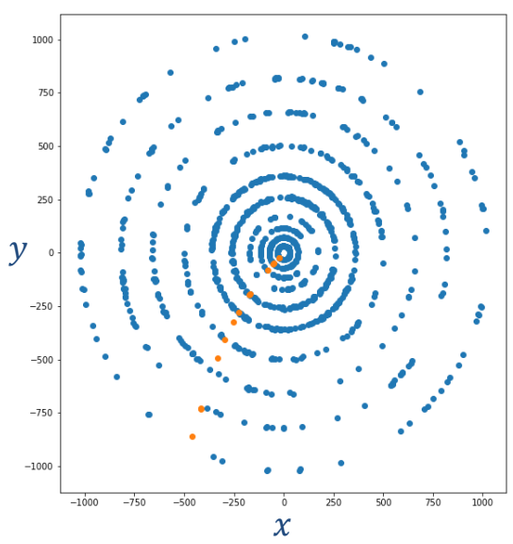
Fermilab U.S. DEPARTMENT OF ENERGY Office of Science

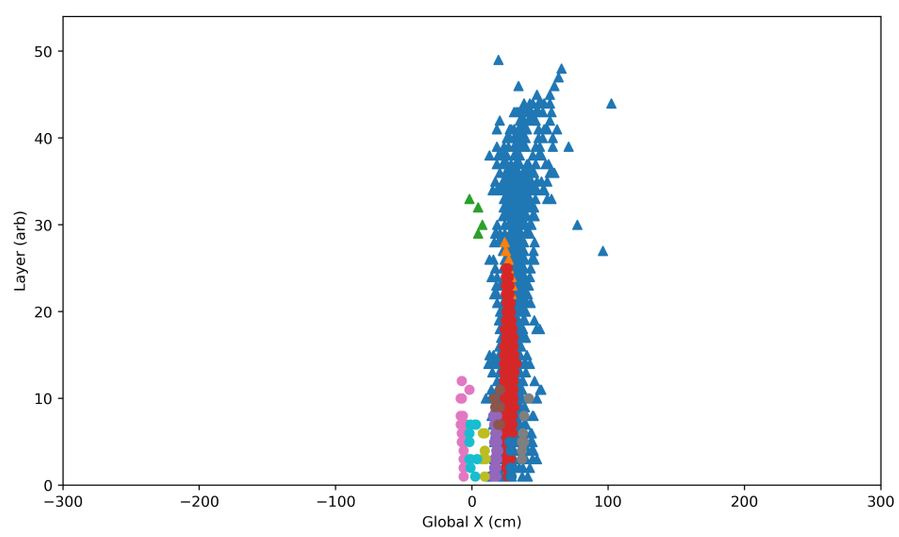
Introduction to Machine Learning in HEP

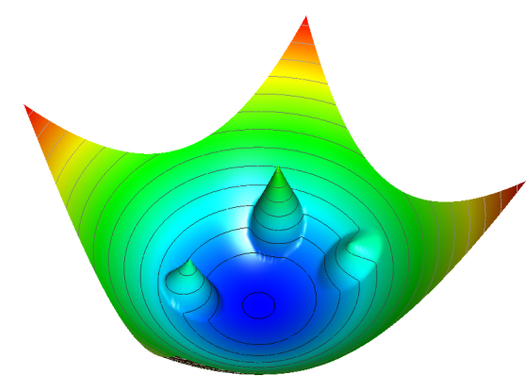
Lindsey Gray

HCPSS 2022

22 August 2022





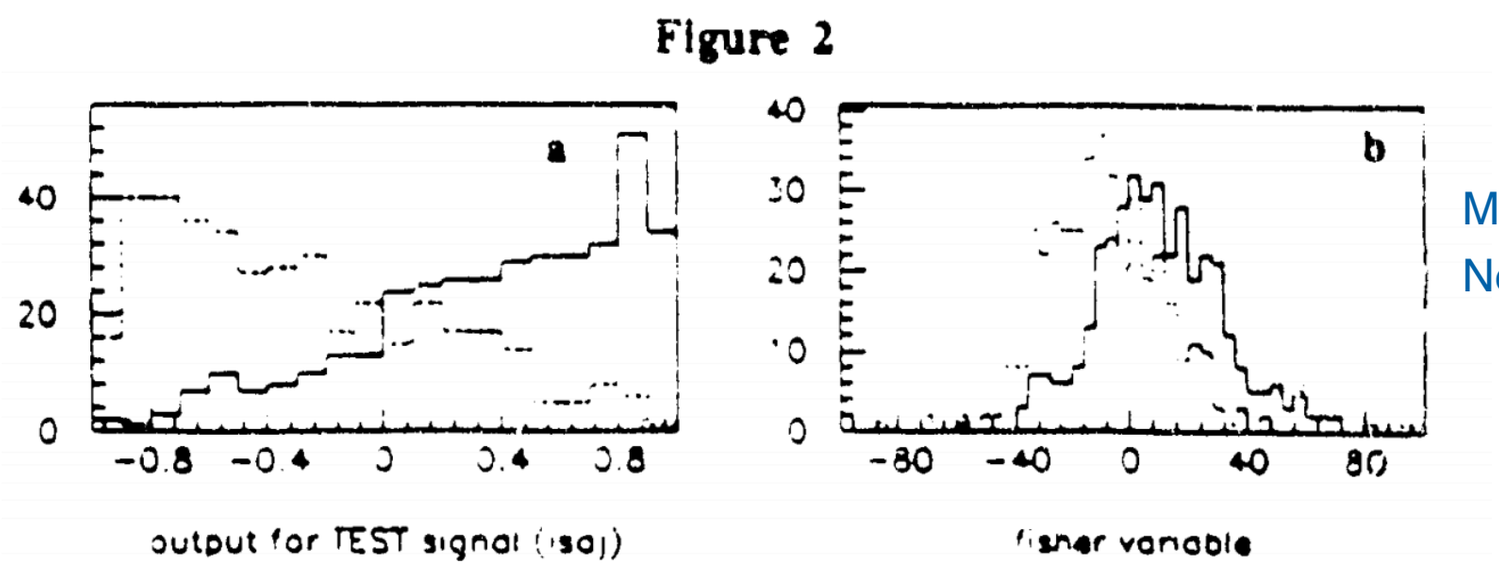


How did we get to where we are going?

- The detectors and challenges, and the tools to address them are the result of a long story in particle physics
  - We always want better discriminators that utilize more information

- HEP Physicists have to demonstrate control over methodologies
  - We can't just separate categories of data from one another
  - Error models and confidence regions are required in order to report our results

- Using ML techniques effectively within HEP is the result of decades of accumulated knowledge



Multijet Top discriminant from 1992
Neural Network (left), Fisher discriminant (right)

https://www.osti.gov/biblio/10110749

ML and Neural Networks in a Nutshell

- Goal: find the parameters w of a function F that best maps x onto y
  - Do this by minimizing a chosen “loss function”
  - ML is the set of numerical algorithms that solve this problem

- Neural Networks are a subset of these algorithms that are defined recursively from inputs to outputs
  - Any mapping function F can be approximated by a sufficiently large NN
- Since inputs are variable and relationship to output is learn, including new information is very straightforward, as is scaling up computation to more data

particle four-momenta

signal or background?

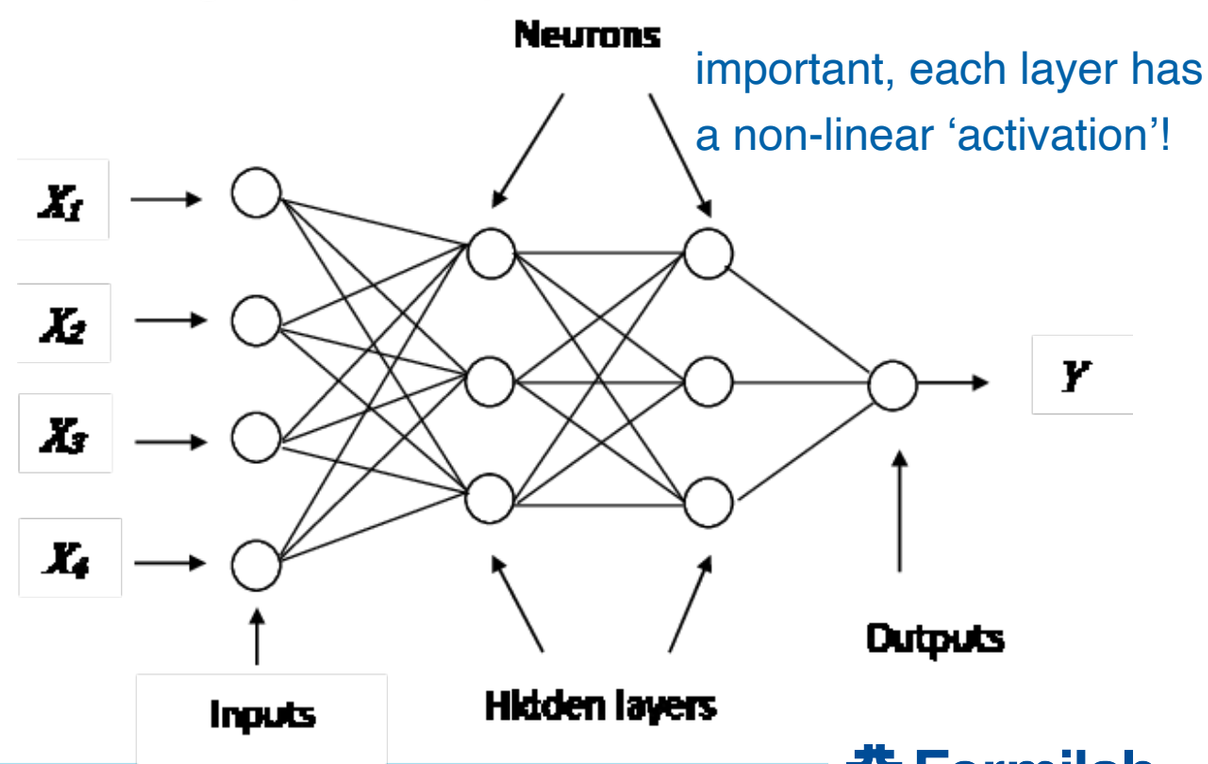
$$\vec{x} \qquad \vec{y} = F_{\vec{w}}(\vec{x}) \qquad \vec{y}$$

