An Introduction to Neural Networks

Dubna University

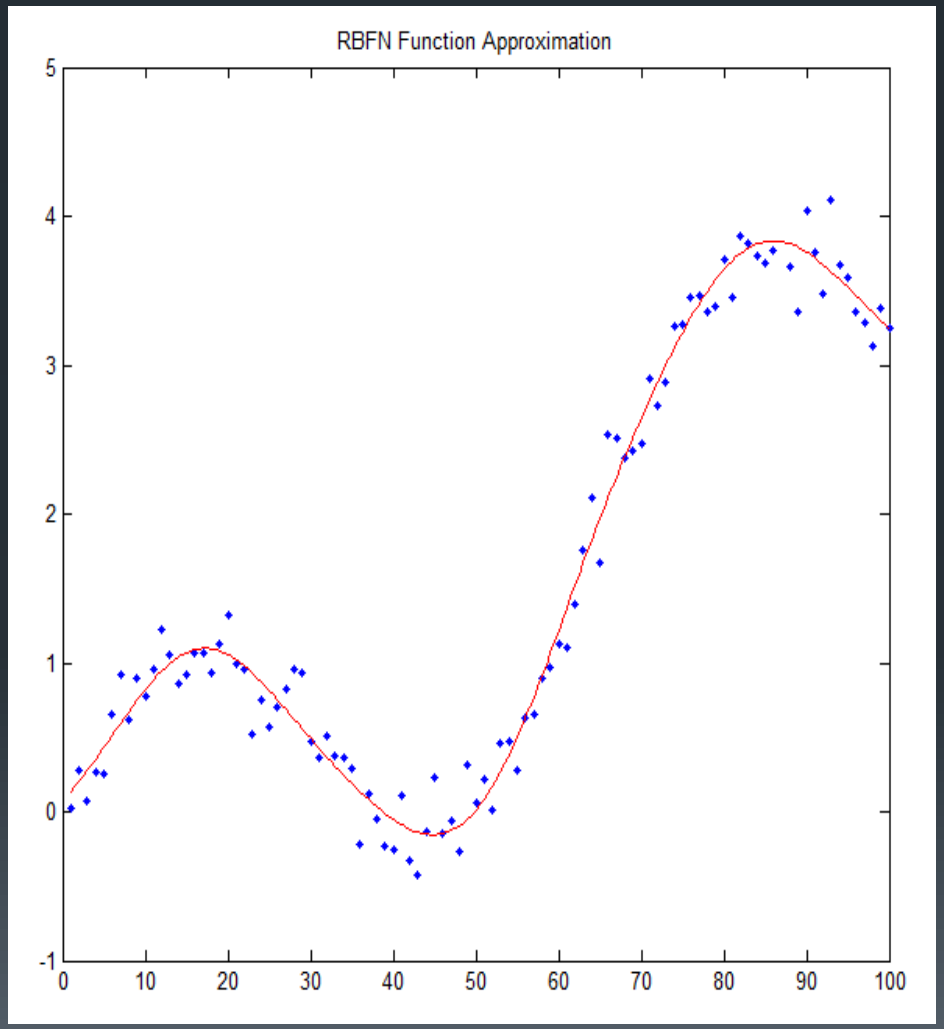
Chris Kullenberg

Oct. 16 2019

What is a Neural Network (NN)?

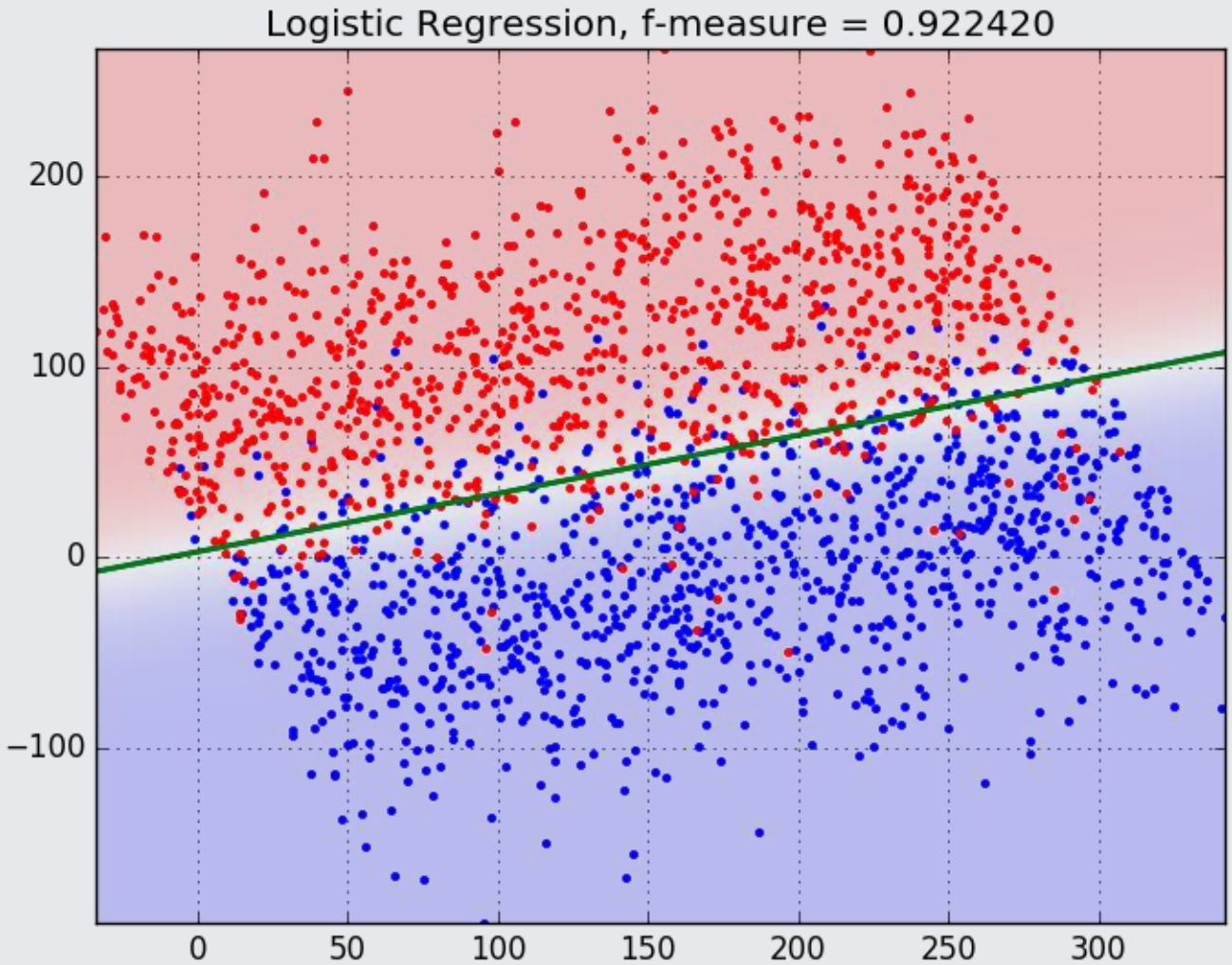
- Neural Networks are effectively function approximators
- They are loosely modeled on neural activity in the brain (neurons fire or remain silent based on sensory input)
- NN^{s} have many, though not unlimited, applications

Function approximator?



- A NN with ∞ neurons can reproduce any function
- In practice this means we can approximate any function
- As we increase the number of neurons we get closer to the actual function
- How is this useful?

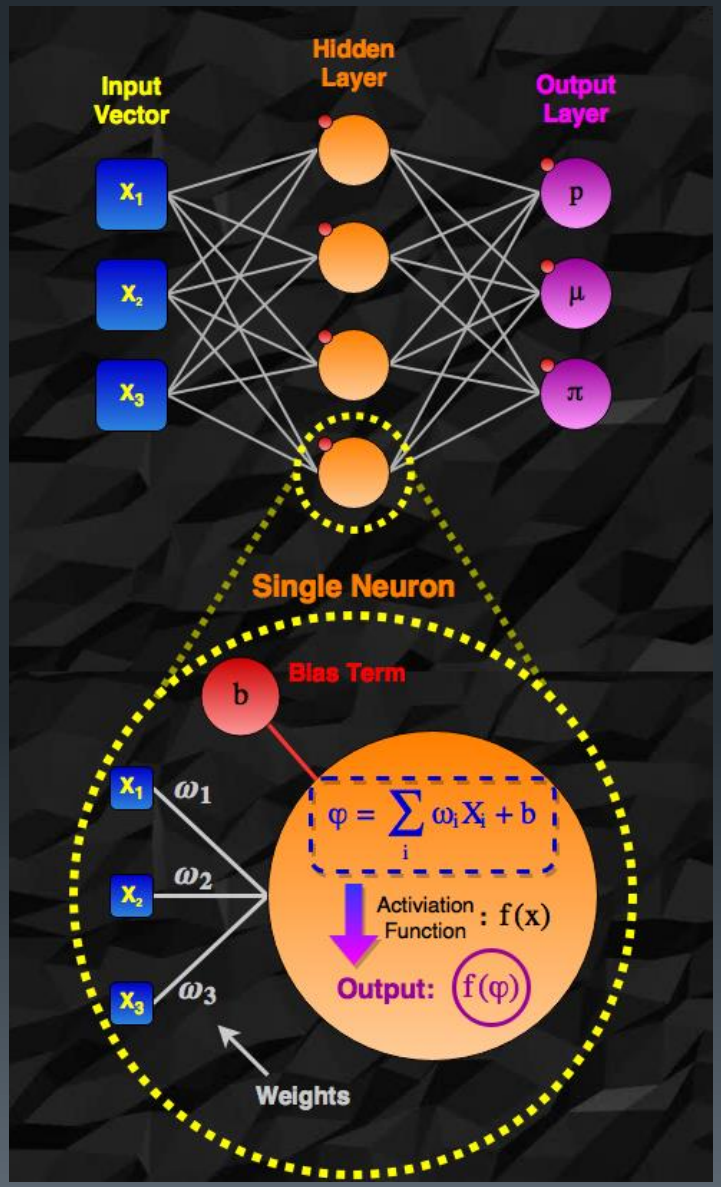
Example: Population Separation



Typical Applications

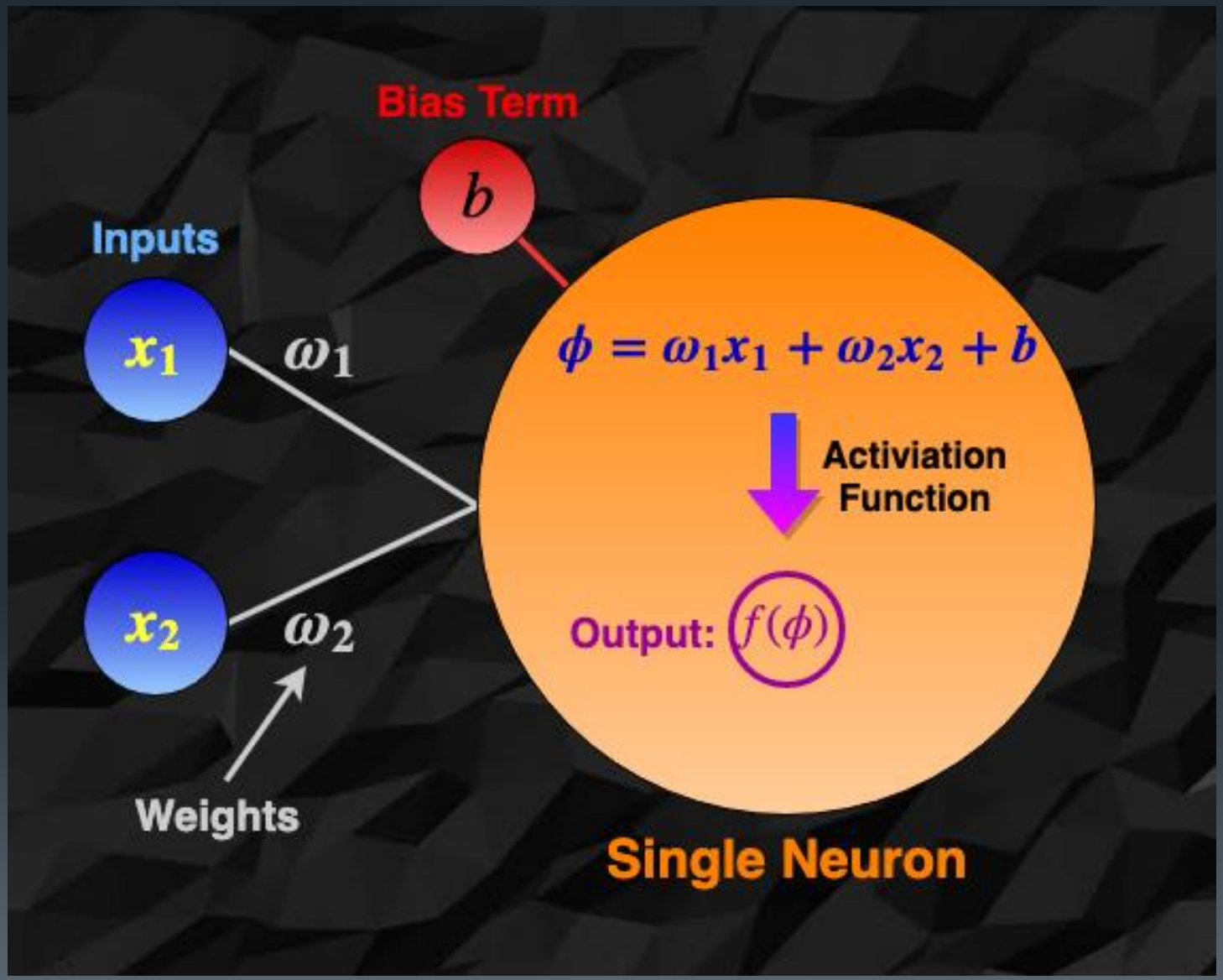
- Physics analysis (classify events, reconstruction)
- Medical diagnosis
- Finance (market prediction, loan classification)
- Image processing (facial recognition, self-driving)
- Nearly any classification problem

Basic Structure of a NN



- Network is composed of neurons fully connected to network inputs and each other
- The output of a neuron is a weighted combination of its inputs, with an added constant (bias term) (and an applied activation function)
- Weights and biases are the parameters of the network
- We want find the parameters that produce the desired output

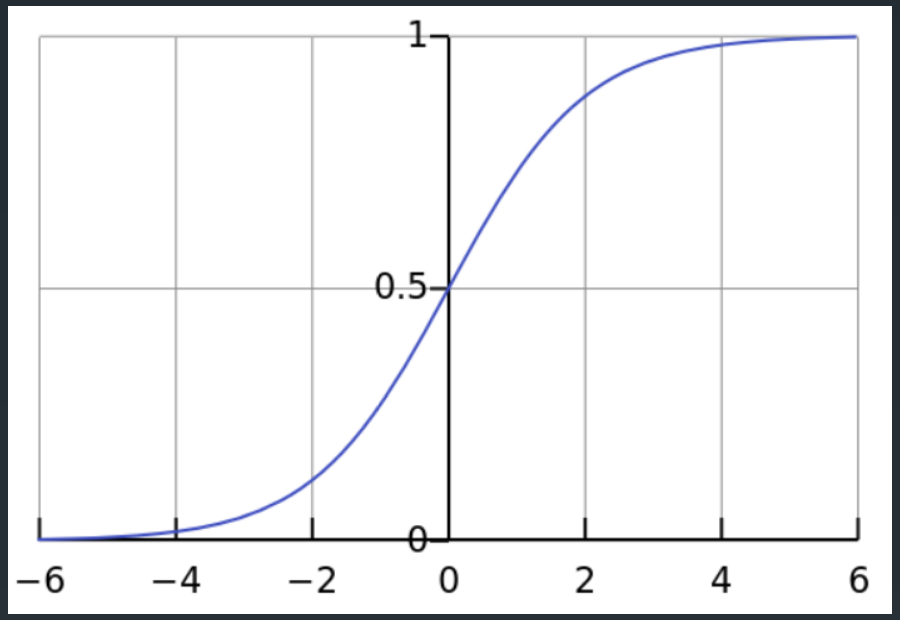
A Single Neuron



Activation Functions

- Activation functions provide non-linearity to the network (necessary for generalization to any function)
- Multiple layers of neurons without activation functions are equivalent to a single layer (just linear combination of the inputs)
- In the same way, a linear activation function will provide no additional complexity.
  So activation functions must be non-linear!

