

Machine Learning for Neutrino Identification

Nitish Nayak
NuSTEAM Workshop

July 13th, 2023

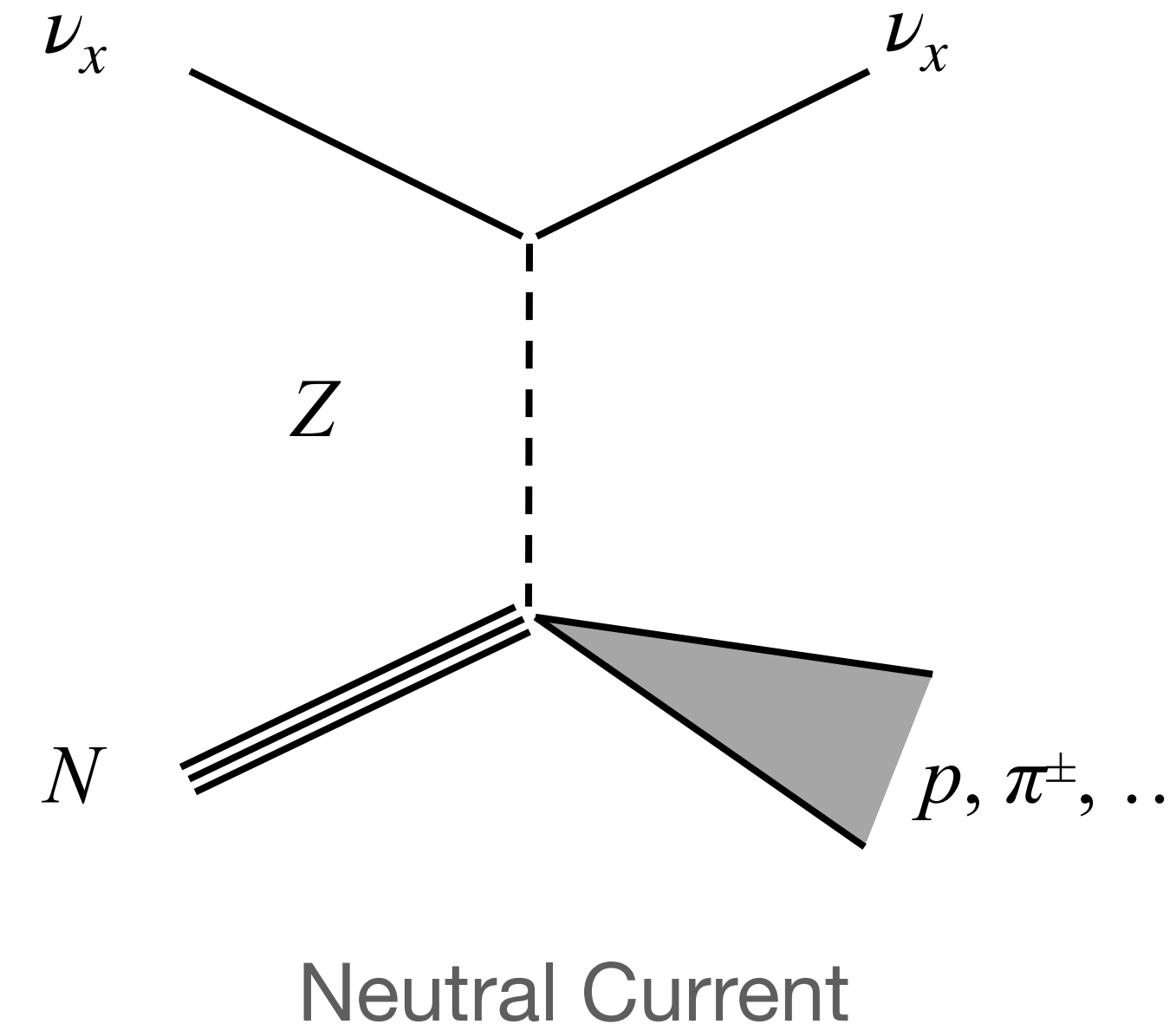