input space

mapping function

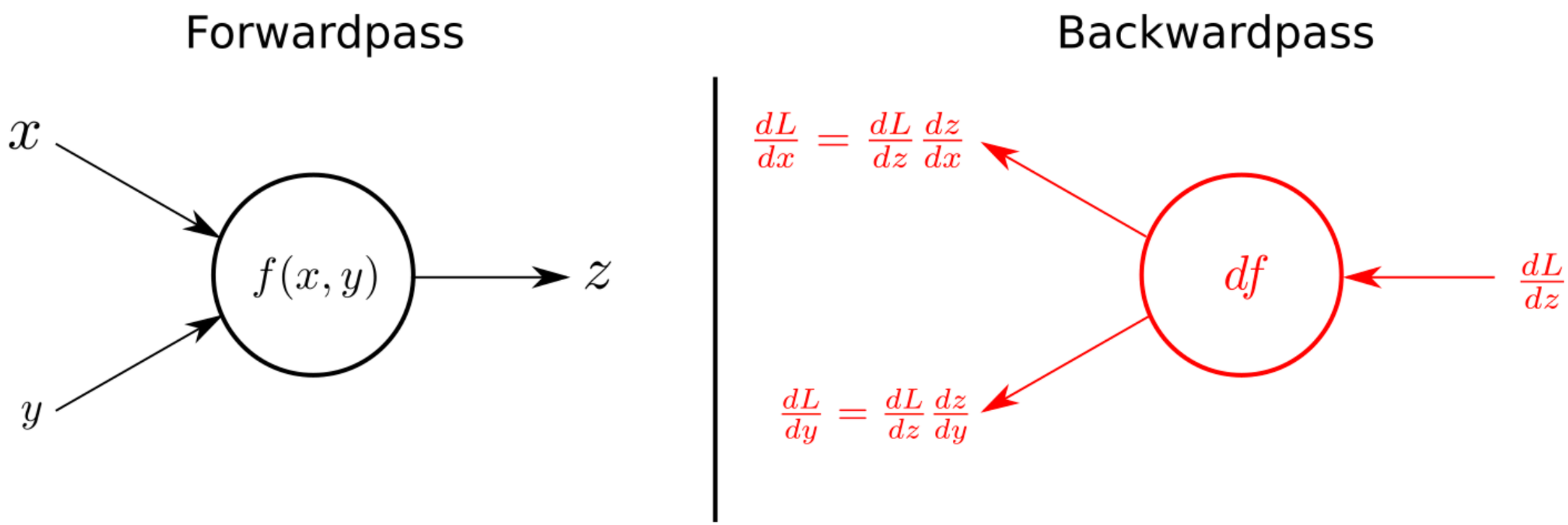
weight space

output space



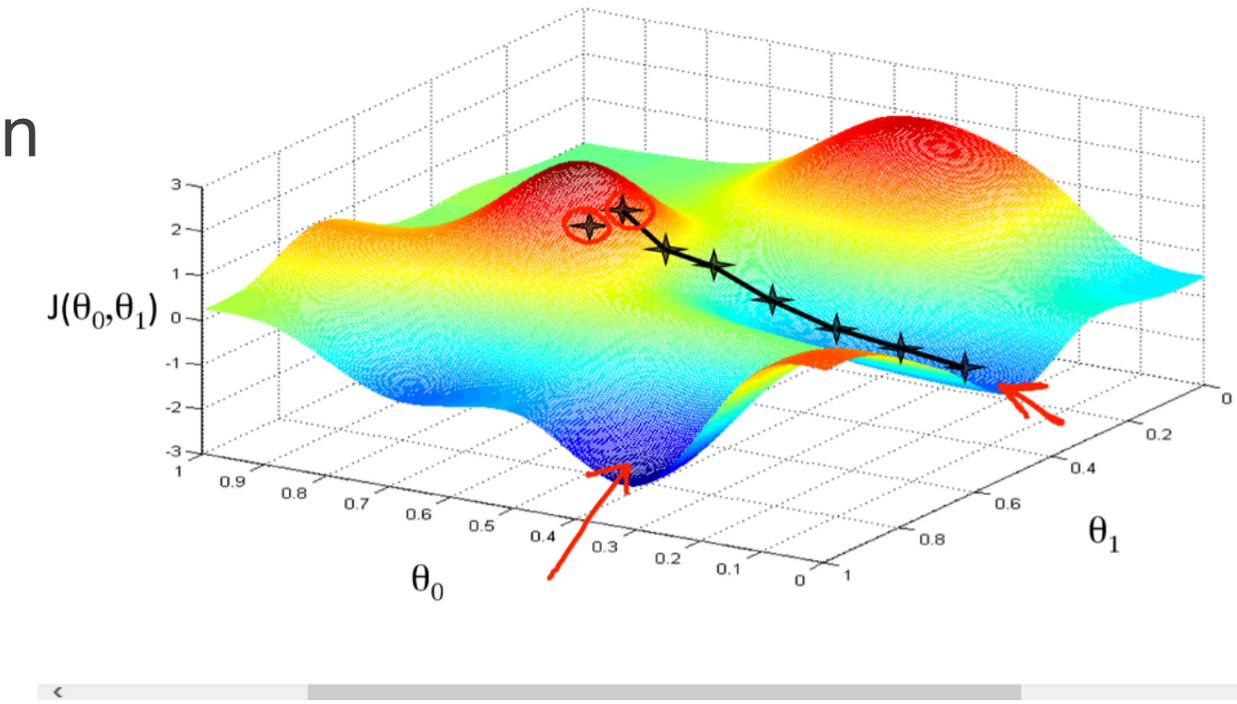
Backpropagation and training of neural networks

- Each node in a network makes some prediction
  - Each parameter's derivative can be calculated using the chain rule
- Encode task to be done in a 'loss' function and minimize that loss
  - Recursively update parameters based on estimated error at each pass over the data



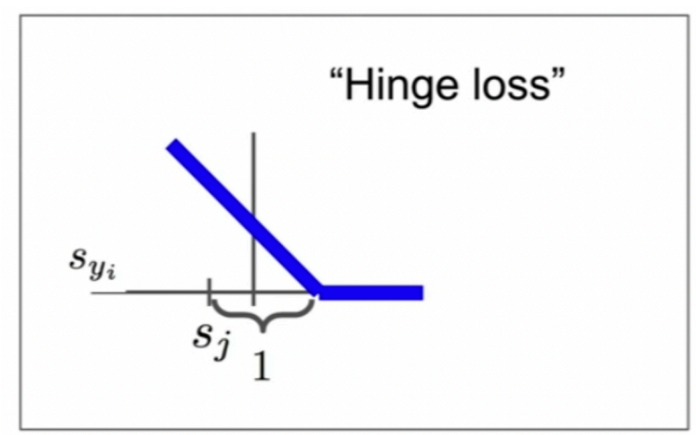
Training of neural networks

- Encode task to be done in a ‘loss’ function and minimize that loss
  - Recursively update parameters based on estimated error at each pass over the data



Some example loss functions:

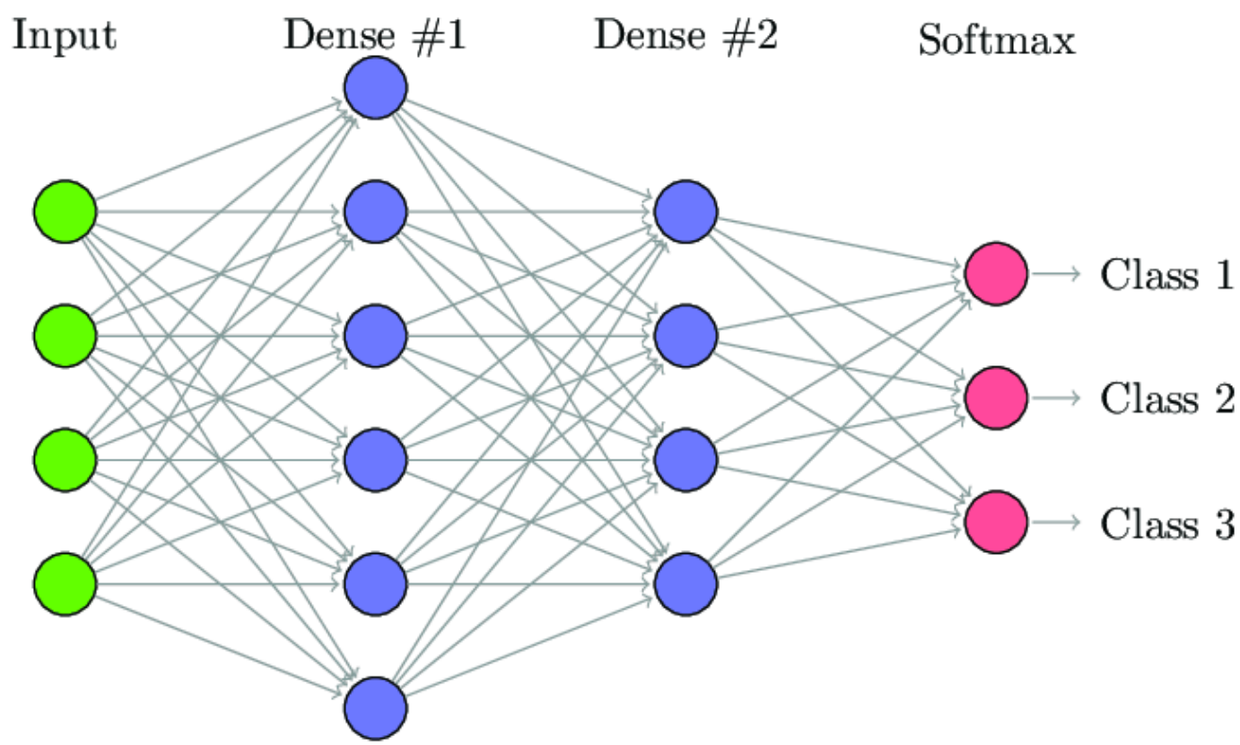
$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$



$$L_i = \sum_{j \neq y_i} \begin{cases} 0 & \text{if } s_{y_i} \geq s_j + 1 \\ s_j - s_{y_i} + 1 & \text{otherwise} \end{cases}$$