Sigmoid: the classic activation function

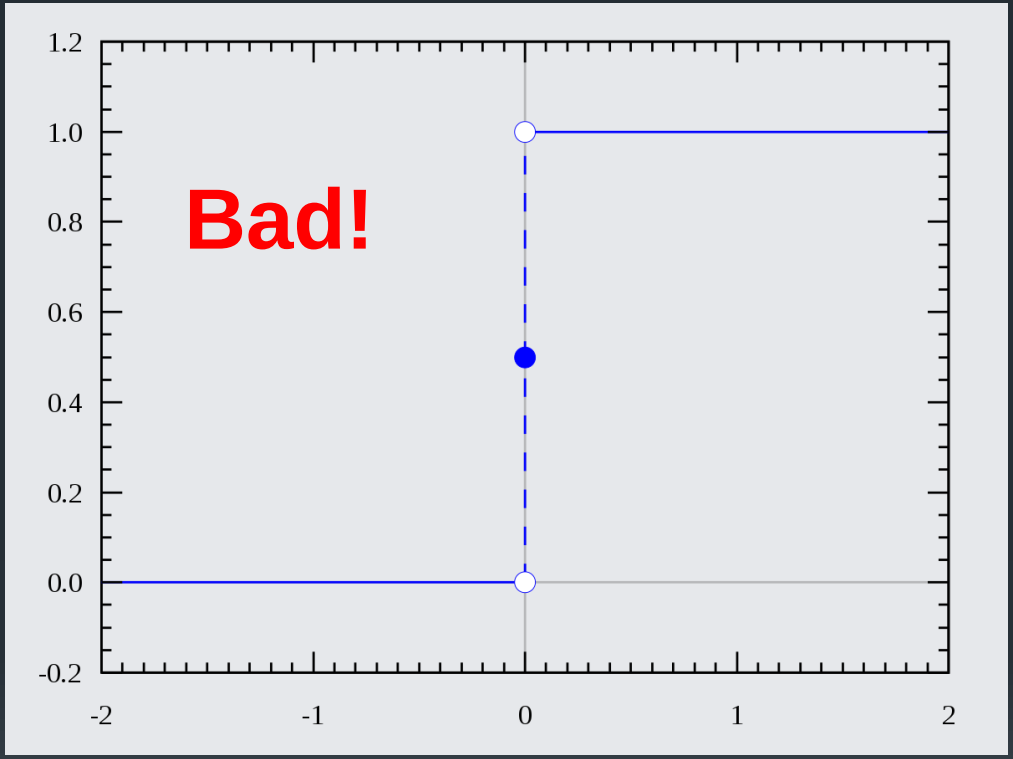


- Output range: (0,1)
- Like On/Off state of a real neuron
- Continuous!

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

Between x range (-4,4) we have a smooth transition from 0 (inactive) to 1 (activated). This is important!

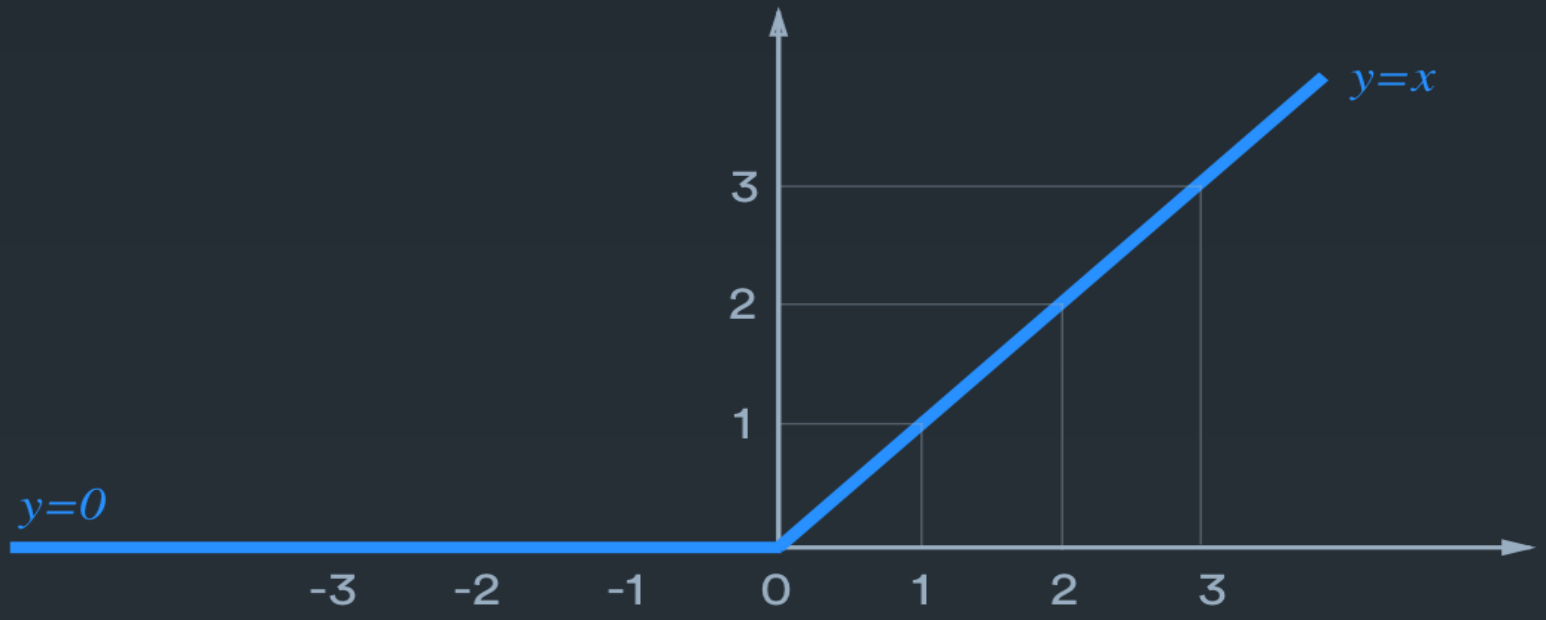
Why do we want a smooth activation function?



- Small changes in weights move us along this curve
- At some point a small change will cause a sudden change in the activation of the neuron
- This will cause chaotic behavior in the network

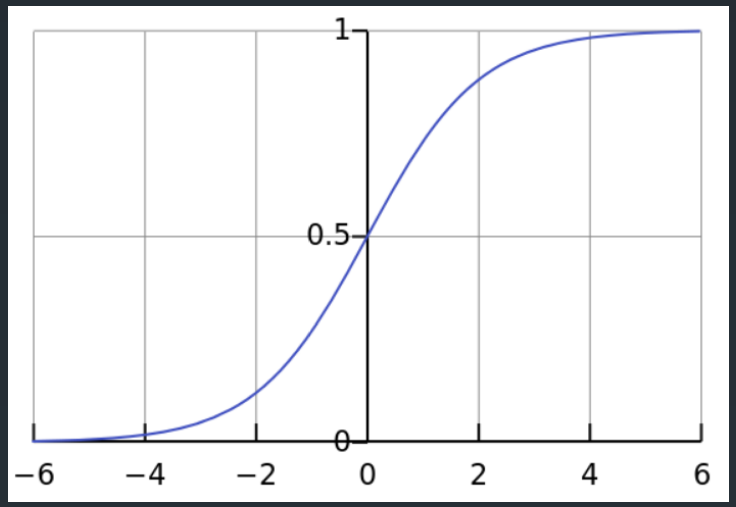
We need a continuous activation function so that the network changes smoothly when modifying the parameters!

ReLu: The new activation function



- ReLu is non-linear, because all negative inputs are changed to zero
- Used to avoid small contributions from neurons in early layers in deep networks

Why a bias term in the sum?



- Let's use the Sigmoid activation function
- Let's also assume that the input to the function is always very large (without a bias term):

$$(\omega_1 x_1 + \omega_2 x_2) > 1000$$

- The neuron output will always be 1, because the inputs to the activation function are always much larger than 4!
- Always returning 1 (independent of input) is not very useful
- We need to shift the inputs down by ~1000 to put them in a range where the sigmoid function can discriminate
- This is done with the bias term, which is learned for each neuron

Training the network

- First initialize all network parameters
  (traditionally this is done randomly)
- Then pass labeled data (with the correct answers) through the network
- Check how correct the network solution is (Cost function)
- Modify parameters to make the network solution a bit more correct (Backpropagation)
- Continue running data through the network and modifying parameters until desired accuracy is achieved
  (or for a set number of repetitions)

Training Data

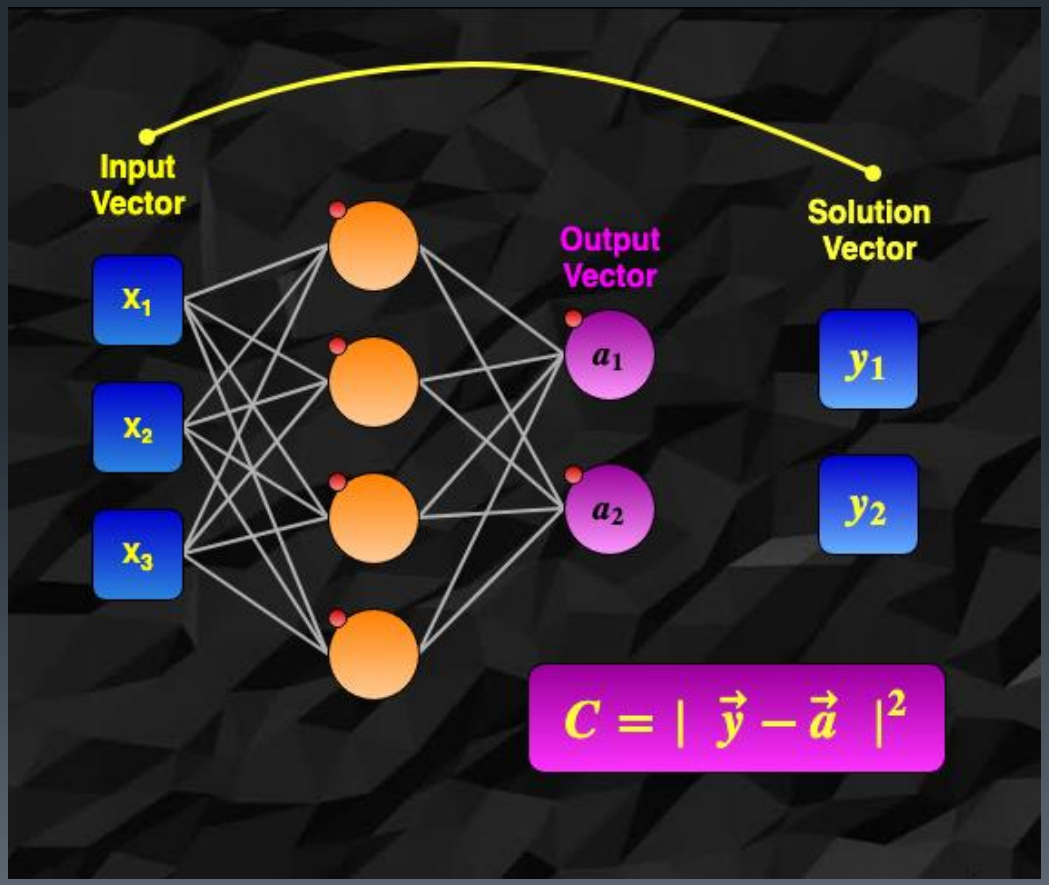
- We need a set of labeled data to train the network (supervised learning). This will be N entries, each with:
  - A list of input variables: x_1, x_2, x_3 ….
  - The “solution” for each entry (correct category, for example)
- We split this data into a training and testing set (generally 2/3 and 1/3 of the full set, respectively)
- The training set is used to train the network, or tune its parameters
- The testing set is used to test the network on data it has not seen during learning

Cost/Loss Function

- The Cost function tells us how correct the network output is. It is any function such that the minimum value occurs at the correct solution.
- We want this to be a minimum because we will use its gradient to modify parameters
- A typical cost function:

$$C = |y - a|^2$$

Where **a** is the network output and **y** is the correct solution from the data.