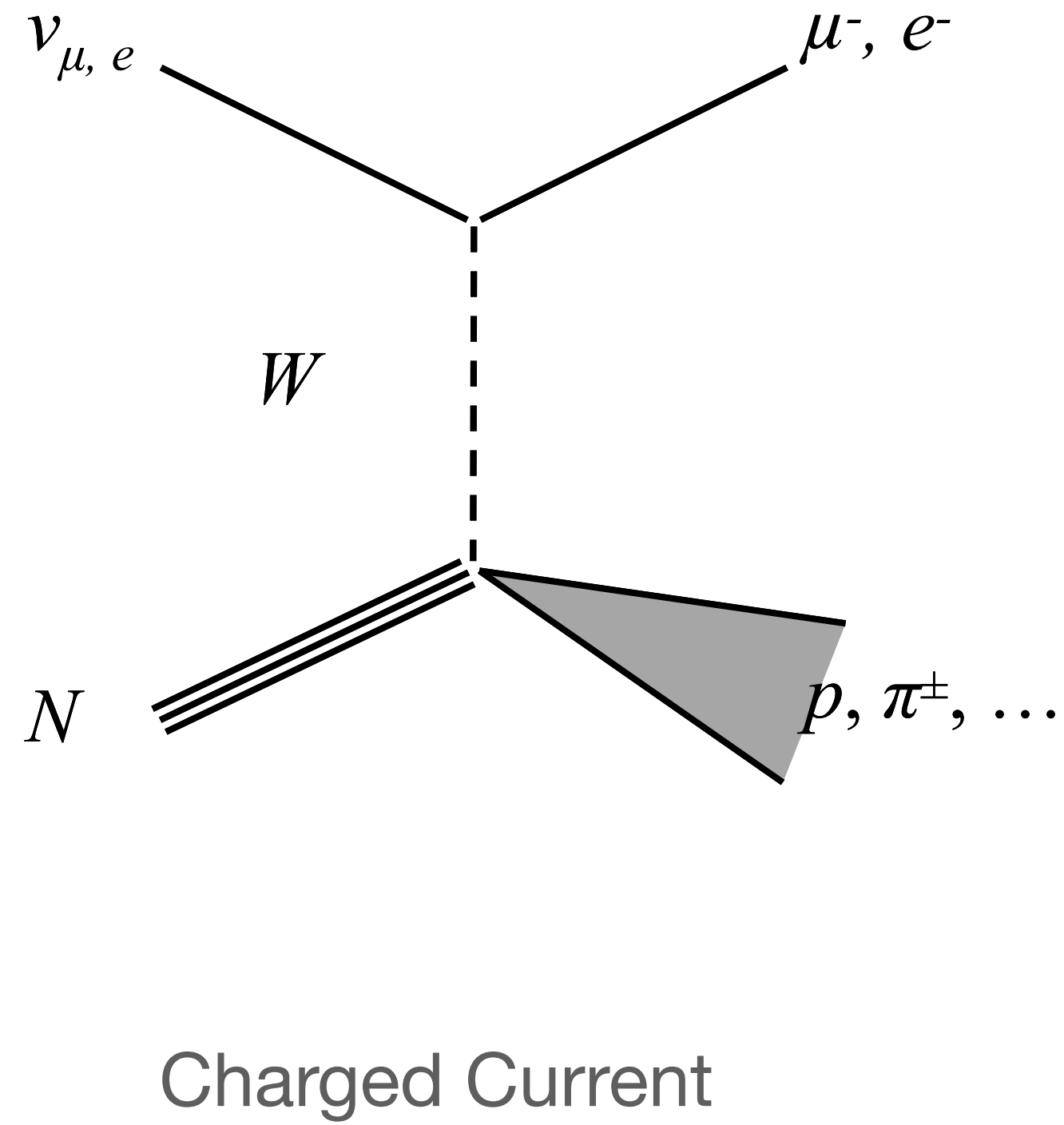
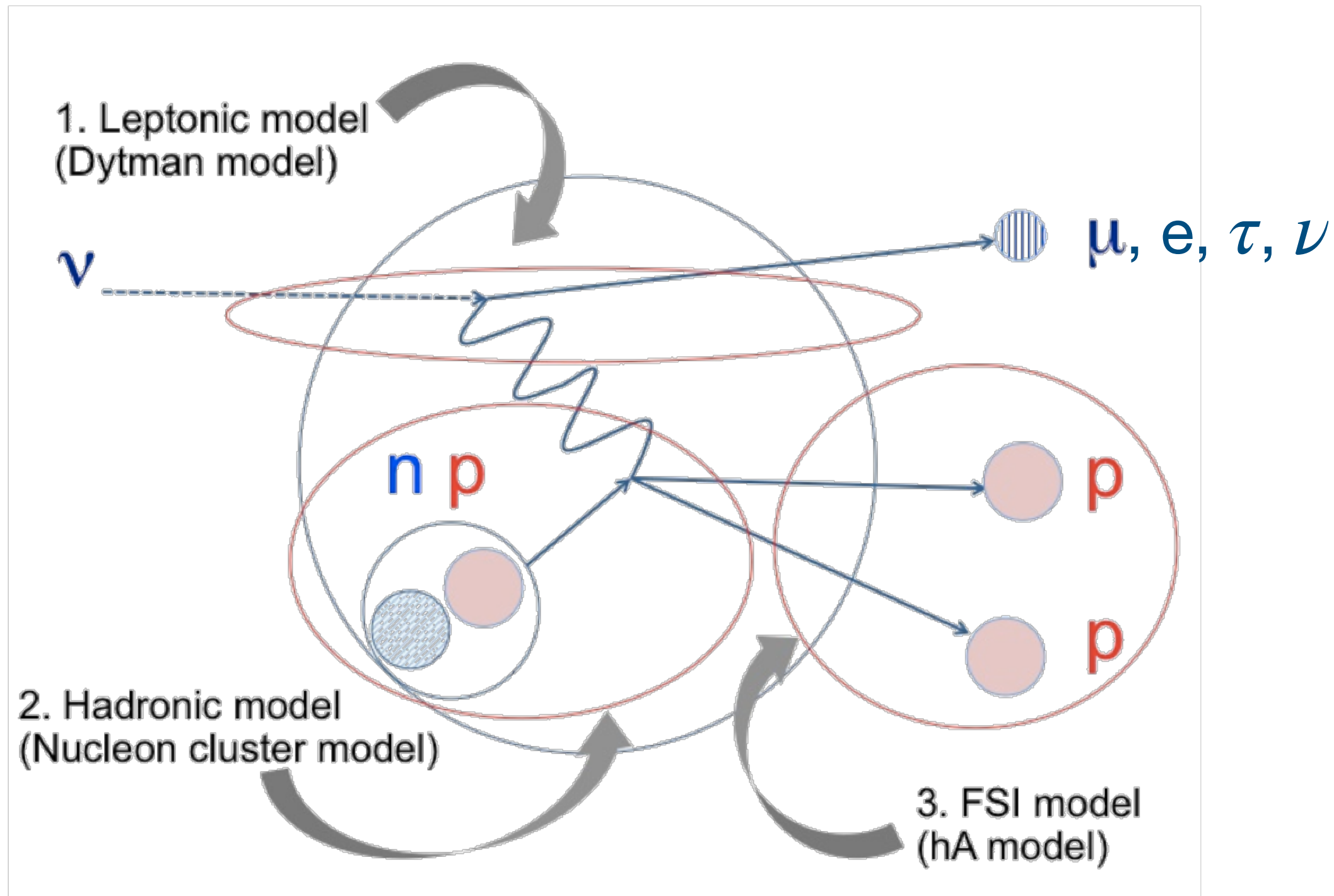
Prerequisites