We’ll go into details after we discuss NNs a bit more

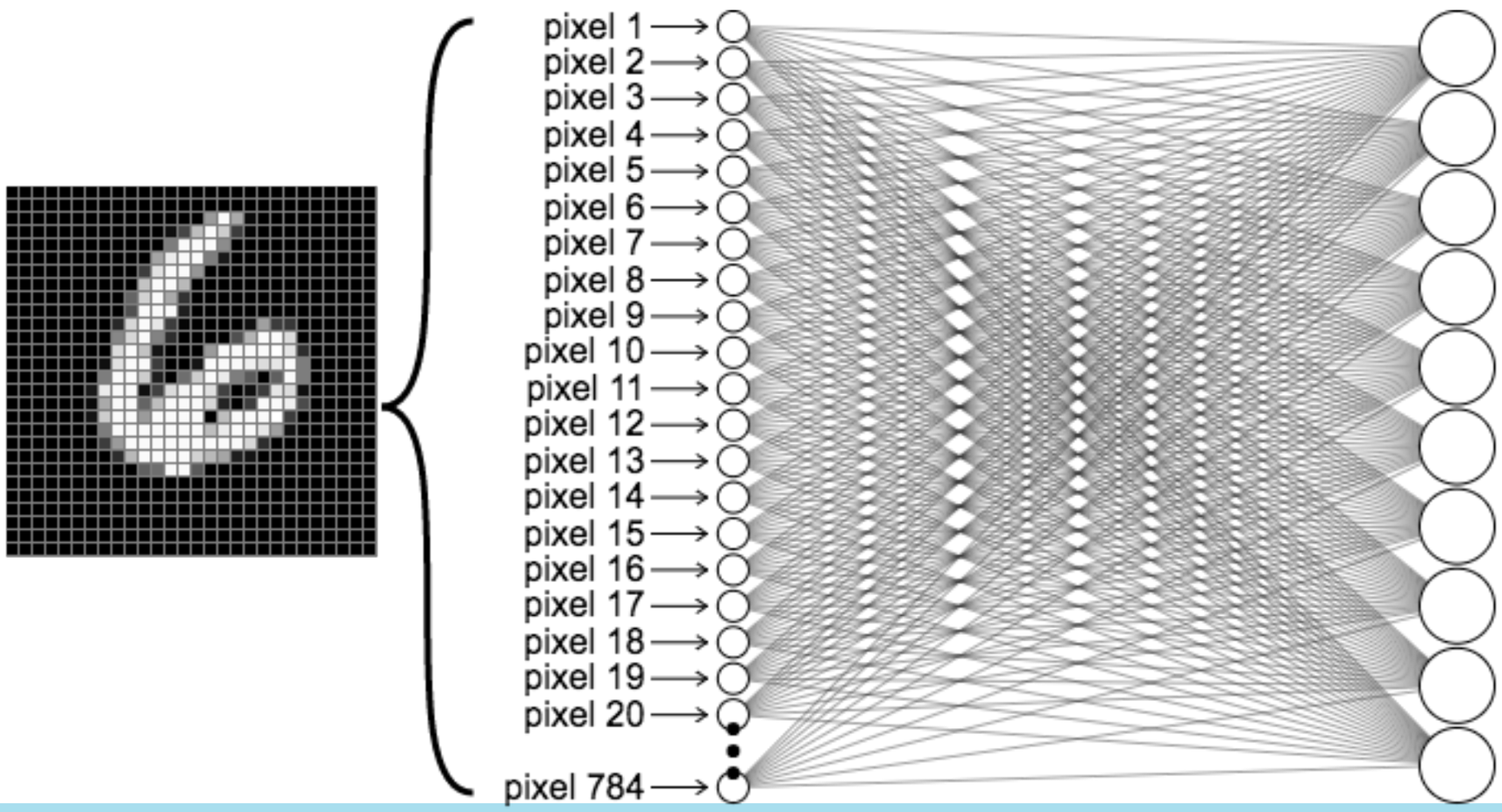
Kinds of Neural Networks: Fully Connected Neural Networks



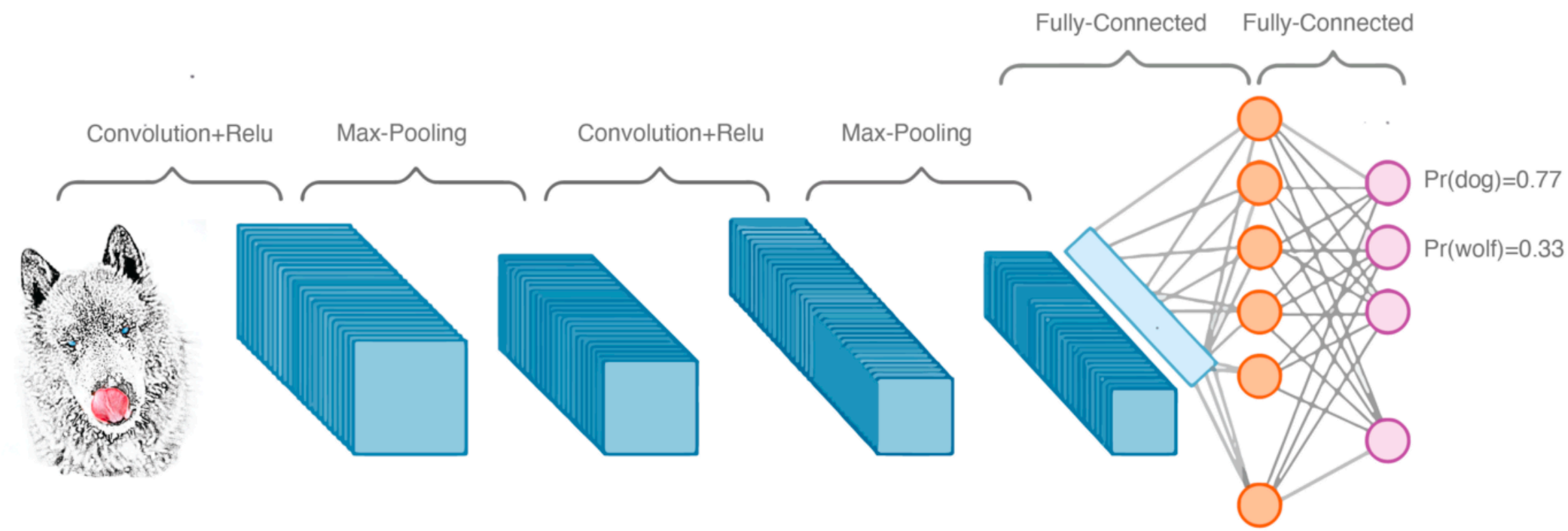
- Fully connected networks - also called “multi-layer perceptrons” (MLP)
  - Basic building block of nearly all network networks
  - Are able to approximate a wide degree of functions when sufficiently layered (deep) or parameterized (wide)
- The number of inputs and outputs for a given observation are fixed
  - Deals well with situations where only a specific set of variables contain meaning but a combination could be better exploited
- Makes no assumption about the symmetries in the data

Kinds of Neural Networks: Fully Connected Neural Networks

- While this neural network can be a powerful tool it can reach some limitations when it comes to images
  - In order to classify the number 6 below correctly, it would need many translated examples of 6 across the image input domain so that dependence on input position is not a discriminating feature
  - Compounding this across all possible inputs means that this network requires an enormous number of examples to avoid over-specialization



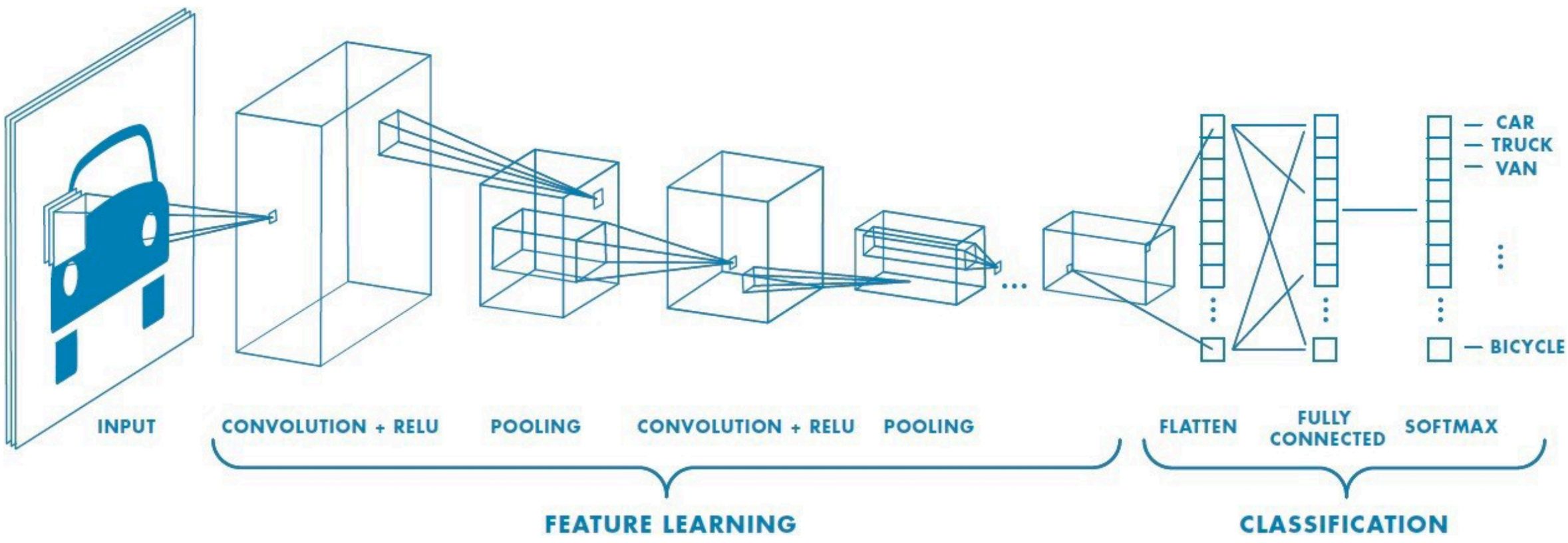
Kinds of Neural Networks: Convolutional Neural Networks



- Dealing with images, if we do not want to create the features ourselves is better dealt with by exploiting the translationally invariant nature of images
  - A number or object is still that same object regardless of its orientation within that image
- Create a new kind of neural network component that applies the *same* filter to subsections of the image
  - Moving that window for the filter over every (group of) pixel(s) in the image
  - This forces translational invariance while still allowing the network to optimize discriminating features

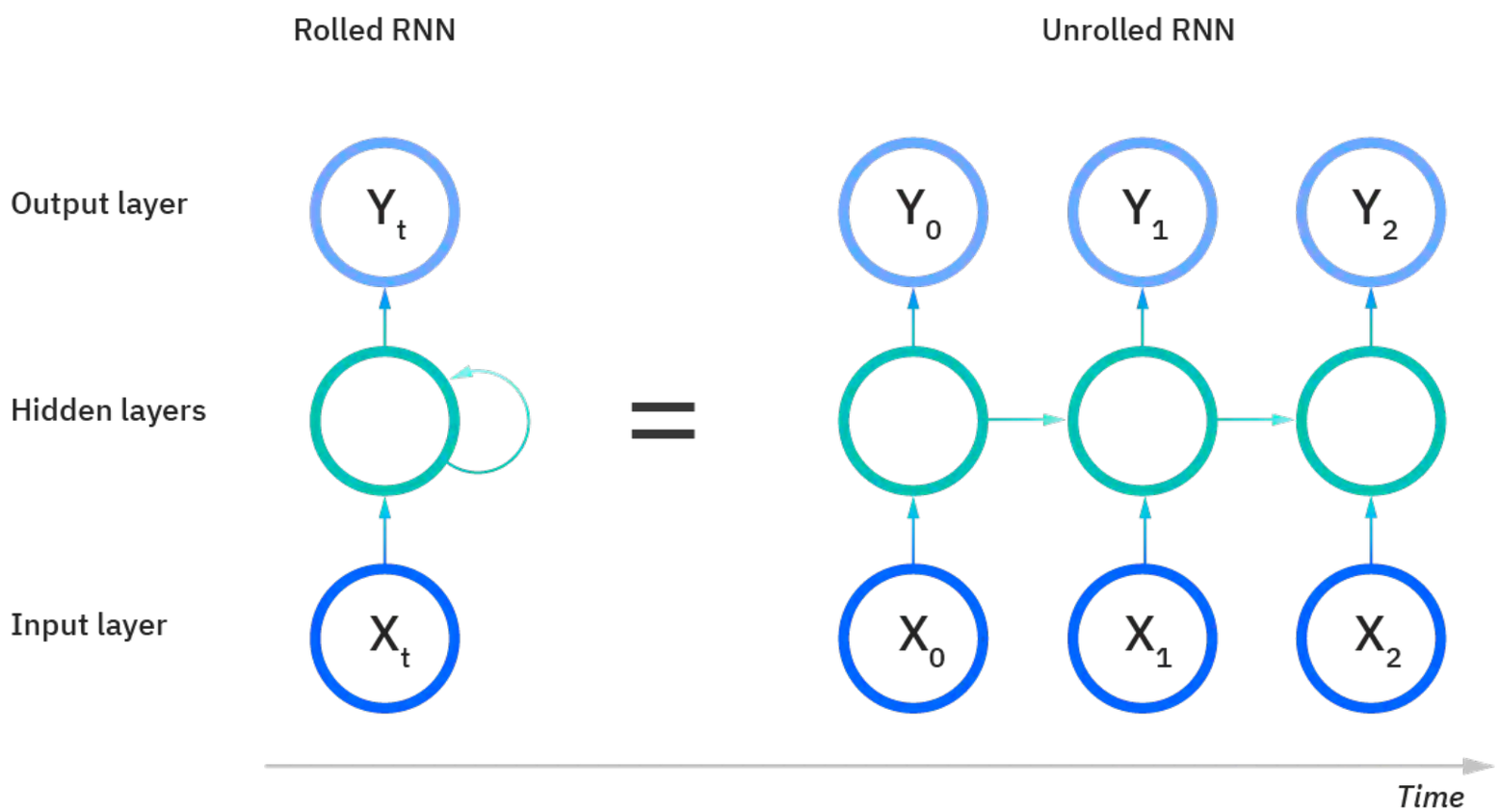
Kinds of Neural Networks: Convolutional Neural Networks