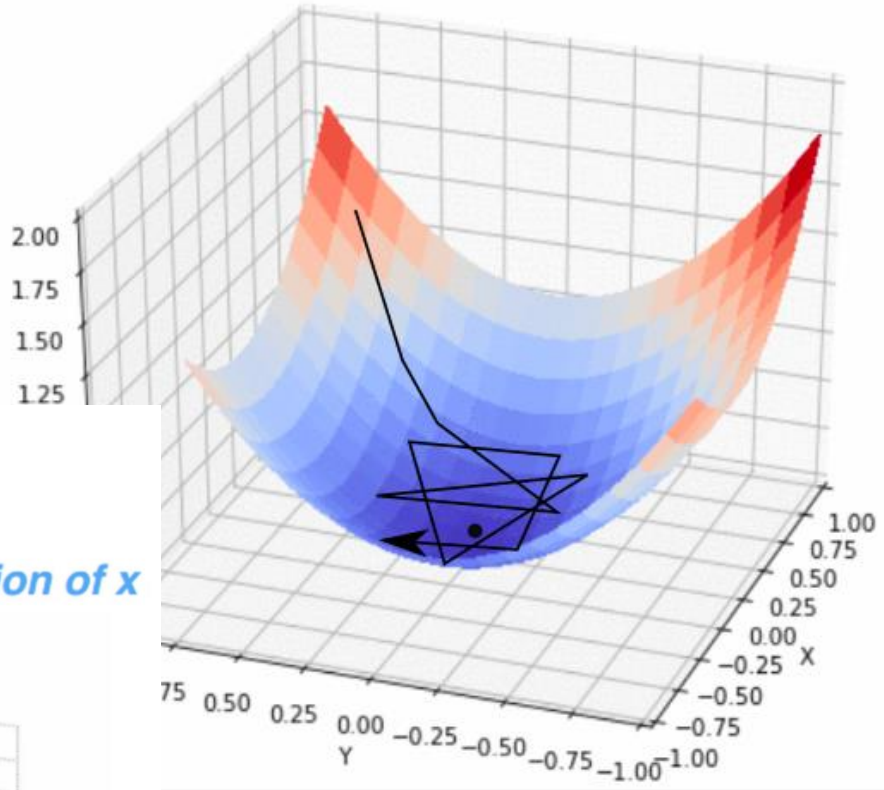


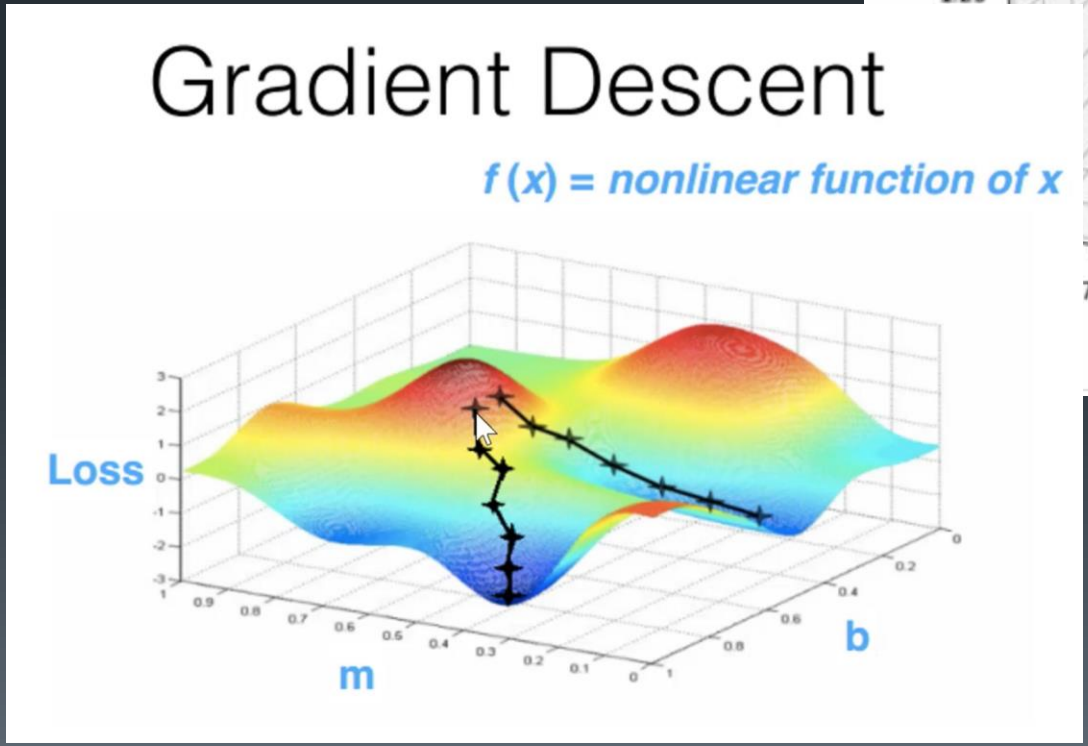
Modify parameters

- To modify the network parameters we use Backpropagation, or Gradient Descent
- Effectively we calculate the partial derivatives of the cost function with respect to all of the parameters. This gives us the gradient.
- We then move a small amount in the direction opposite of the gradient (want lower cost function) by a small change in the parameters
- If we make many steps towards a smaller cost function we should eventually find a minimum (and so a small difference between the network output and the true values)

Gradient Descent

We may find a local minimum, rather than global, which is fine.



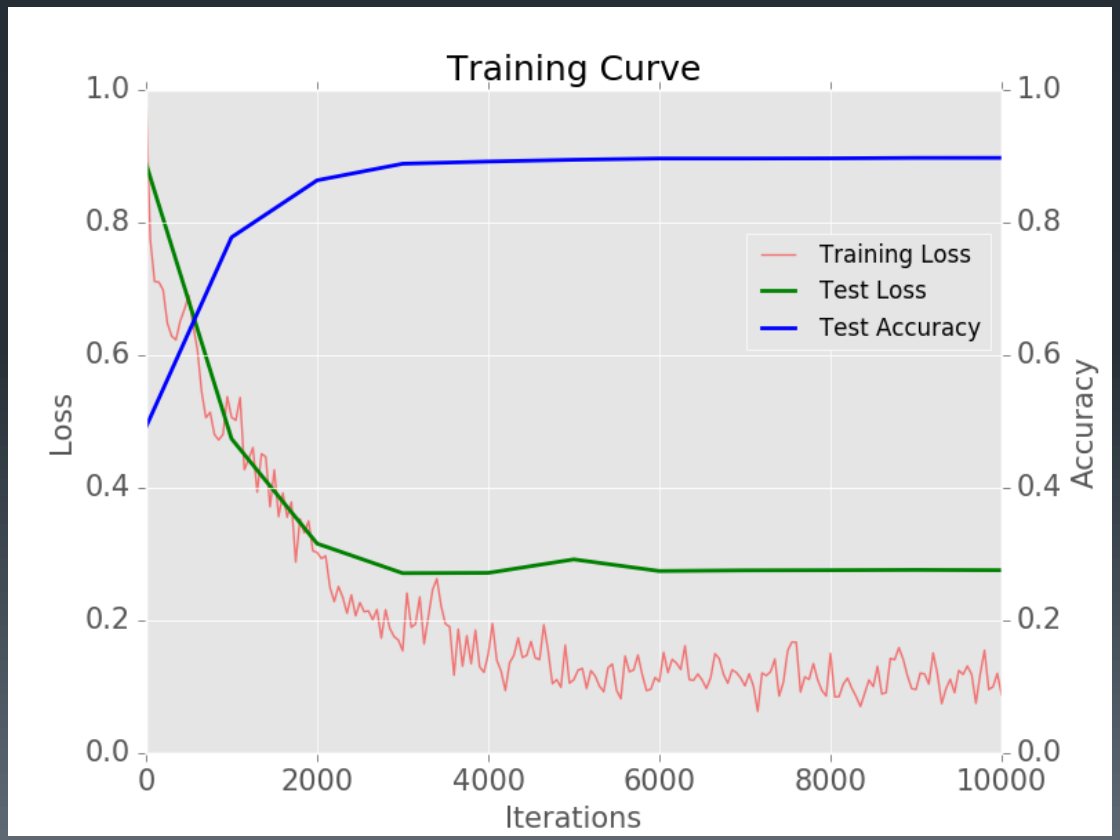


If we are not careful with the step size we may bounce around the minimum.

Accuracy/Training Plots

- Here we have a typical example of training curves for a network.

- **Blue** shows the **accuracy** of the network predictions (test data set)
- **Green** and **Red** show the value of the Cost/Loss values for the **testing** and **training** data sets, respectively.

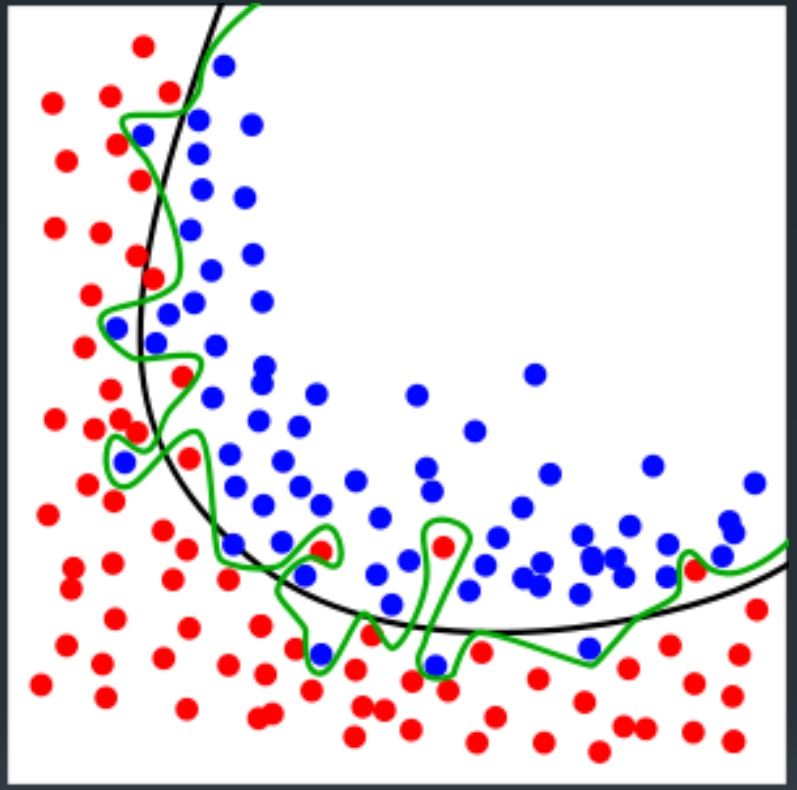


- Here the accuracy reaches ~90%

- This level of accuracy is reached rather early in the training

Overtraining

- It is possible for a network to Overtrain, or for it to learn specific features of its training data too well
- In this case the network will not be as general, and will not perform well on data it has not seen before

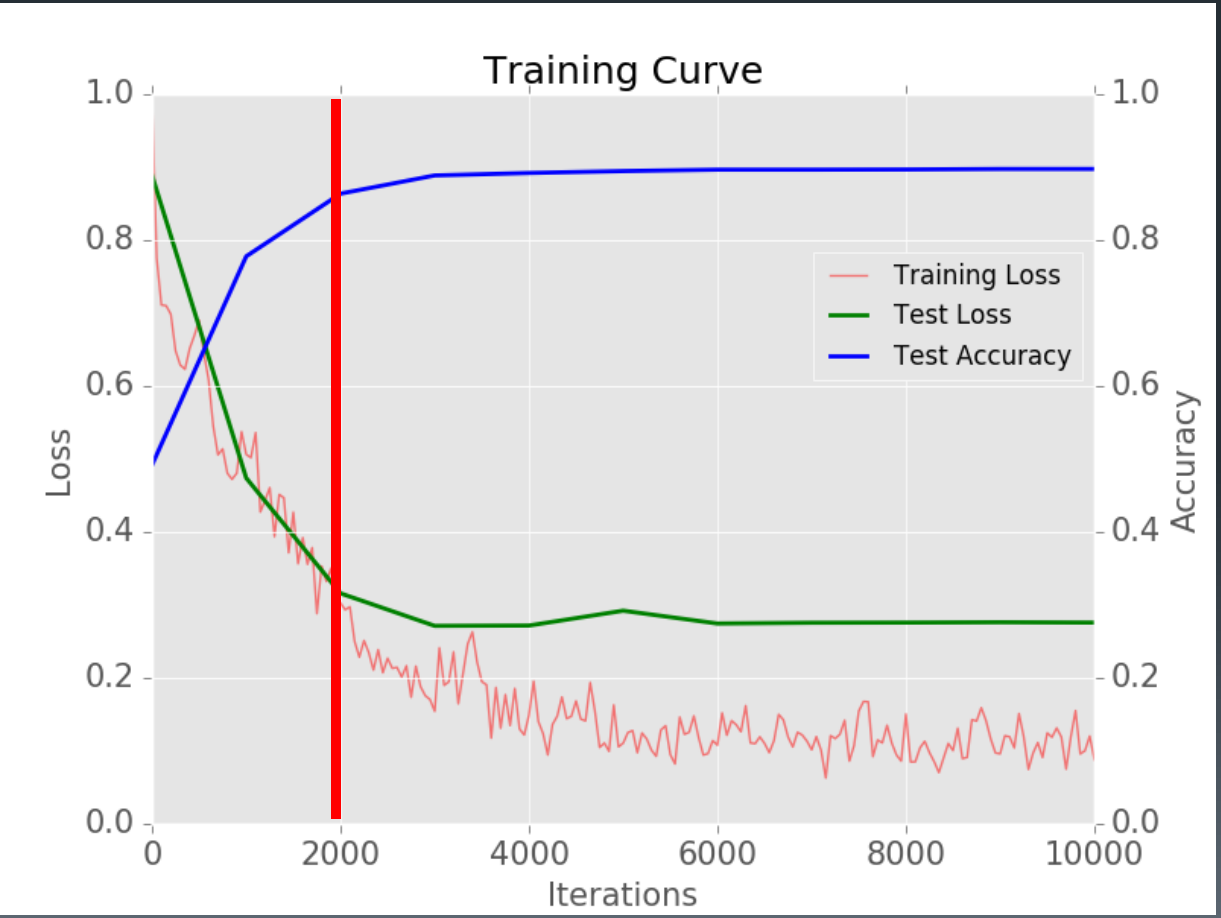


This is much like overfitting, which you can see in this plot. The black curve would be a good separating function, while the green curve focuses too much on this particular data set.