- Types of Neutrino Interactions - ν_e CC , ν_μ CC, ν_τ CC, ν_x NC
- LArTPC detectors
- What neutrino interactions look like
- Why would we want to identify different types?

Outline

- Curve fitting/Classification/Regression
- Gentle introduction to Neural Networks
- Predicting type of neutrino interactions in LArTPCs
- Hands-on demo

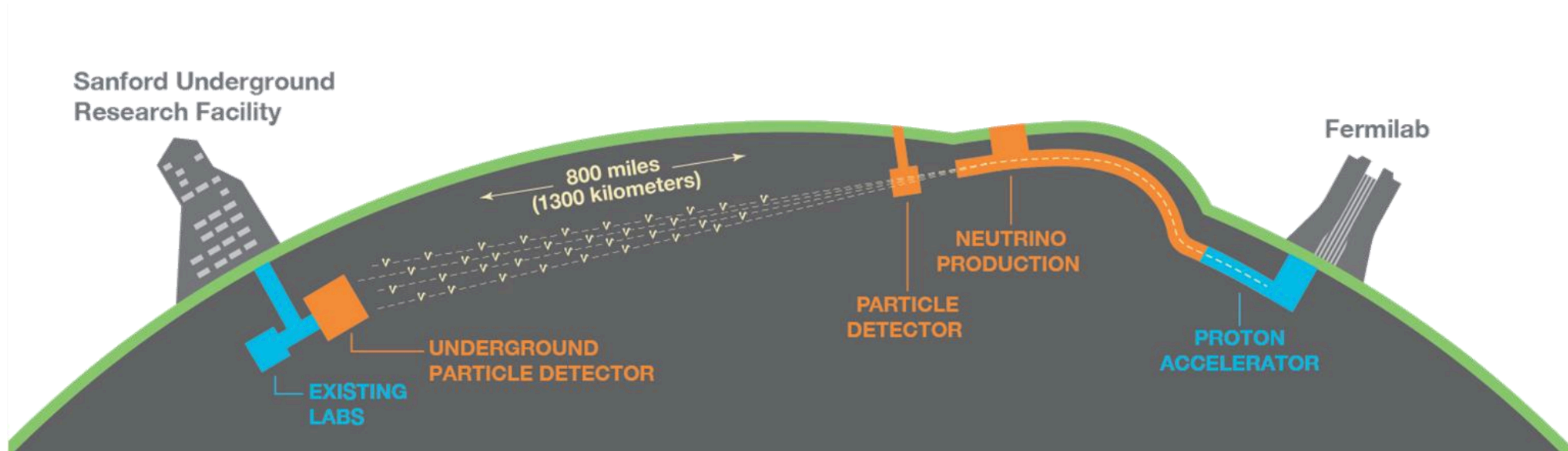
Types of ν -Interactions



- Different interactions look characteristically different
- Hadronic output (coming from the nucleus) broadly similar
- Interactions mainly differ in the nature of the final-state lepton
- Presence or absence of detected lepton tells us about type of interaction