- By creating more and more distilled and coarse translationally invariant features of the network we can arrive at list of high-level abstract but guaranteed to be translationally invariant features about an image
  - This ability to engineer features with desired symmetries vastly improved the quality of handwritten character recognition, the benchmark for ML at the time
  - Moreover, it made large scale data driven image classification as it is known today possible

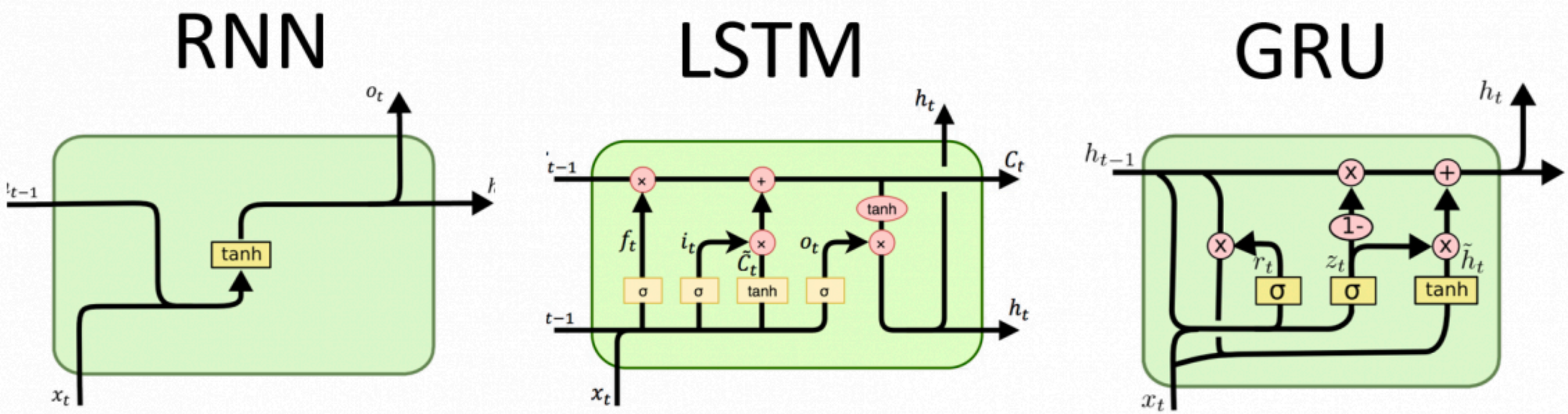


Kinds of Neural Networks: Recurrent Neural Networks

- Recurrent neural networks are especially adapted to series data like measurements of a system at various points in time
  - This is achieved by updating each piece of input data to the *same* neural network, updated by the output of the previously seen data
  - This means that correlated



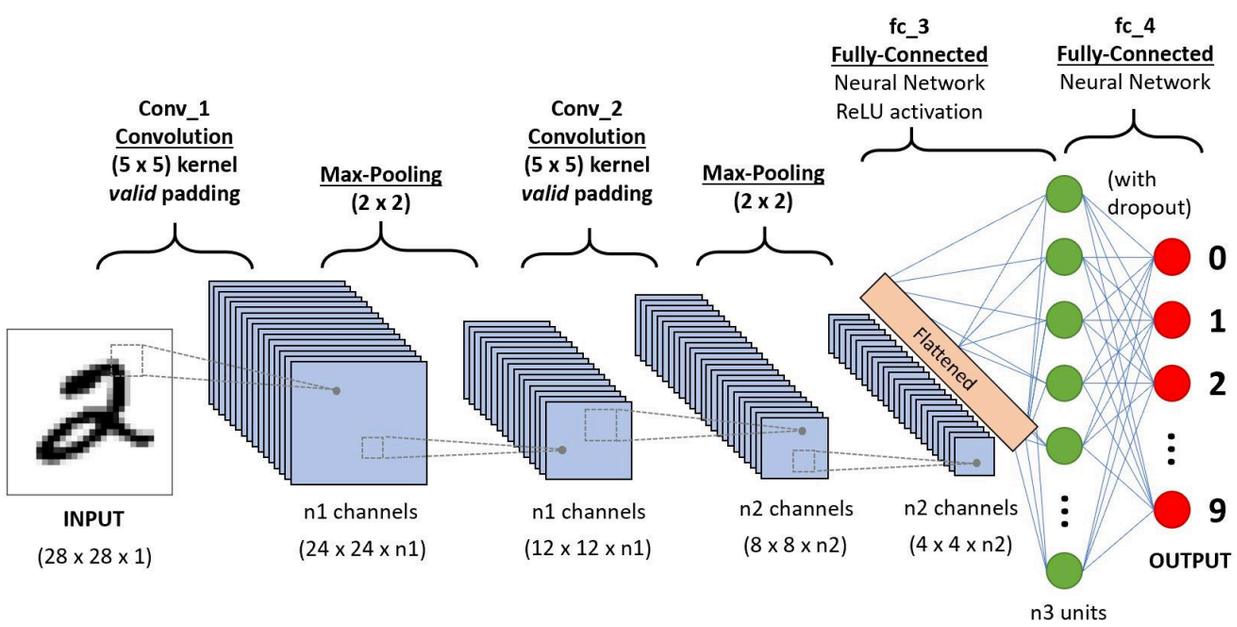
Kinds of Neural Networks: Recurrent Neural Networks

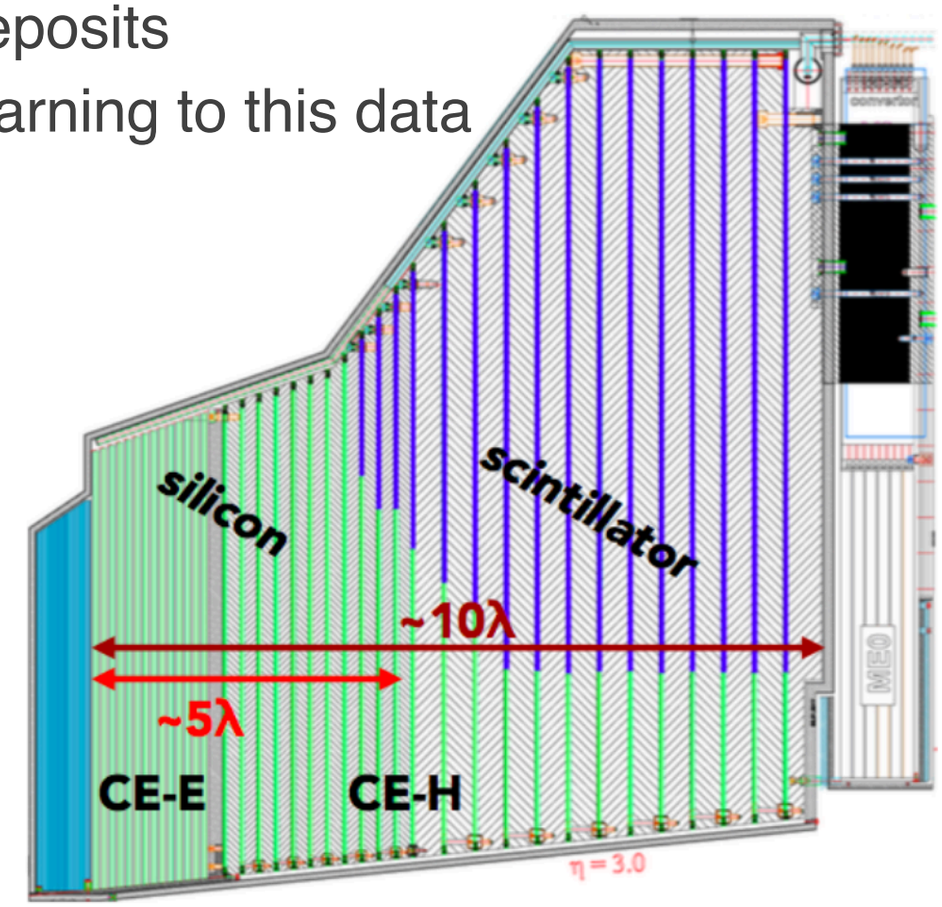


- RNN - recurrent neural network with no hidden state (memory)
- LSTM - long short term memory with hidden state using “forget” gates that modulate the expression of features in its hidden state, allowing the unit to maintain proper responses even for rare occurrences
- GRU - Gated recurrent unit a simpler form of an LSTM that does not use information about the unit’s output to govern its internal state