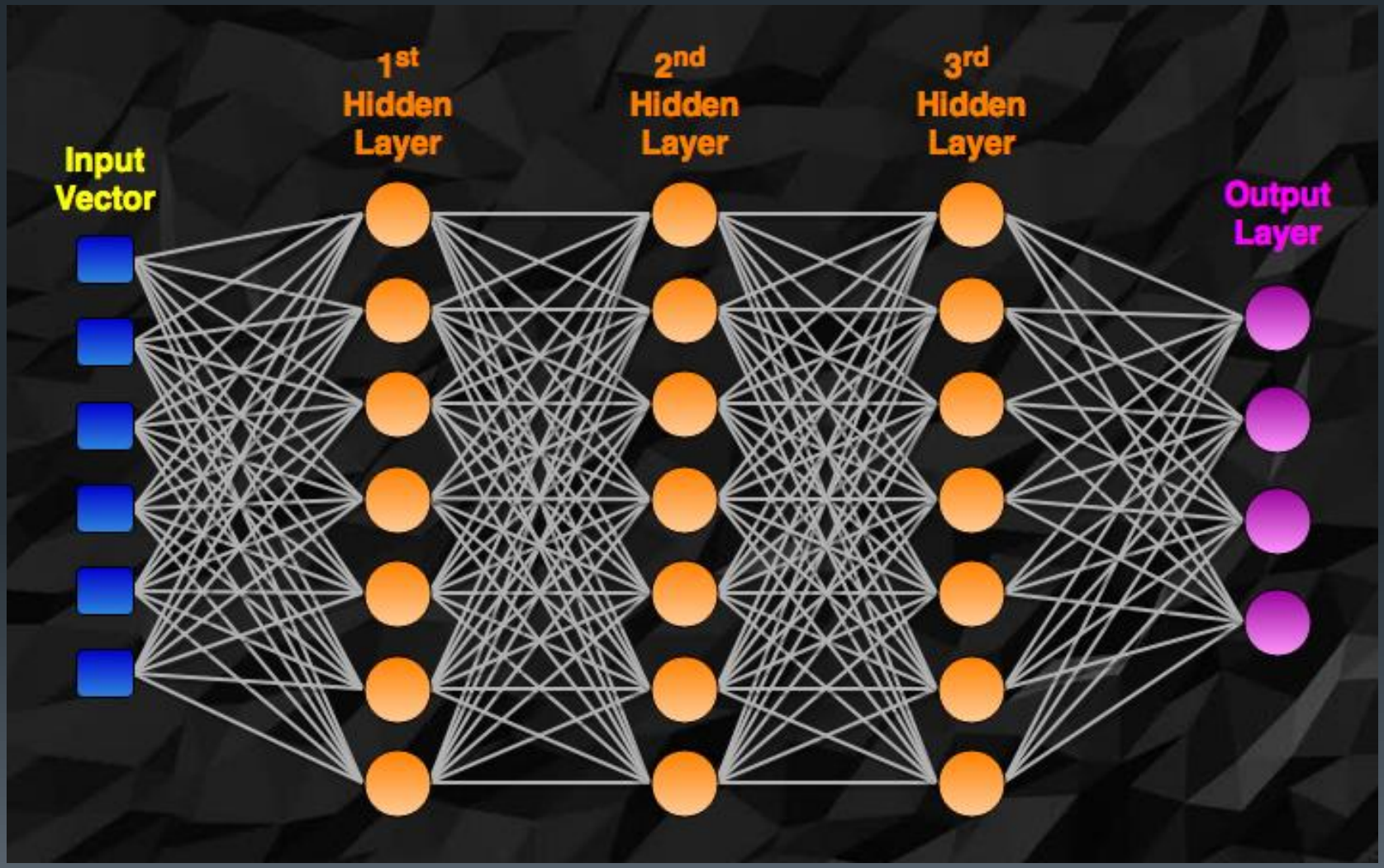
Test/Training loss when overtraining

We can see overtraining occur in our example plot. After 2000 iterations the loss from the training data continues to drop, while the test loss does not improve. These curves should overlap.

It is probably best to stop training this network near 2000 iterations (maybe 3000).



Deep Neural Networks

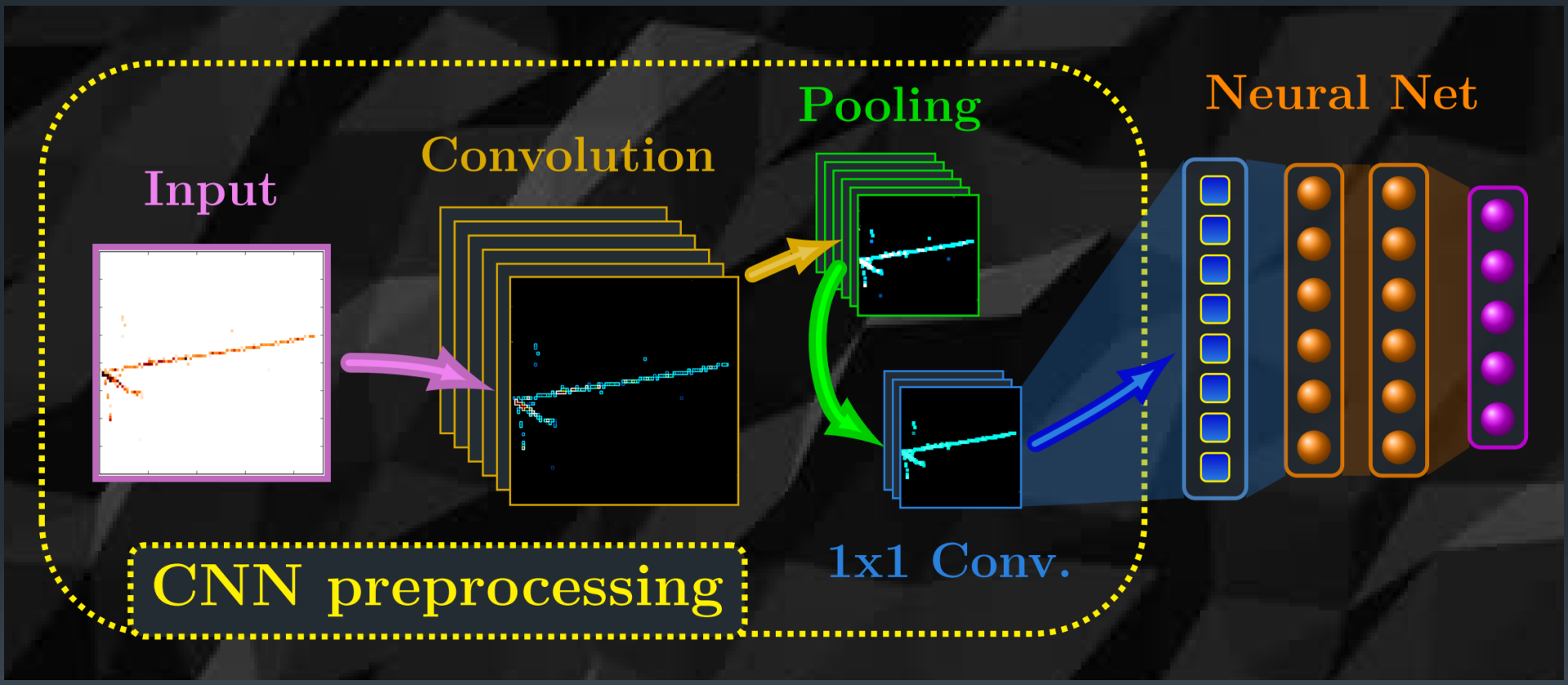


Why are Deep NNs possible now?

Traditionally Deep networks (large NNs) were not feasible. They are now possible due to improved hardware, as well as the following:

- The ReLu activation function allows neurons in early layers to have an effect on the Cost function's gradient
- It was found that most local minima are as good as the global, so it is not necessary to search too hard for it
- Stochastic Gradient Descent (checking the gradient after small batches, rather than after the full training set)
- New methods of parameter initialization (instead of random)

Convolutional Neural Network (CNN)

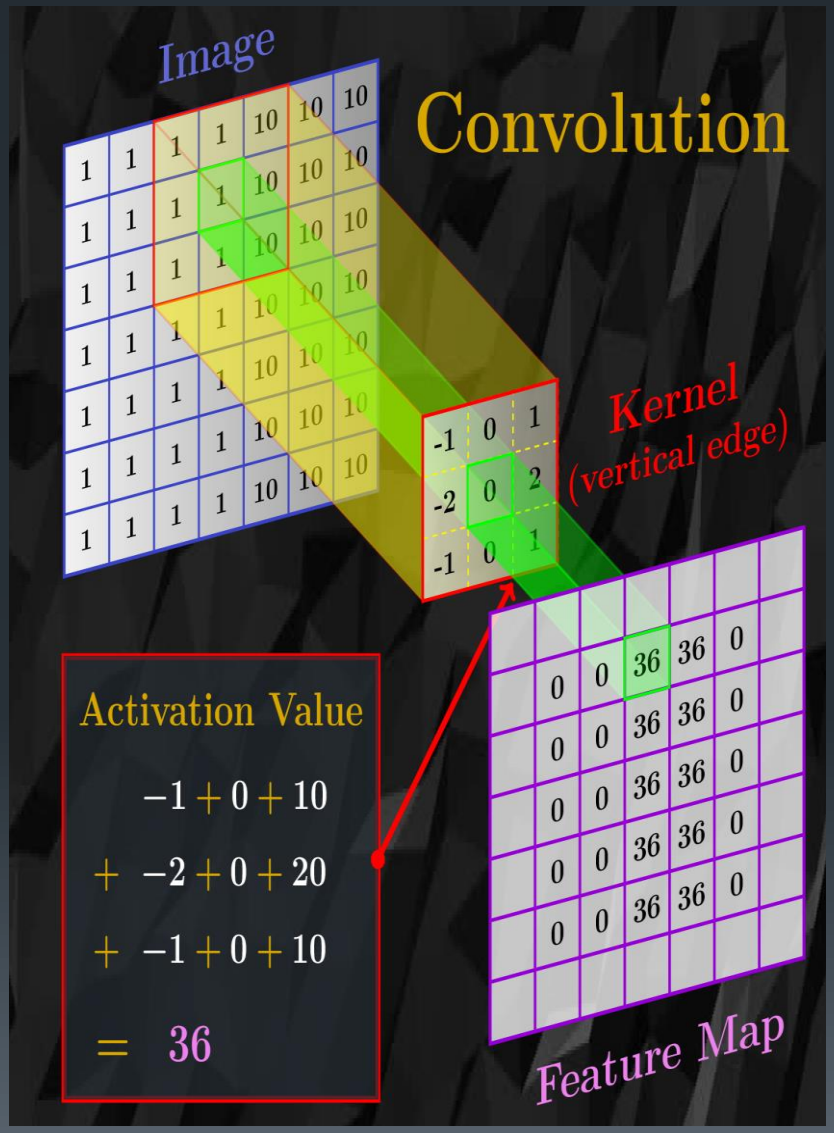


Effectively a set of image preprocessing steps which feed into a standard Neural Network

Why is preprocessing necessary?

- Using each pixel of a large image as input will create a massive Neural Network
- A simple NN may not be able to find the desired object in a different position of an image (spatial invariance)
- Convolution can also find “features” that a pure NN may not ever learn

Convolution (pattern searching)



The **Kernel** / **Filter** is a simple matrix

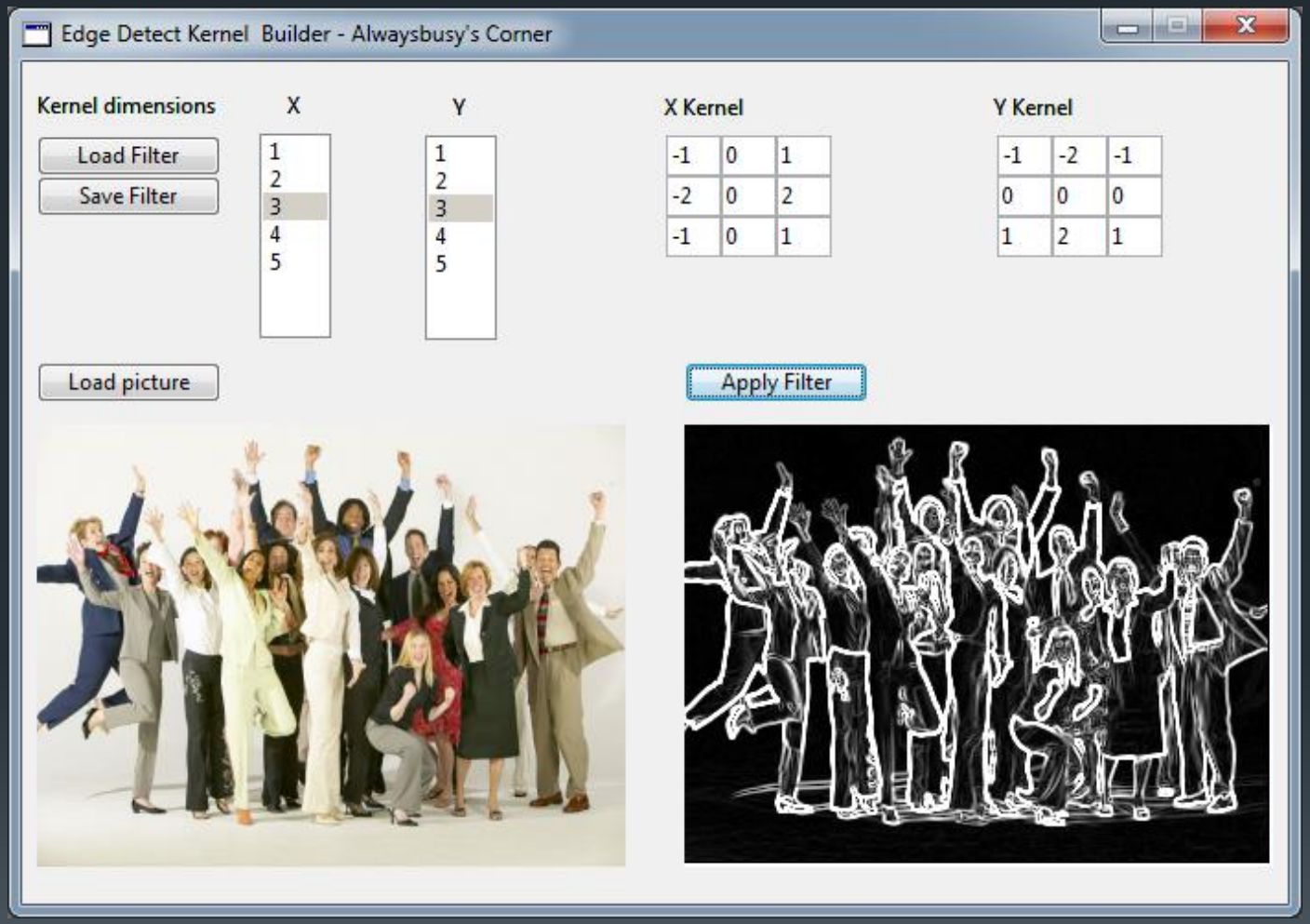
The kernel is moved along the input, and the sum of the products of the kernel values and the image values under the kernel are used as output (simply a weighted average)

