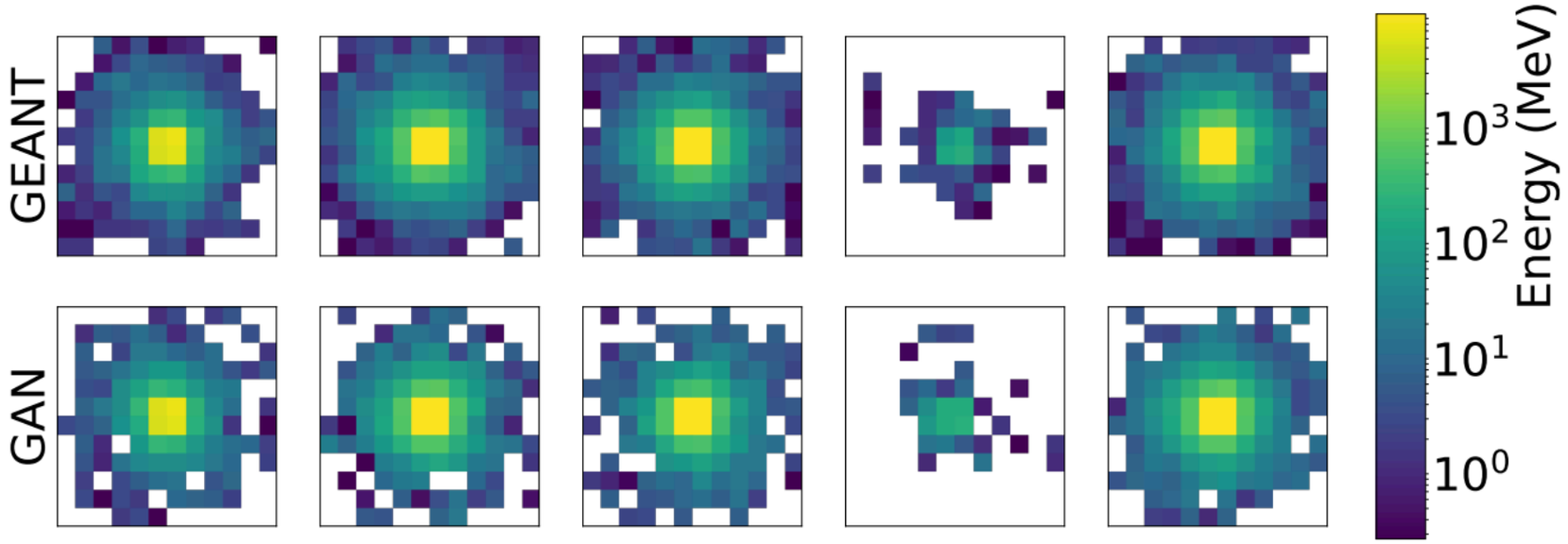
GANS



https://ddcolrs.wordpress.com/2017/07/03/generative-adversarial-networks-gan/

GANS

[1705.02355] and others: Fast detector simulation



[1810.11509], [1901.00875], [1907.03764] , and others: look at phase space integration / event generation

Conclusion

1. Neural networks are trained very similarly to fitting a line, define a loss and minimize it.

2. Deep learning can outperform high-level variables if the high-level information does not capture all of the information within the data

3. How the data is represented effects the style of network to use, and the results

4. Training can be done on real data (unlabeled)

5. Many more applications of machine learning

Updates to tutorial

Your results from the tutorial should be saved on your computer. To ensure that doing a git pull does not affect your answers, change the name of Tutorial_1.ipynb

I have added my solutions to the git lab repository, use

```
git pull
```

to download them to your computer

All of the required packages should now be in the docker container:

```
docker pull cteqschool/tutorial:mltools-2.0.0
```