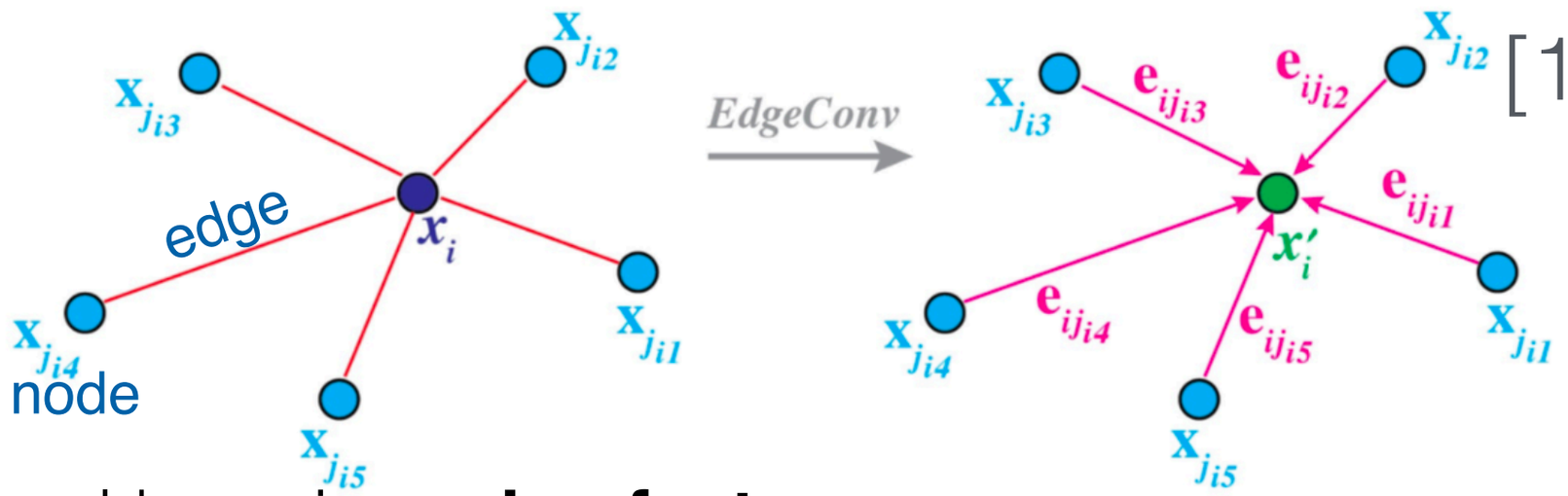
Exploiting granular information with machine learning

- Modern machine learning can determine important discriminating information in the course of training if the input 'shape' is fixed
  - Using convolutional neural networks for example, images are given as-is for training examples, discriminating features encoded in filters and high-dimensional 'latent spaces'
- However, many next generation particle physics detectors have irregular geometries with zero-suppressed outputs
  - Varying material with sparse sampling of energy deposits
  - Requires different approaches to apply machine learning to this data





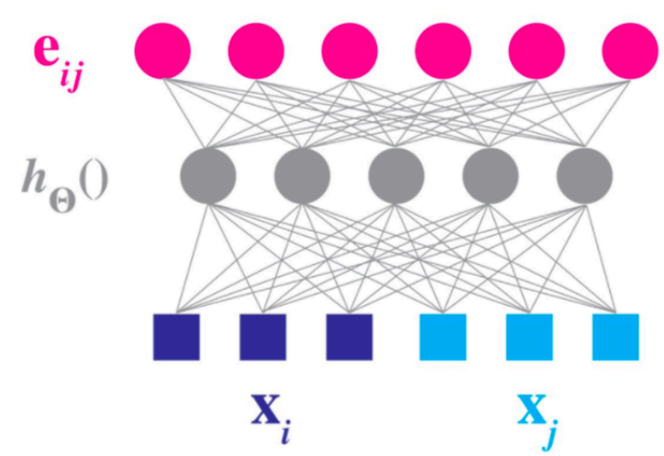
Graph Neural Networks: Edge Convolution



[1801.07829]

- Update $x_i \rightarrow x_i'$ by using **edge features**
  - i.e. learned features of the edges that connects x_i with its neighbors
  - Still independent of ordering of points, but uses **local geometry**
  - '**Convolutional**' as the operation is applied point by point to obtain **x**'
- These edge features and aggregation steps mimic the functionality of loops with if-statements in them (i.e. hand-written pattern recognition)

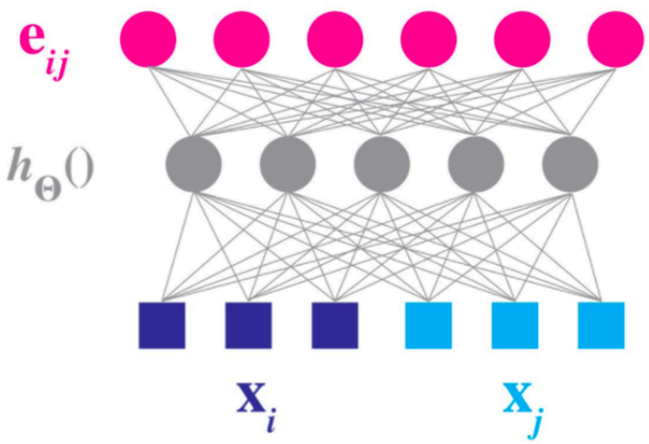
$$\mathbf{x}_i' = \underset{j:(i,j)\in\mathcal{E}}{\square} h_\Theta(\mathbf{x}_i, \mathbf{x}_j)$$



Can set dim of edge feature vector

Graph Neural Networks: Dynamic Graph Convolutions

[1801.07829]



$$\mathbf{x}_i' = \square_{j:(i,j)\in\mathcal{E}} h_\Theta(\mathbf{x}_i, \mathbf{x}_j)$$

$$h_\Theta(\mathbf{x}_i, \mathbf{x}_j) = h_\Theta(\mathbf{x}_i)$$ No neighborhood info (only global)

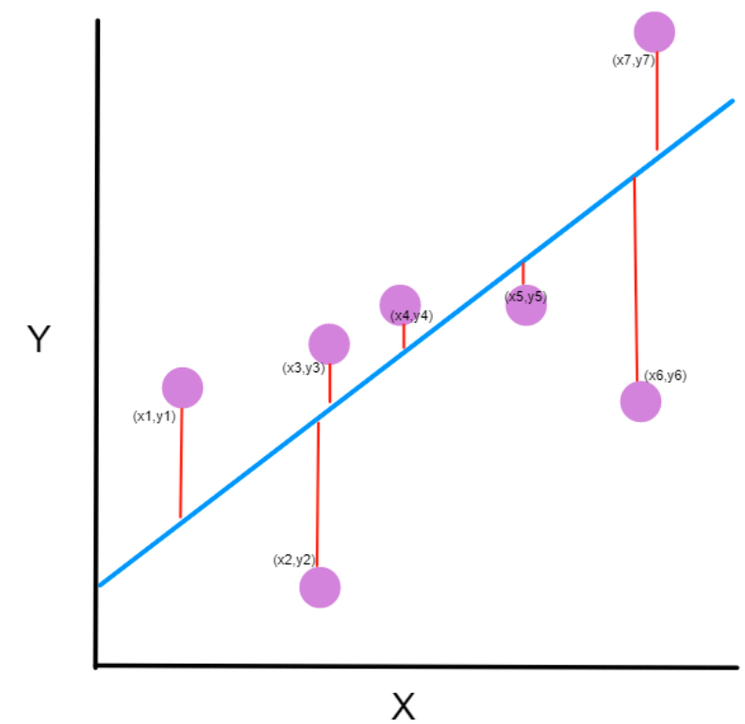
$$h_\Theta(\mathbf{x}_i, \mathbf{x}_j) = h_\Theta(\mathbf{x}_j - \mathbf{x}_i)$$ Only local information

$$h_\Theta(\mathbf{x}_i, \mathbf{x}_j) = \bar{h}_\Theta(\mathbf{x}_i, \mathbf{x}_j - \mathbf{x}_i)$$ Combination of both

- **Dynamic:** Redo kNN after every update
  - The connectivity matrix changes after every update

Losses Using the Deviation from the Mean

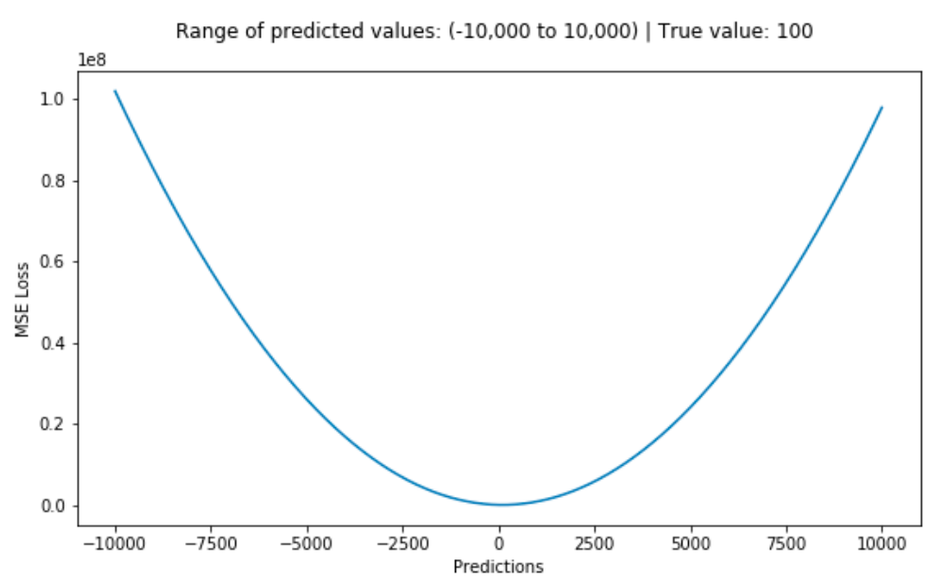
- A loss function encodes how much it "costs" to make an incorrect prediction
  - Usually this involves placing an object in a category and you want a function which grows if the prediction is incorrect enough
  - Therefore you can use gradient descent algorithms to minimize the cost
  - All the "work" comes from finding that function!
- Choosing the wrong function can really impact effectiveness

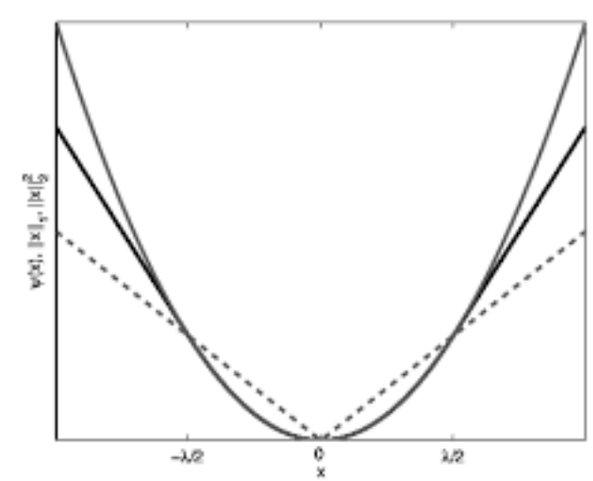


$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

$$Huber = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{2}(y_i - \hat{y}_i)^2 \qquad |y_i - \hat{y}_i| \le \delta$$

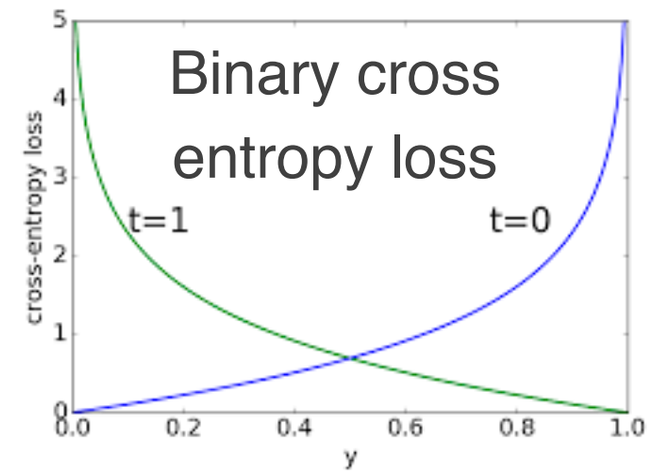
$$Huber = \frac{1}{n}\sum_{i=1}^{n}\delta\left(|y_i - \hat{y}_i| - \frac{1}{2}\delta\right) \qquad |y_i - \hat{y}_i| > \delta$$