The resulting output is the **Activation** / **Feature Map**

The **Stride** is simply the number of spaces the kernel moves with each step

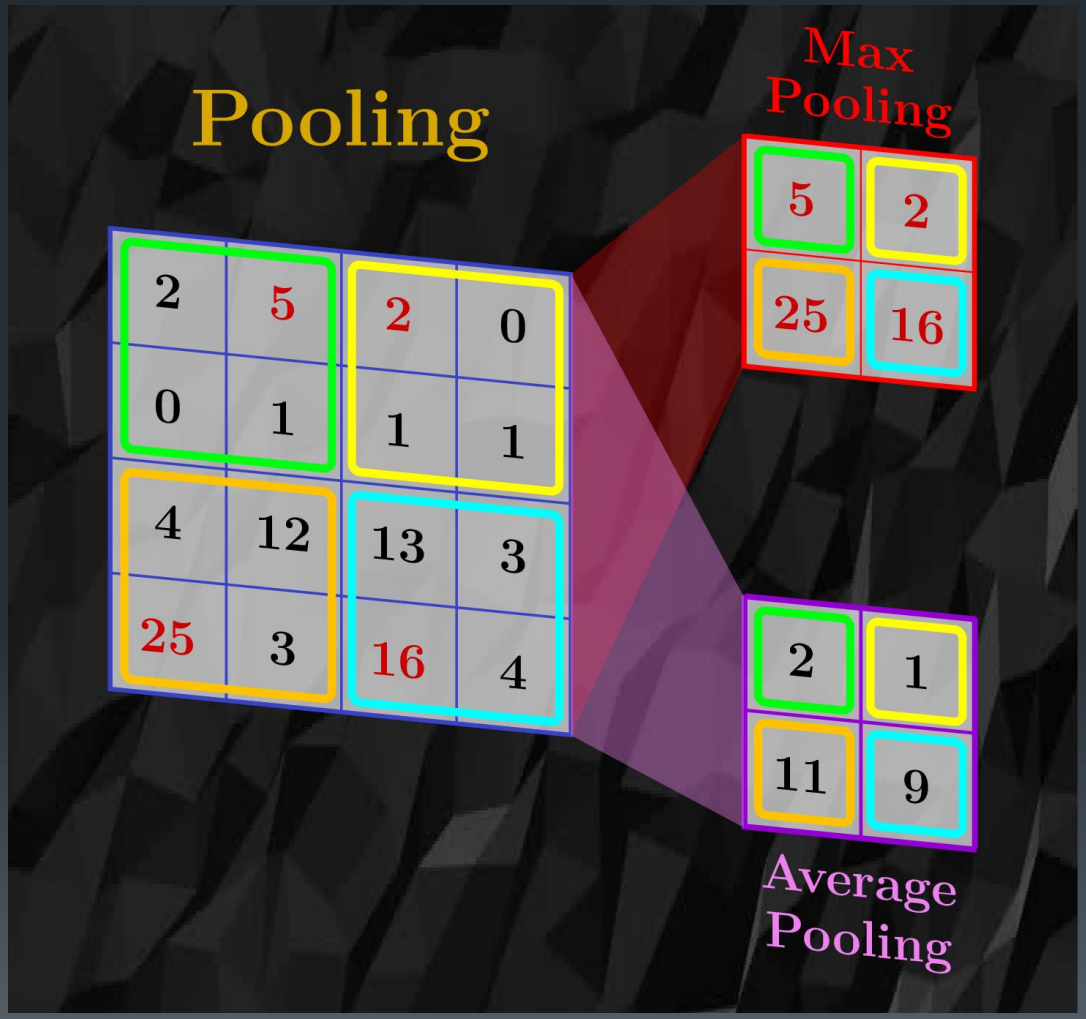
The output layer is sometimes padded with zeros to have a size consistent with the input layer (**Padding**)

Convolution example: Edge finding



Kernels are generally learned by the network!

Pooling (information reduction)

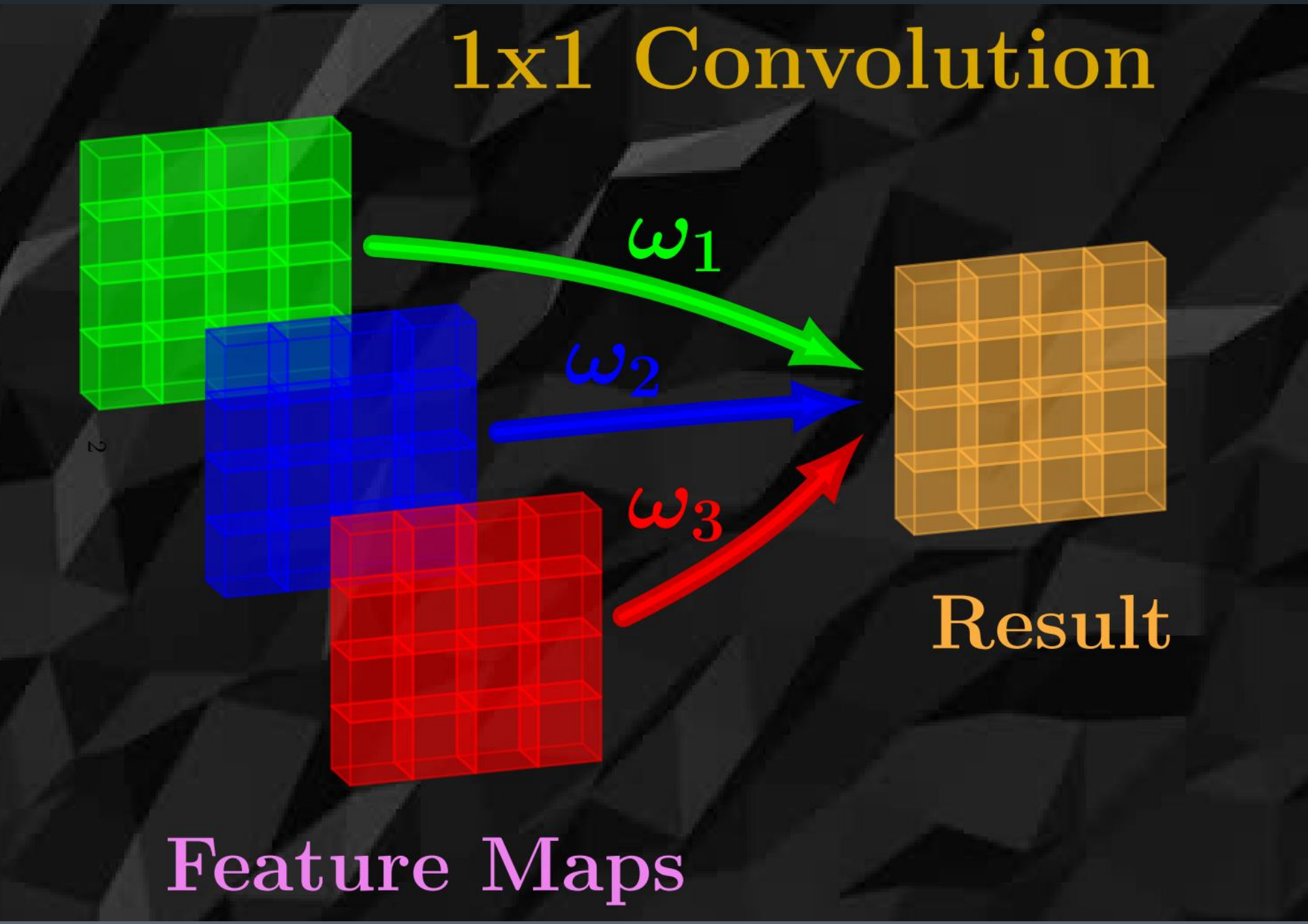


Pooling is used to reduce the information flowing to the network, while keeping important activations

In this way we retain the general locations of features (kernel activations), without keeping less interesting information

Max Pooling: Keep only the largest activation in an area of the feature map

Average Pooling: Average all activation values within the area


1x1 Convolution
ω_1
ω_2
ω_3
Result
Feature Maps

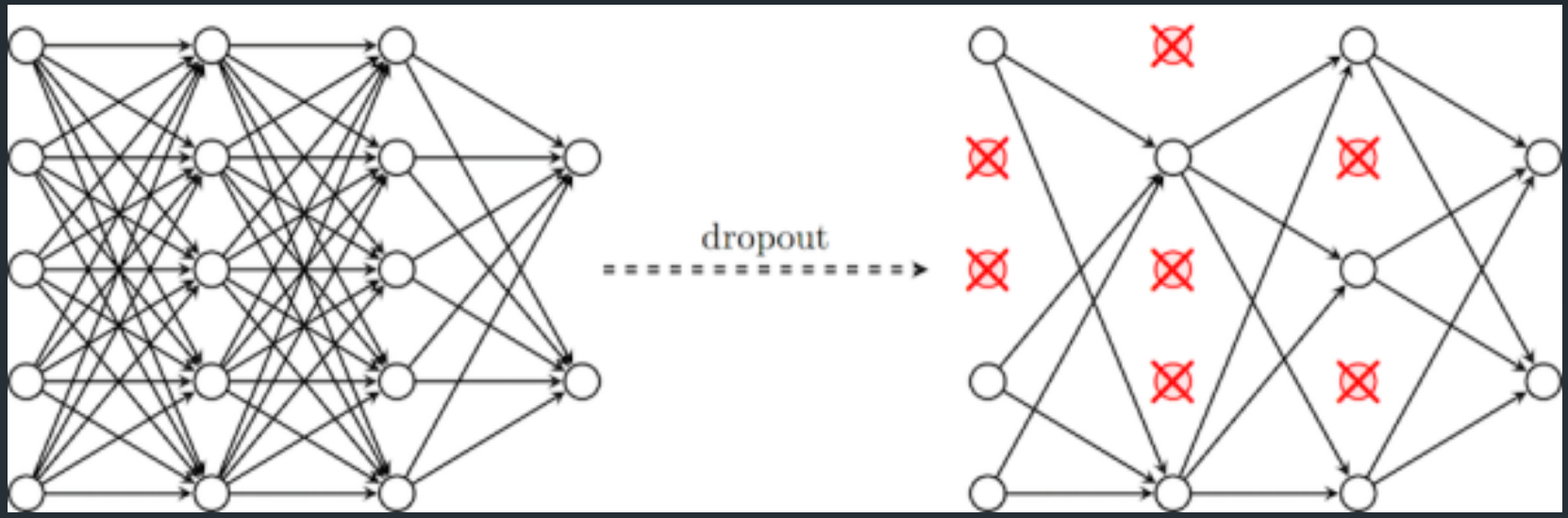
1x1 Convolution
(information reduction and pattern interweaving)

- This layer is used to reduce the parameter space and interweave patterns from feature maps
- It is surprisingly powerful, and further helps to detect features while reducing data flow
- Basically, feature maps are added together with weights, which become network parameters.

So if we have 200 kernels, and so 200 feature maps, we can add them together with weights to make as many/few outputs as we like.

One can imagine adding a vertical edge feature map to a horizontal one, to get a general edge detection, so 2 maps become 1 more powerful map

Dropout Layer



Used only during training!

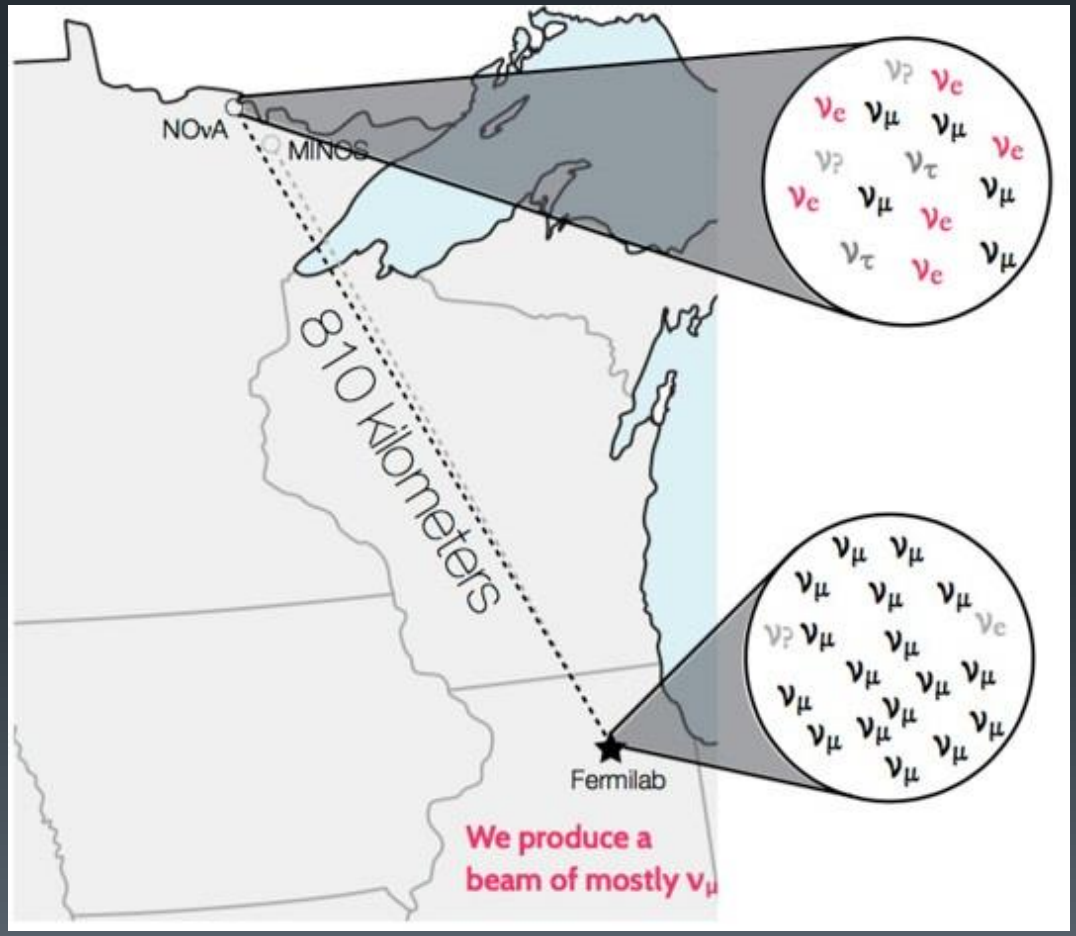
- Applied to fully connected layers (the neural network at the end)
- Nodes are randomly removed from the training process by setting their weights to zero
- This promotes redundancy in the network, and guards against overtraining

Example CNN use in physics? NOνA!