Probabilistic Categorization Losses

$$L = -\frac{1}{N}\left[\sum_{j=1}^{N}\left[t_j \log(p_j) + (1-t_j)\log(1-p_j)\right]\right]$$



- "Cross entropy loss" - maximize the probability you label something correctly
- In particular the cross-entropy losses are directly related to maximum likelihood estimators
  - Since in this case we are estimate the probability that an input is a given category directly
  - We are "fitting"/predicting the probabilities directly and simply want to predict the categorization rather than estimating a parameter value

The "softmax" function:

$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

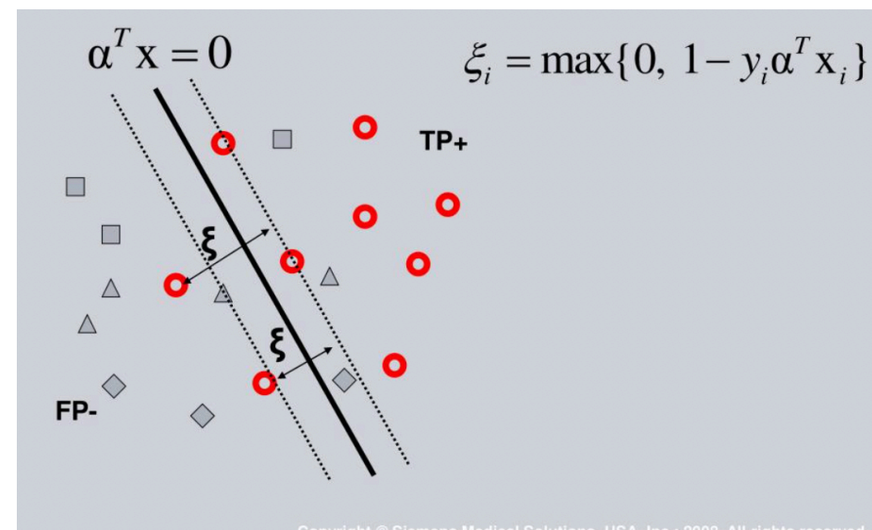
Extension to multiple categories:

$$\text{cross-entropy} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{k} t_{i,j}\log(p_{i,j})$$

Embedding Categorization Losses

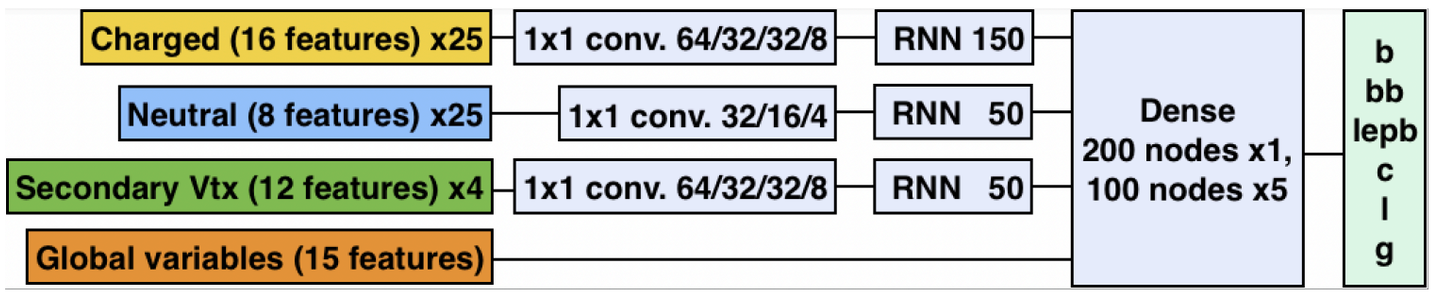
- "Hinge loss" - manipulate coordinates so that things in the same category are near each other
  - These losses are useful if you want to try to separate populations from each other and predict if a newly observed piece of data belongs in a given population.

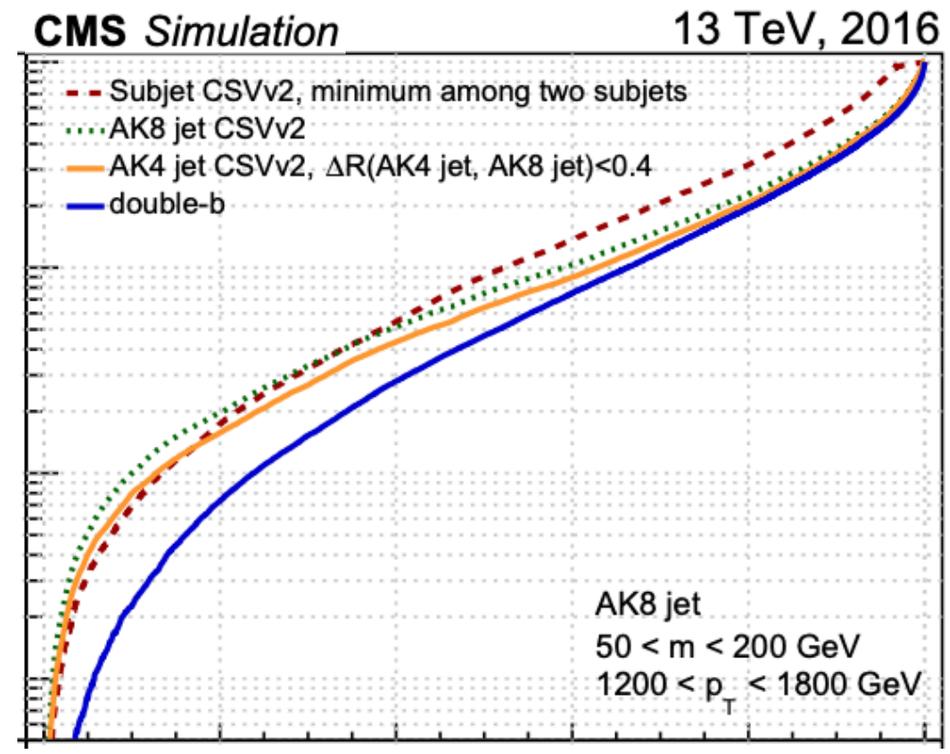
$$l_n = \begin{cases} x_n, & \text{if } y_n = 1, \\ \max\{0, \Delta - x_n\}, & \text{if } y_n = -1, \end{cases}$$



- In particular:
  - This has been used heavily in helping to create "embeddings" that put words of similar meaning near each other
  - More abstractly this loss function is very useful for making clustering problems simpler so that non-bespoke algorithms can help you categorize your data further

Examples in HEP: Jet Tagging with Expert Features



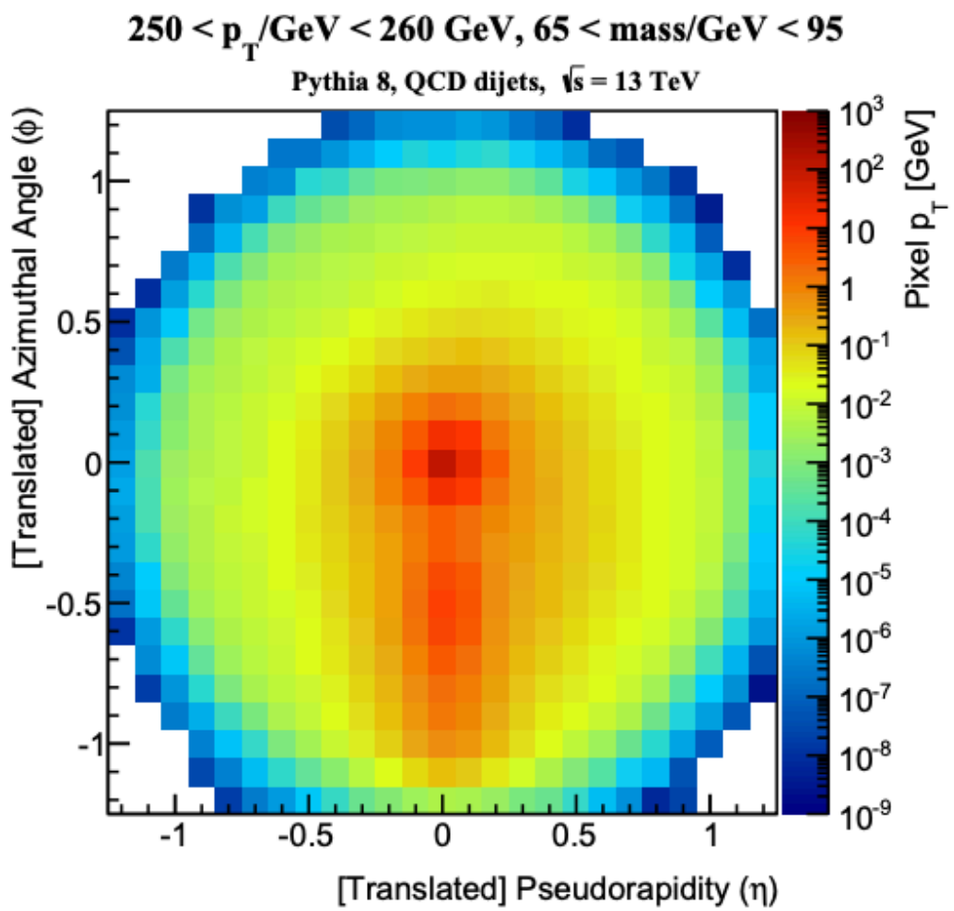


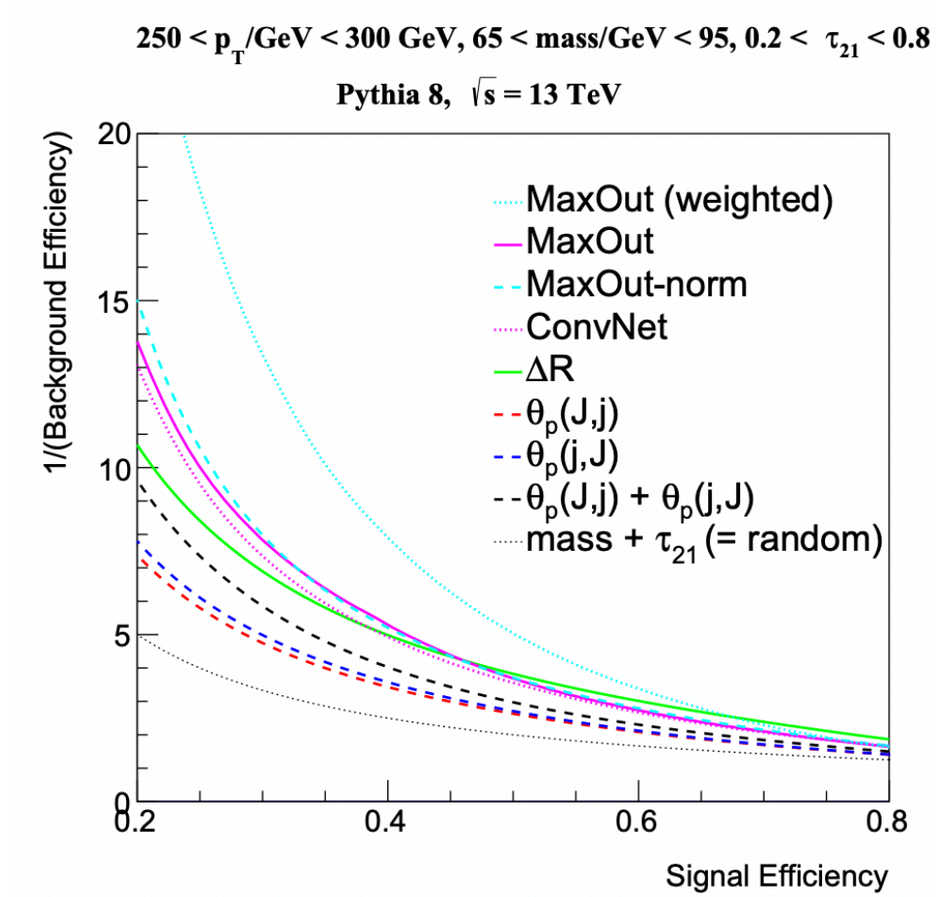
https://arxiv.org/abs/1712.07158

- Given the complex nature of jets, ML techniques have been commonplace
  - Every particle in a jet has some information about that jet's nature
- Common uses in b-tagging, and more recently merged-jet tagging
  - Evolution from using the original jet clustering very rigidly to allowing the ML algorithm to pick out what relationships are interesting
  - Only possible with modern ML techniques like deep networks or CNNs

Examples in HEP: Jet tagging and Jets as images

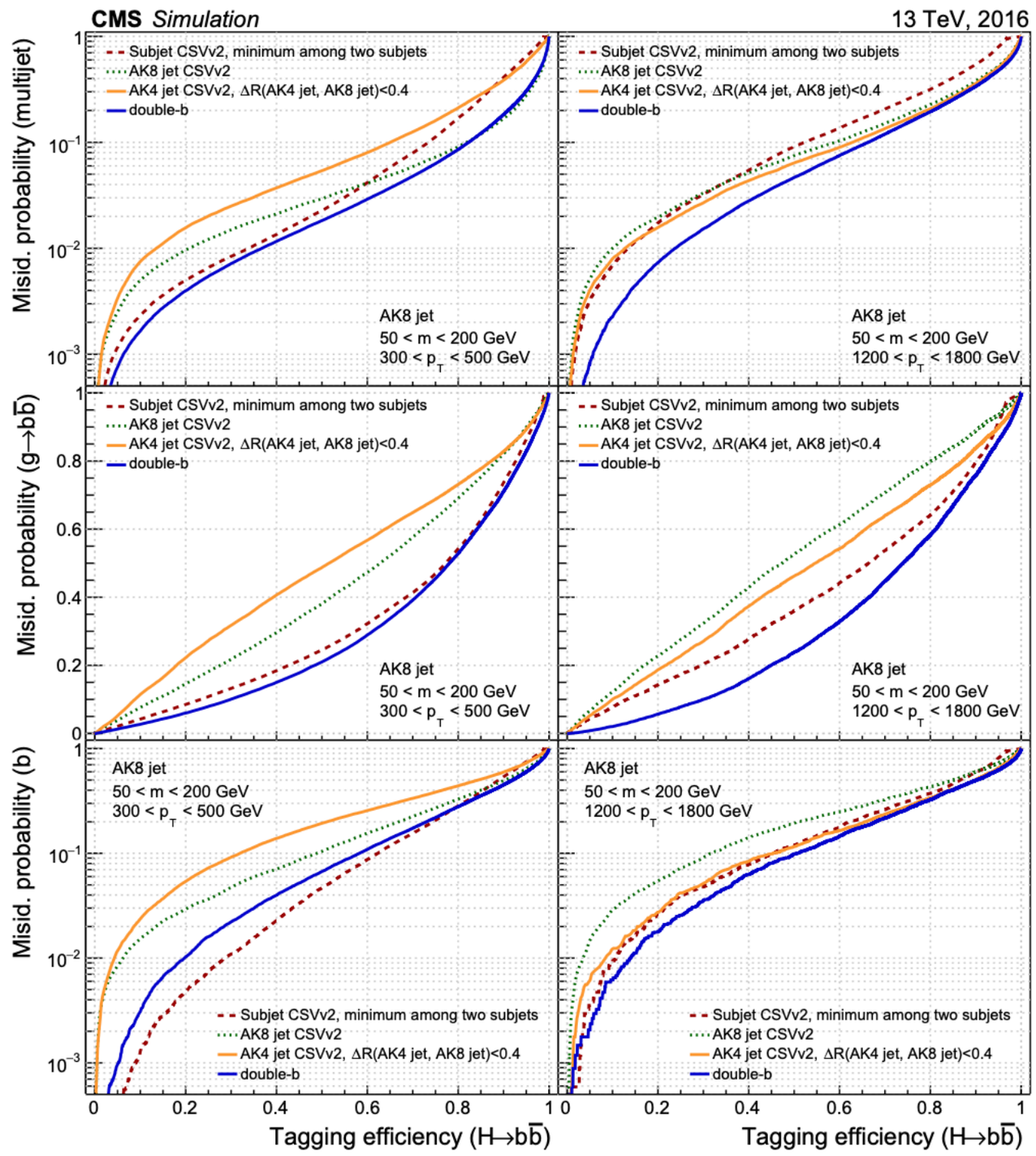
https://arxiv.org/abs/1511.05190





- Given that it is possible to display all the particles in a jet in a 2D image people have also used
  - Even for something as abstract as a jet, the convolutional neural network can do better than using a strict list of input features

Full results for deep double b-tag

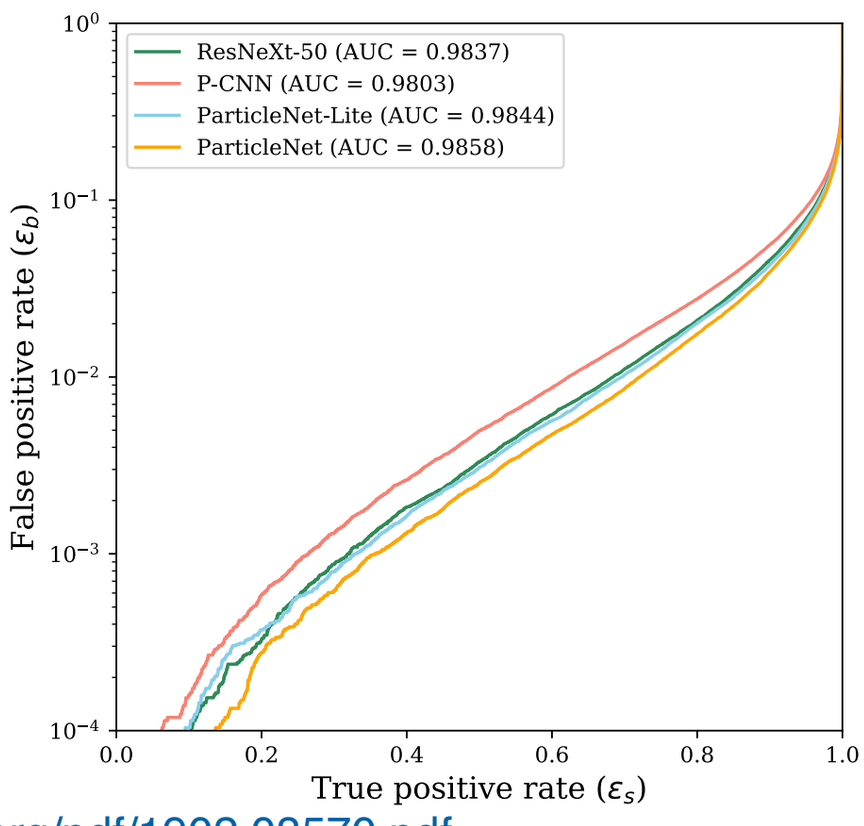


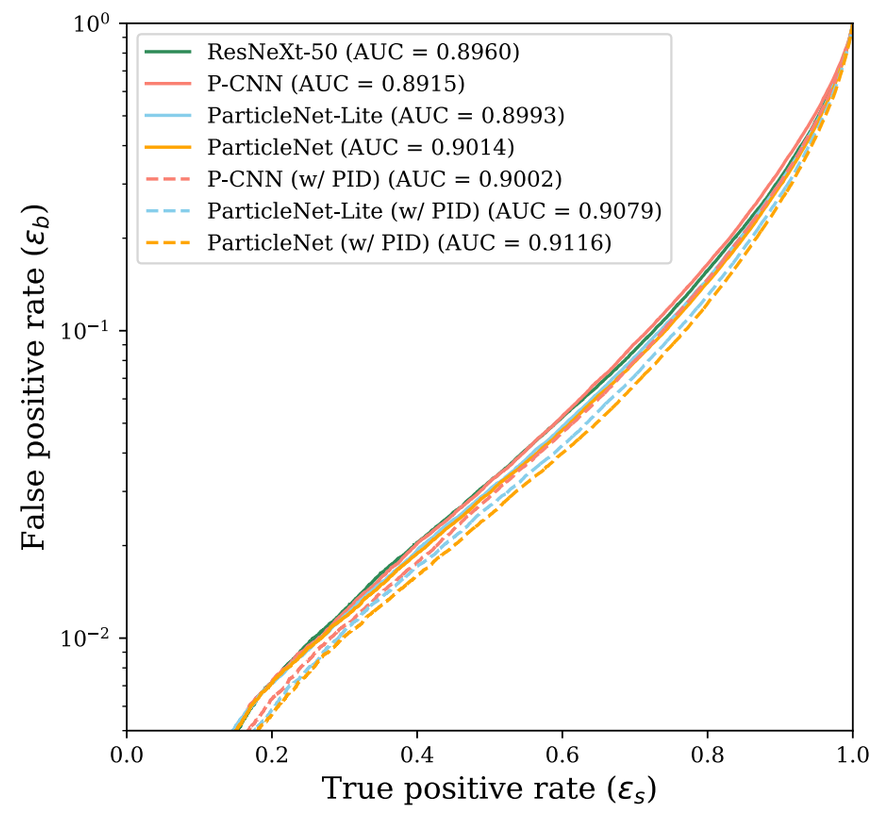
Jet tagging using graphs

- Below are results for using a graph to describe associations in jets
  - Improved performance with respect to image-based architectures!