NuMI **O**ff-axis ν_e **A**ppearance

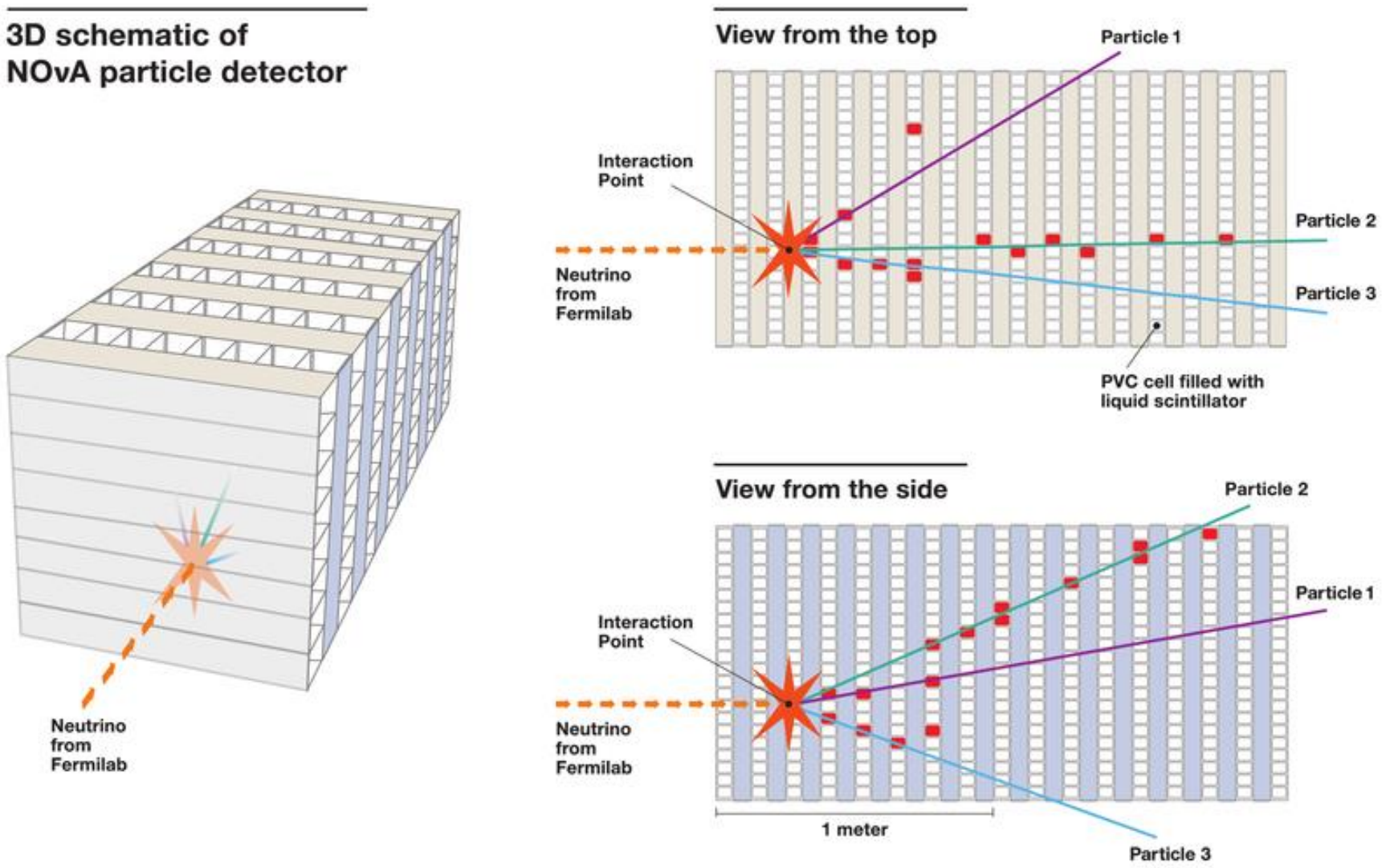
Measure neutrino oscillation parameters!

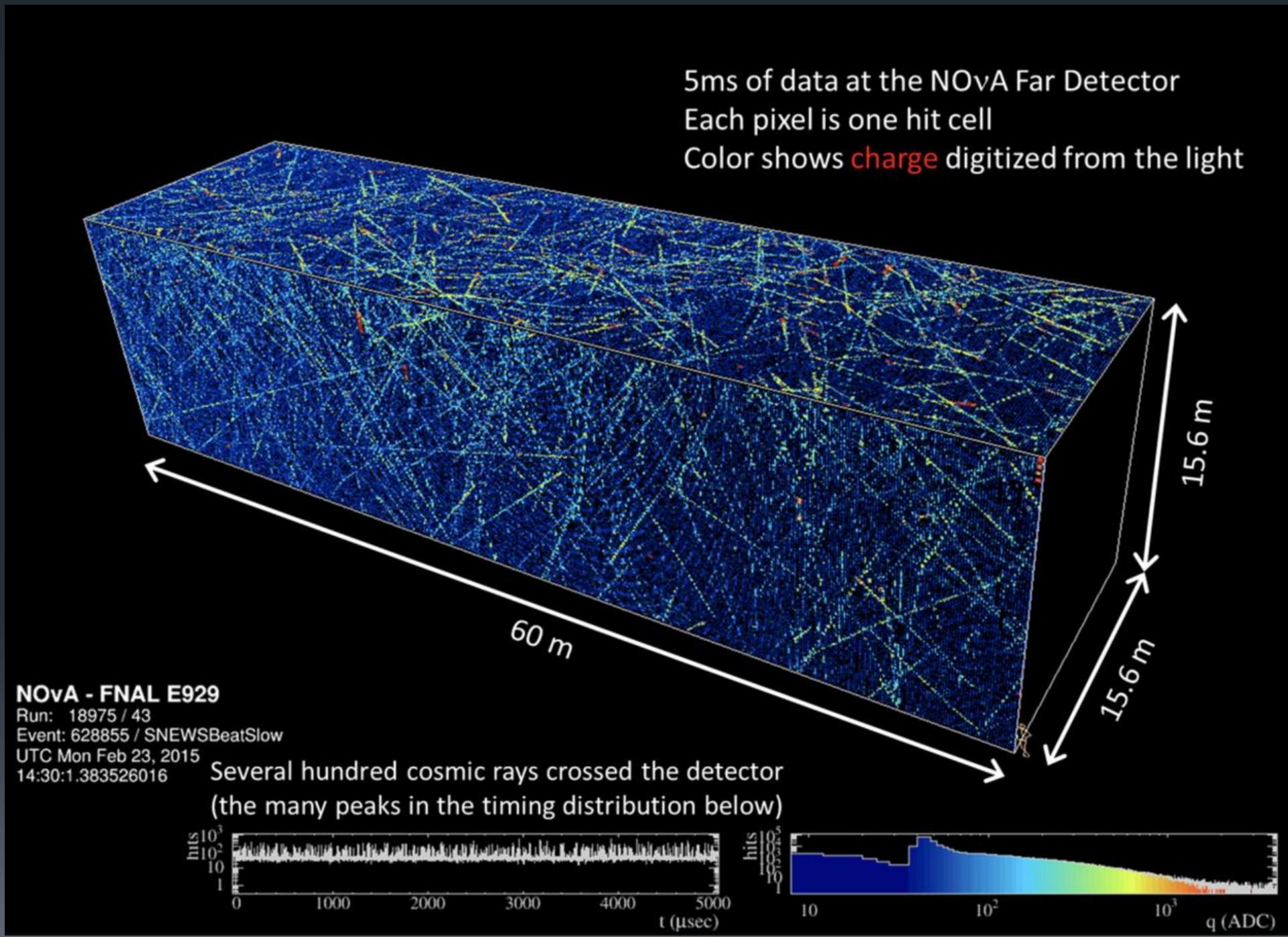
(among other interesting neutrino topics)



- Neutrino oscillations cause different neutrino flavors to appear during interaction than when created during production
- Understanding these oscillations helps us understand fundamental properties of the neutrino
- The beam composition is measured at the Near Detector (mostly muon neutrinos)
- The number of electron/muon neutrinos appearing at the Far Detector provides the oscillation measurement

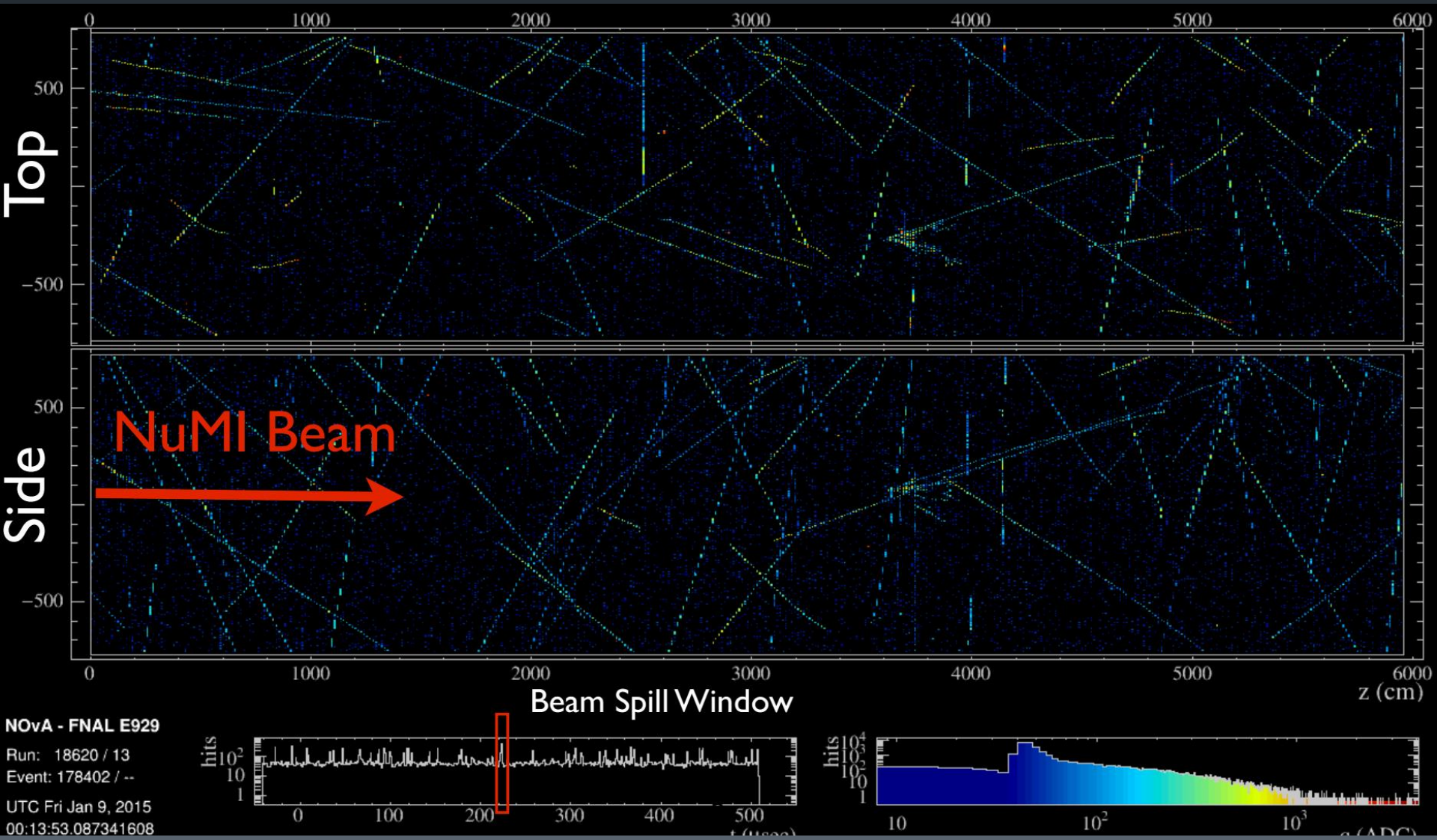
NOvA Data


3D schematic of
NOvA particle detector
Neutrino
from
Fermilab
View from the top
Particle 1
Interaction
Point
Particle 2
Particle 3
Neutrino
from
Fermilab
PVC cell filled with
liquid scintillator
View from the side
Particle 2
Particle 1
Interaction
Point
Neutrino
from
Fermilab
1 meter
Particle 3


5ms of data at the NOvA Far Detector
Each pixel is one hit cell
Color shows charge digitized from the light
15.6 m
60 m
15.6 m
NOvA - FNAL E929
Run: 18975 / 43
Event: 628855 / SNEWSBeatSlow
UTC Mon Feb 23, 2015
14:30:1.383526016
Several hundred cosmic rays crossed the detector
(the many peaks in the timing distribution below)
hits
t (μsec)
q (ADC)