- Train the classifier to learn what connections between particles are important

- Less need to pre-process the input properties for classification
  - Little to no transformation of data about jet constituents





https://arxiv.org/pdf/1902.08570.pdf

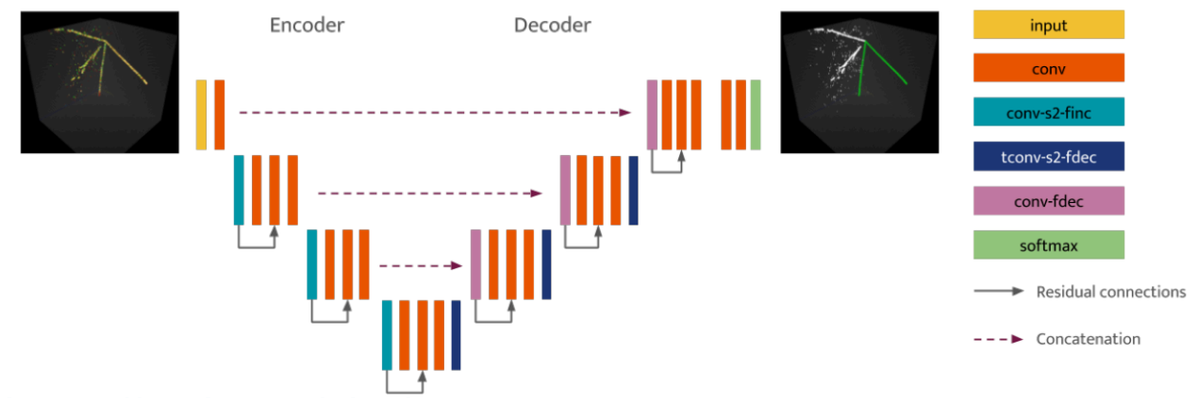
Dealing with extremely large and/or sparse data

- LArTPCs and current calorimeters are being used to test ML based reconstruction as well
  - Early successes for convolutional based approaches

- ML based approaches starting to take the lead in neutrino physics

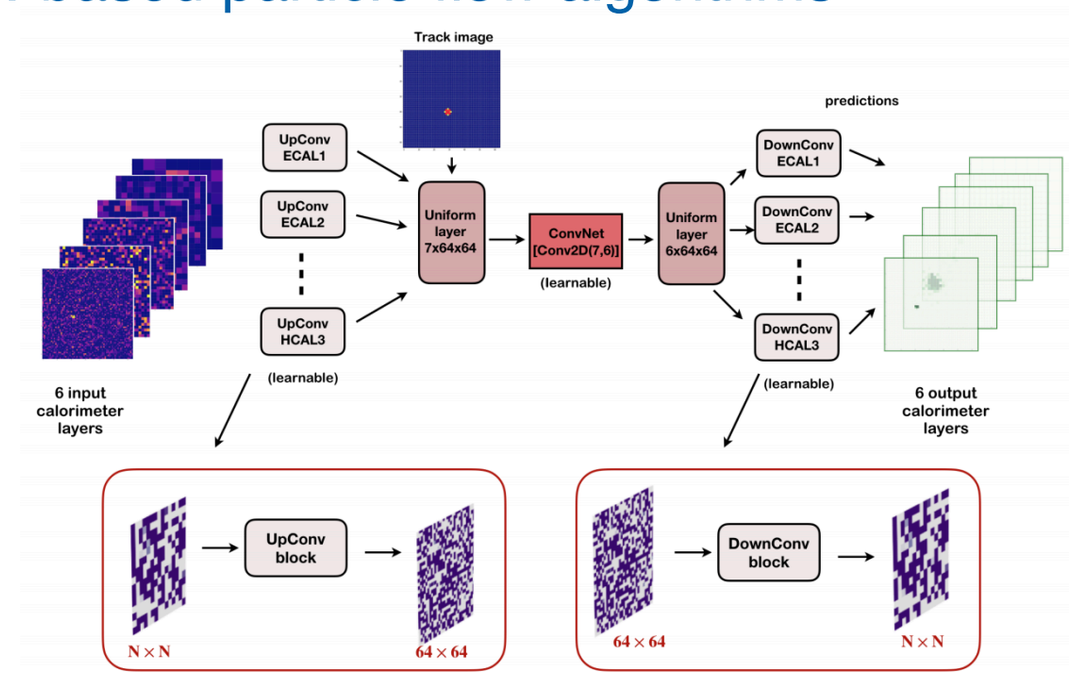
- Collider detectors exploring the use of CNNs and Graph techniques to reconstruction particle-level information
  - Similar or improved performance
  - Some issues still left: variable sized outputs

Sparse-CNN clustering for LArTPCs



https://arxiv.org/abs/1903.05663

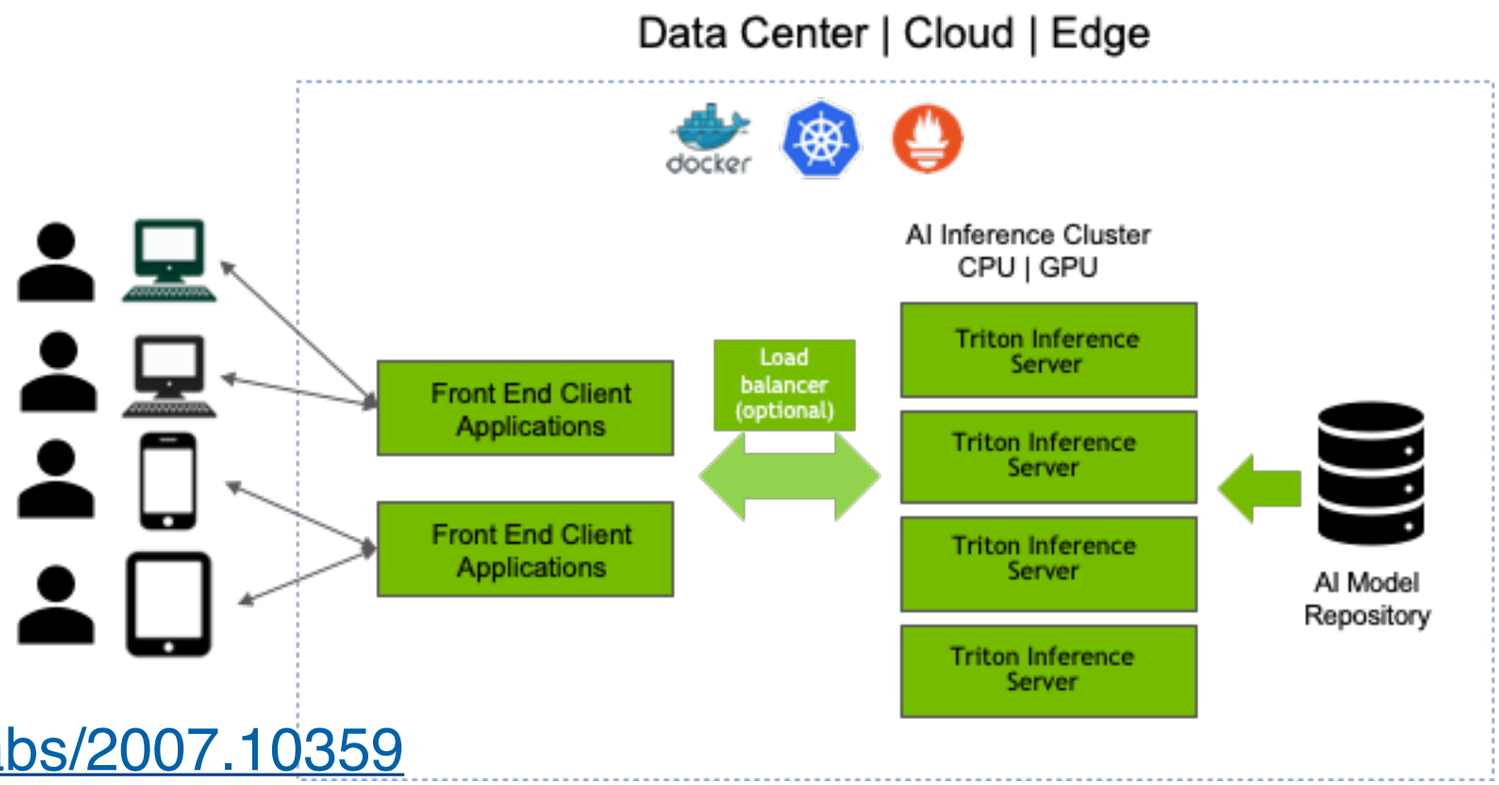
CNN-based particle flow algorithms



https://arxiv.org/abs/2003.08863

Deploying these techniques in the experiments (I)

- Experiment software stacks are often difficult to deal with, but are requiring more and more ML inference as part of their standard operation
  - Difficult to update to cutting edge software without expert knowledge
  - Already many moving parts, difficult to maintain
  - Adding machine learning frameworks to this means even more complexity!
- We are exploring decoupling this by using machine learning *as a service*
  - Experiment framework then makes standardized, lightweight api calls to a separate server running a big stack of GPUs and all the models one may require for reconstruction



https://arxiv.org/abs/2007.10359

Deploying these techniques in the experiments (II)

- GNNs, being rather new, were not readily compatible with inference as a service frameworks
  - Often required some contortion or limitation on how you were defining the model
  - Makes the process of model development and maintenance a pain!
- Together with authors of a GNN package for pytorch we developed a way to make the models immediately deployable
  - With no code changes, access to these powerful models to experiments is fairly easy
  - First large-scale tests of GNNs in CMSSW using models from this talk underway

```python
import torch
import torch.nn.functional as F
from torch_geometric.nn import GCNConv

class Net(torch.nn.Module):
    def __init__(self, in_channels, out_channels):
        super(Net, self).__init__()
        self.conv1 = GCNConv(in_channels, 64)
        self.conv2 = GCNConv(64, out_channels)

    def forward(self, x, edge_index):
        x = self.conv1(x, edge_index)
        x = F.relu(x)
        x = self.conv2(x, edge_index)
        return F.log_softmax(x, dim=1)

model = Net(dataset.num_features, dataset.num_classes)
```

change to →

```python
def __init__(self, in_channels, out_channels):
    super(Net, self).__init__()
    self.conv1 = GCNConv(in_channels, 64).jittable()
    self.conv2 = GCNConv(64, out_channels).jittable()
```

↓ forward stays the same

```python
model = torch.jit.script(model)
```

serialize jittable model

https://pytorch-geometric.readthedocs.io/en/latest/notes/jit.html

Concluding Remarks

- Machine learning is to a state where it can process and learn features for extremely complex data
  - Starting from lists of expert features with MLPs we now have convolutional and graph-convolutional networks that can effectively discriminate varieties of jets and showers
  - ML is well adopted in high energy physics and has brought about a revolution in the study of jets (heavy flavor, merged jet tagging, etc.)
  - Beginning to see the use of ML directly on detector inputs as low-level reconstruction

- Choosing the right loss function for a task is extremely important for getting high quality results
  - Spend the time to think about what your loss function is encoding, in addition to how you are structuring your network
  - While estimating a value, predicting a category, and assigning coordinates are all fundamentally related, the way you use these outputs are vastly different in practice

- We can use these tools at scale in our experiments through the use of GPUs and scalable software infrastructure
  - There is very little reason not to exploit these tools in your analysis if it makes sense to!