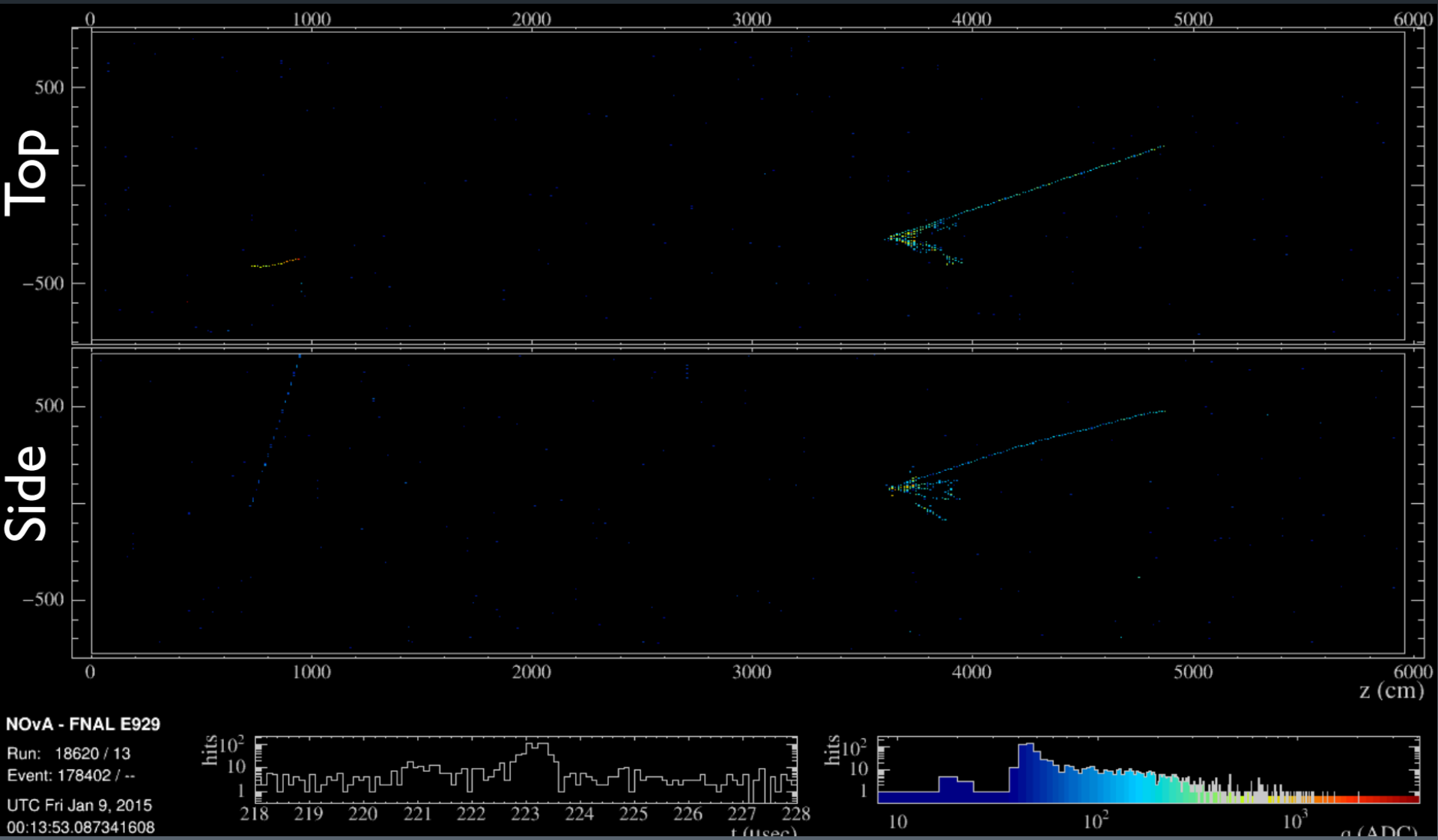
NOvA Data

550 μs window in the Far Detector

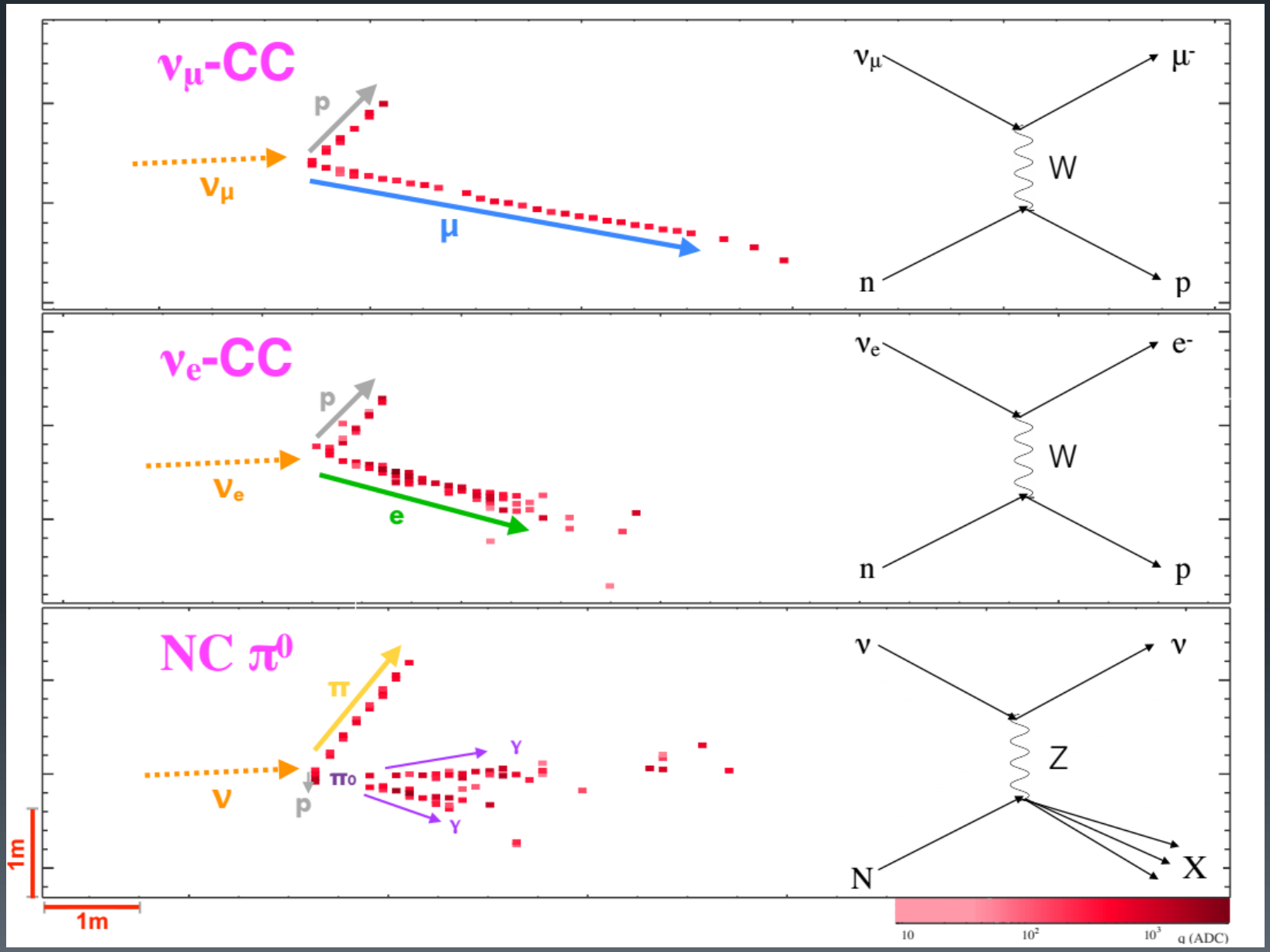


NOvA Data

10 μs window during active beam



How to Distinguish Neutrino Types?

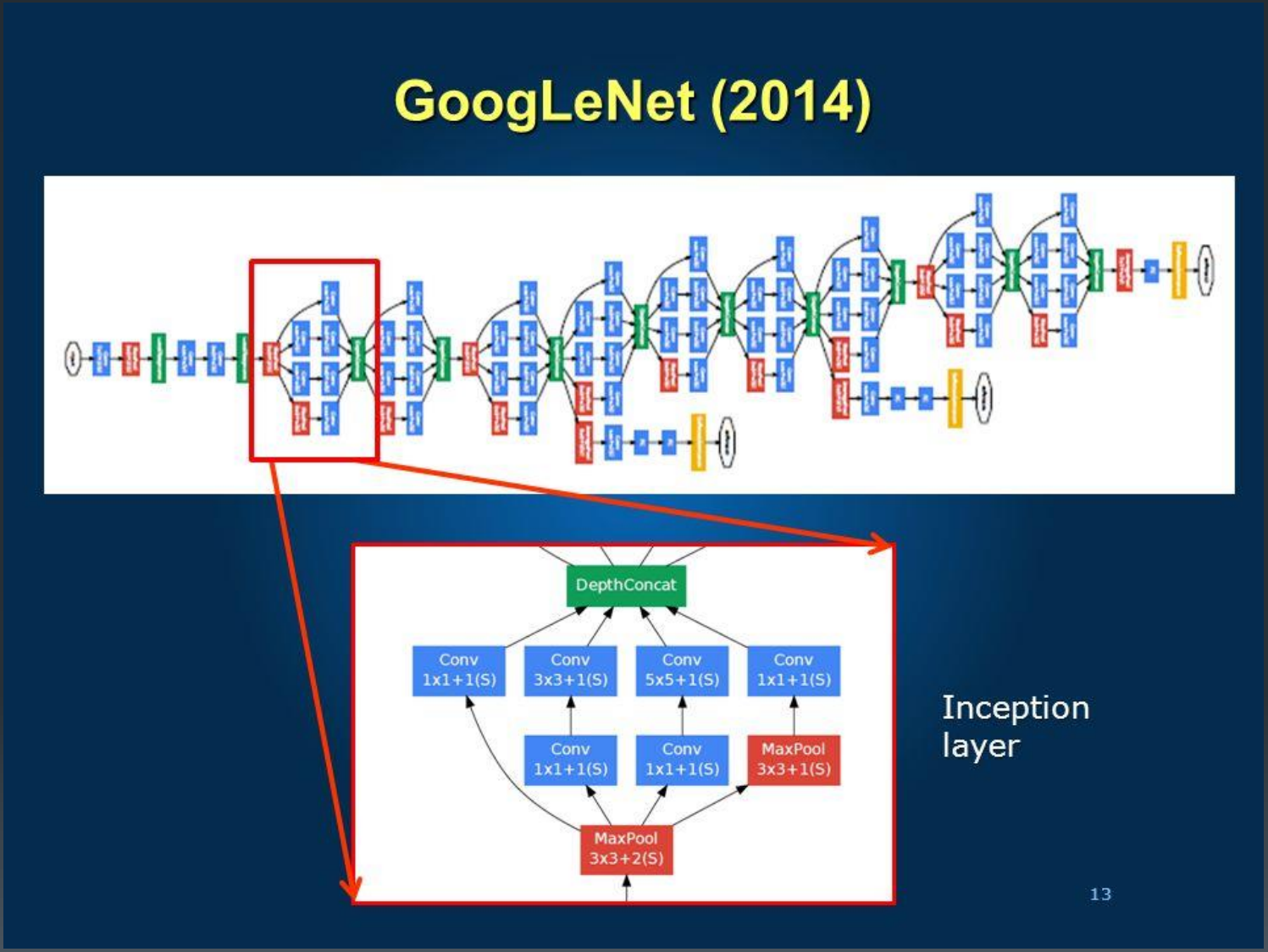


ν_μ

ν_e

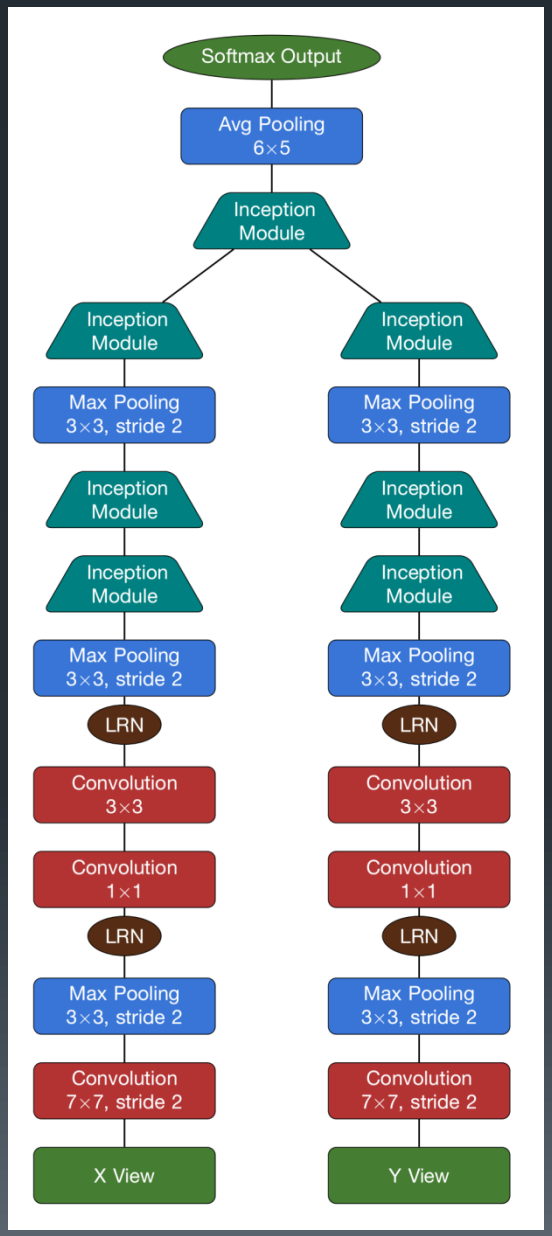
NC

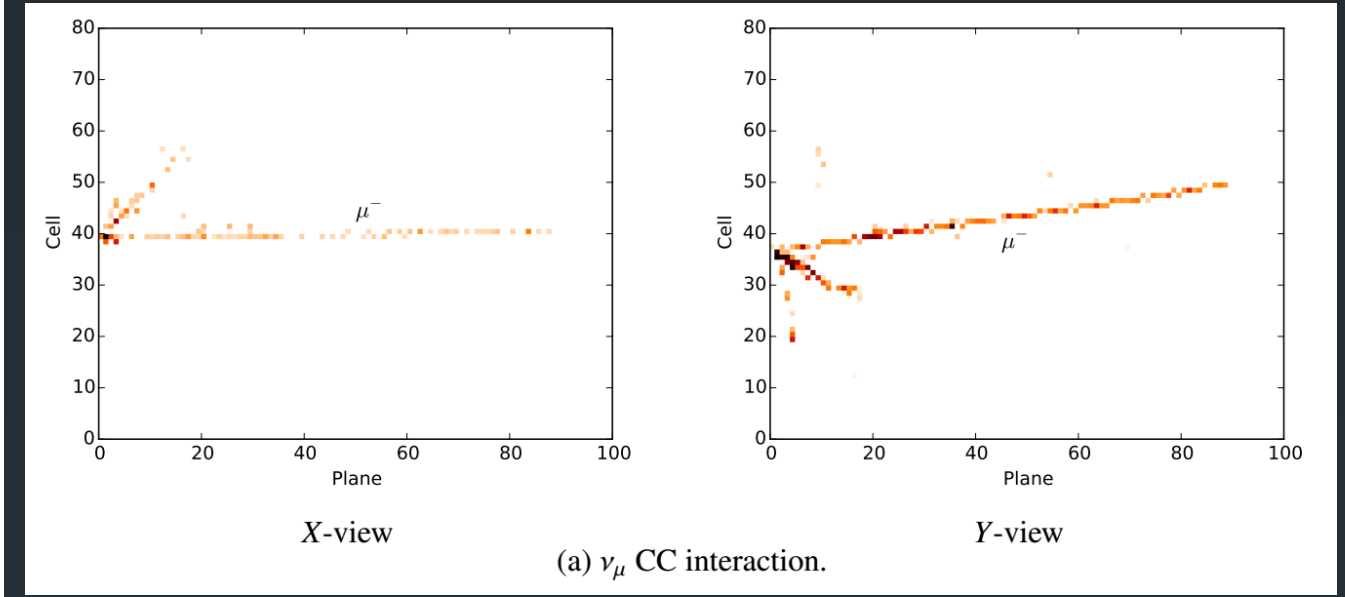
Network Inspiration



Inception modules are groups of Convolution/Pooling layers that include 1x1 Conv. to reduce the parameters in the network, allowing for deeper networks.

NOvA's Event Classification CNN

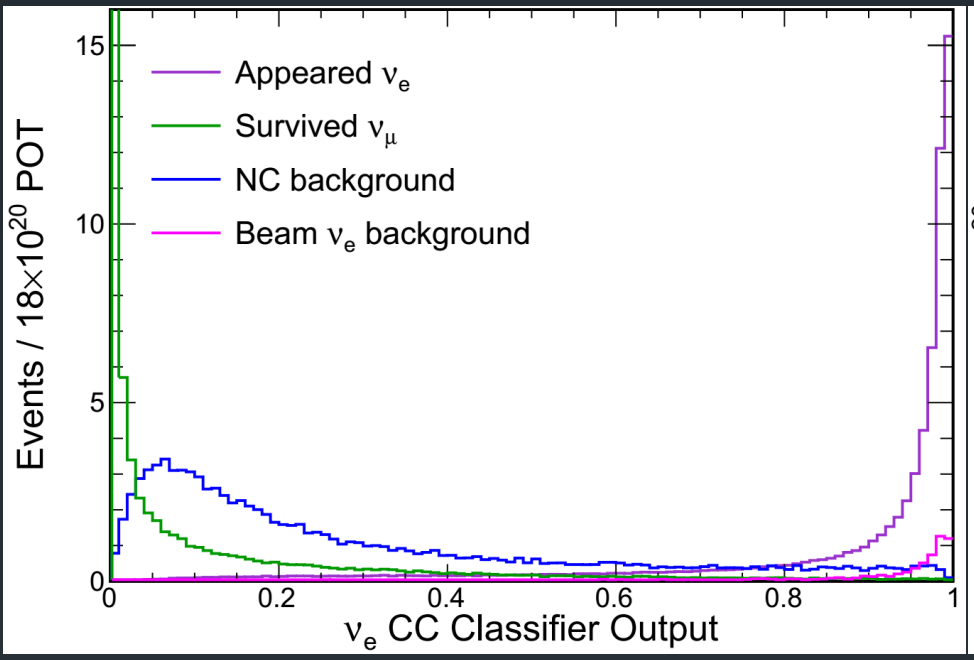


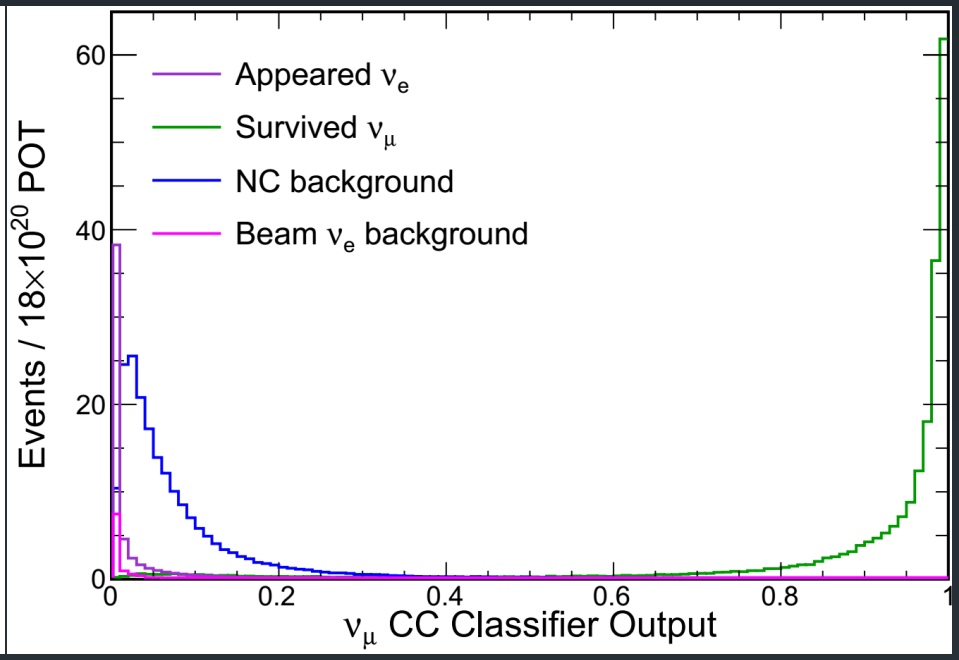


(a) ν_μ CC interaction.

- 80x100 matrix of energy deposits are extracted from the detector data. Greatly reduces data flow through network
- X and Y-views are sent separately through the two towers of NOvA's network
- The towers are merged, and the network outputs a value for each interaction type, which is Softmax normalized

Event Classification Network Results





Published. 2016

JINST 11 (2016) no.09, P09001

A Convolutional Neural Network Neutrino Event Classifier

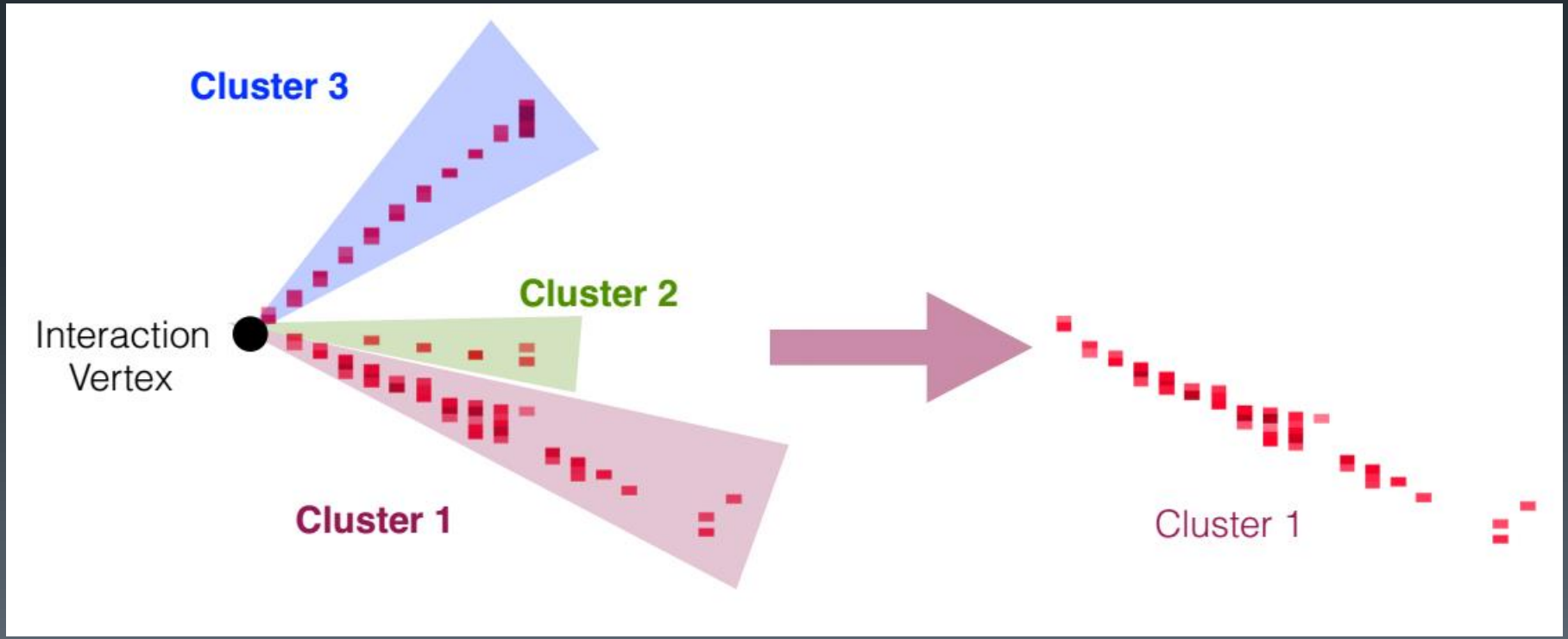
A. Aurisano, A. Radovic, D. Rocco, A. Himmel, M. D. Messier, E. Niner, G. Pawloski, F. Psihas, A. Sousa, P. Vahle

(Submitted on 5 Apr 2016 (v1), last revised 12 Aug 2016 (this version, v3))

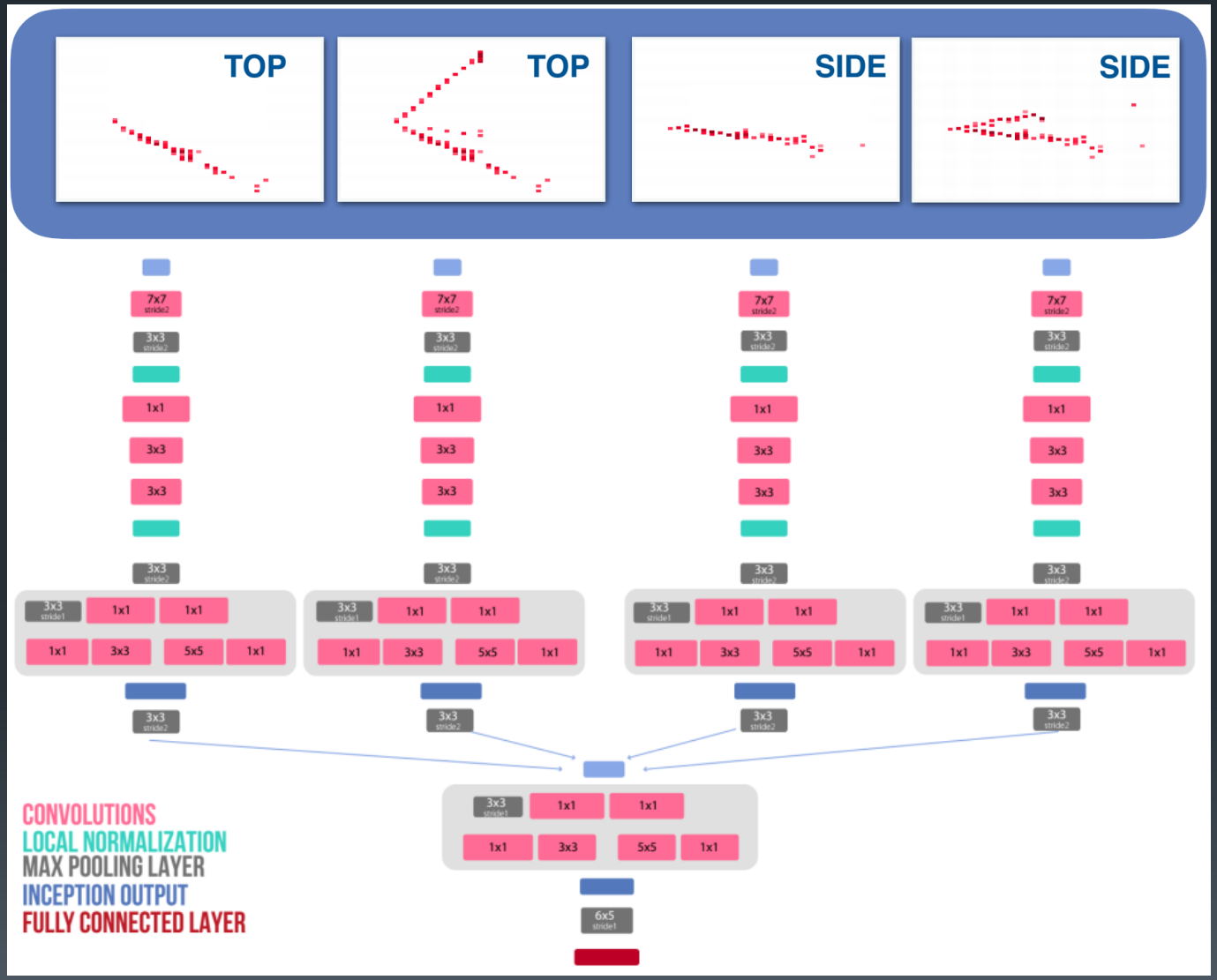
Particle Identification

Particle identification is necessary for in-depth physics analysis.

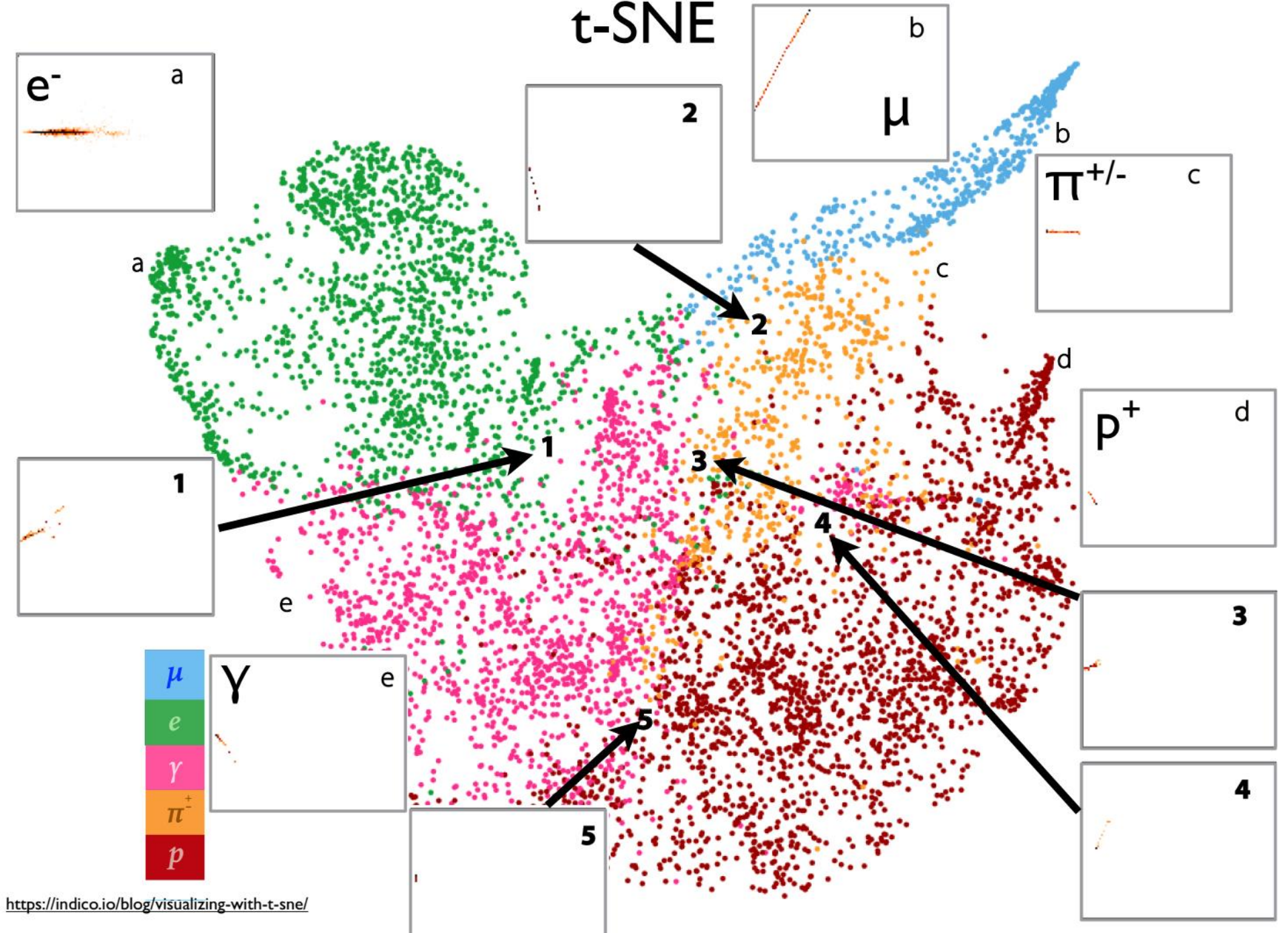
- Neutrino interactions in the detector are simulated
- Hits are clustered into tracks
- An interaction vertex is determined
- Single particle tracks are separated for training



Particle ID Network



Context provides up to an 11% improvement in efficiency and purity!



https://indico.io/blog/visualizing-with-t-sne/

More CNN work in NOvA

- Continue to improve event classification network and particle ID network
- NOvA has created a CNN to improve ν_e and electron energy estimation:
  PRD: DOI: 10.1103/PhysRevD.99.012011
- Creation of LSTM network to improve ν_μ energy estimation
- CNN to reduce cosmic ray background
- The NOvA test beam detector will provide labeled data from single-particle interactions allowing for data-driven checks of deep learning methods
- And more….

Problems with traditional NNs

- They require a huge amount of training data (whereas humans can infer from relatively small amount of initial information)
- They are a computationally heavy method (requiring large amounts of parallel processing and memory)
- They aren’t generally very adaptable, requiring retraining when new situations arise
- Constructing a suitable NN is a bit of an art form. There are guidelines, but NNs are not generally understood well enough to make rules based on first principles

Conclusions

- We have only looked at one type of NN: the feed-forward NN with supervised learning (labeled data)
- Neural Networks are not the only method of Machine Learning (Decision Trees, Regression methods, K-means…)
- Innovations happen all the time, as this is a young field!

Questions?