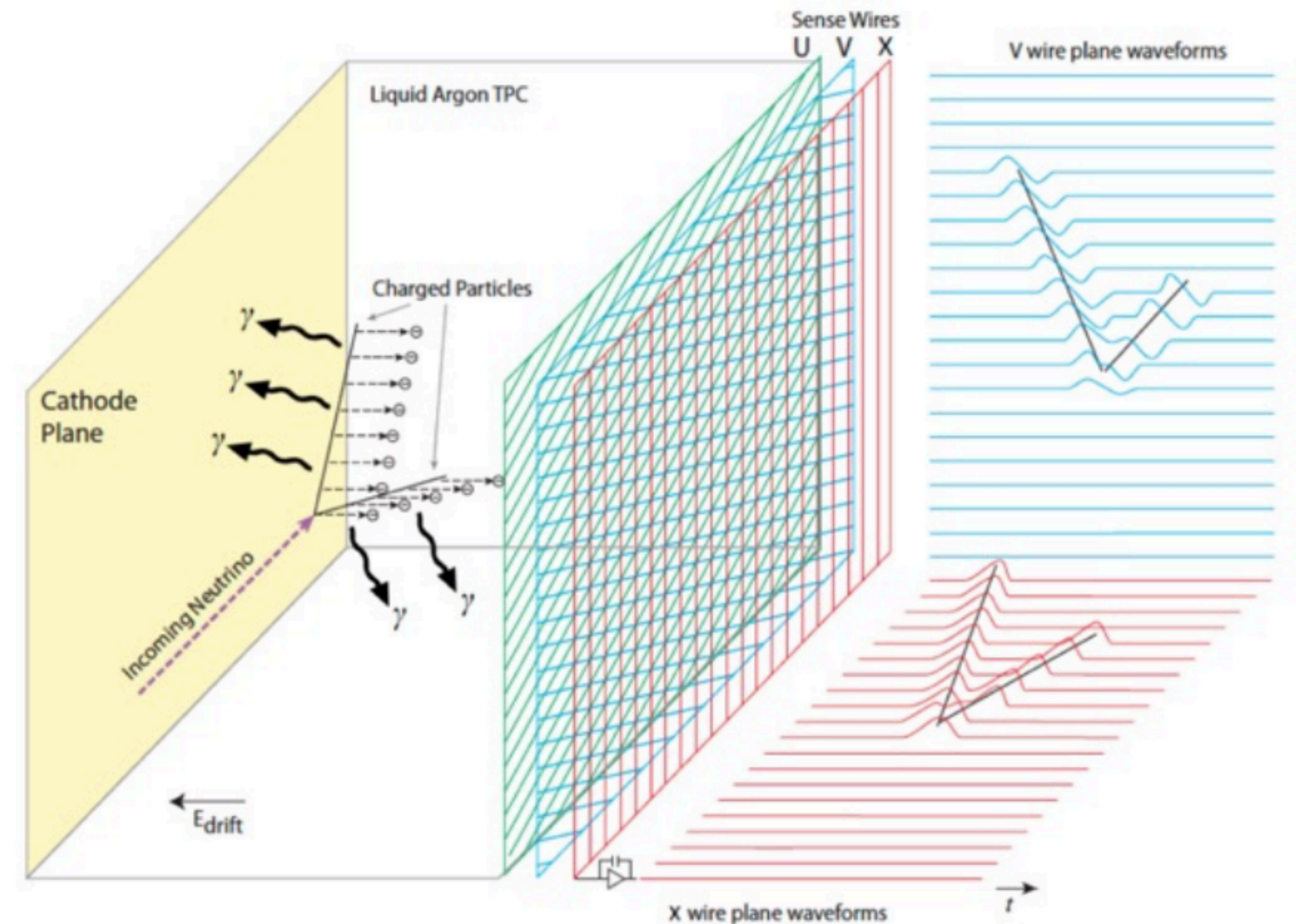
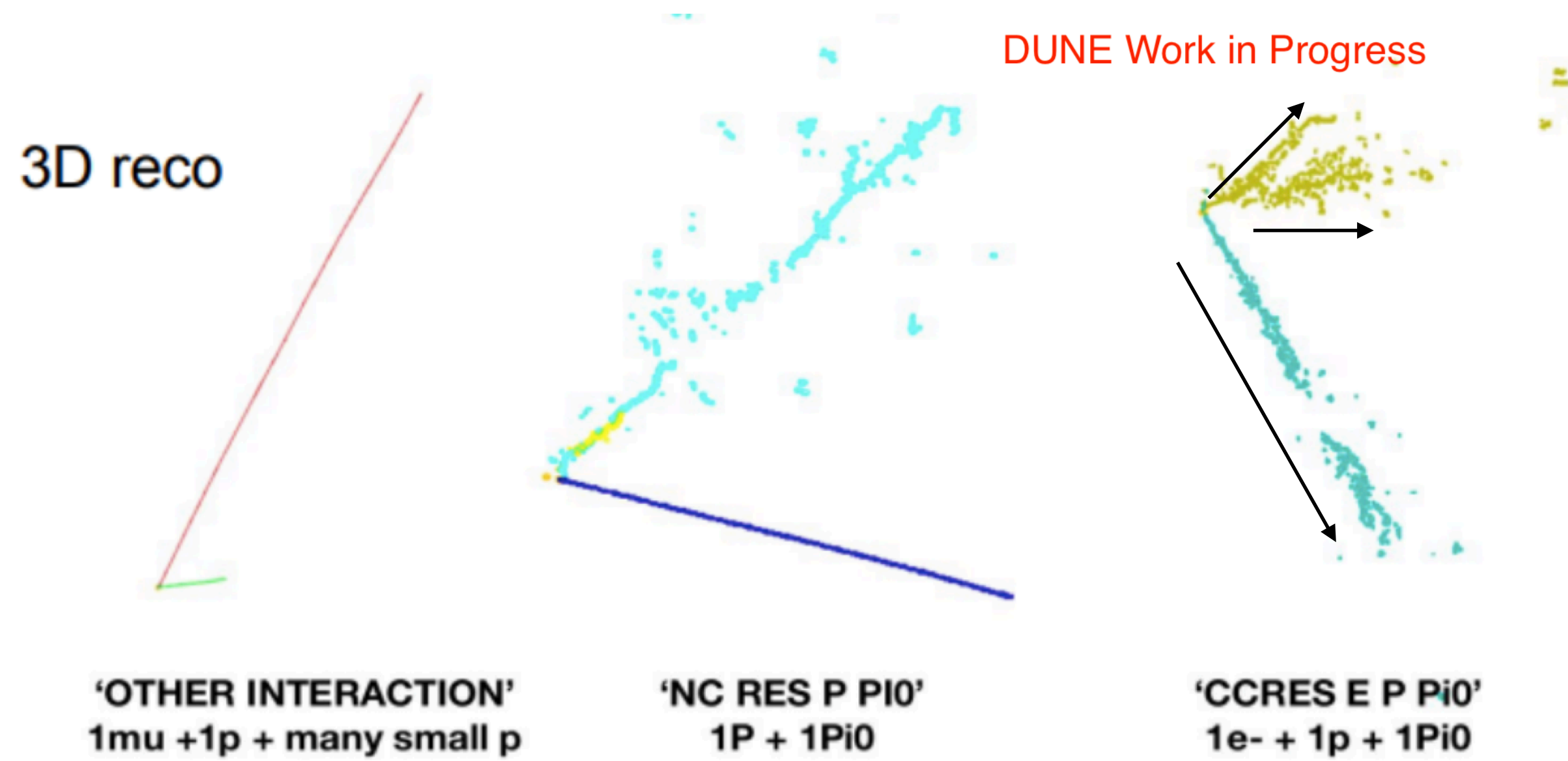
Identifying Neutrinos

- Fundamental objective of any neutrino oscillation experiment
- Neutrinos change flavor, tagging flavor is crucial to understand what's happening
- We have an idea of how many ν_μ , ν_e we start off with from beam, identify and count how many there at FD => measure probability

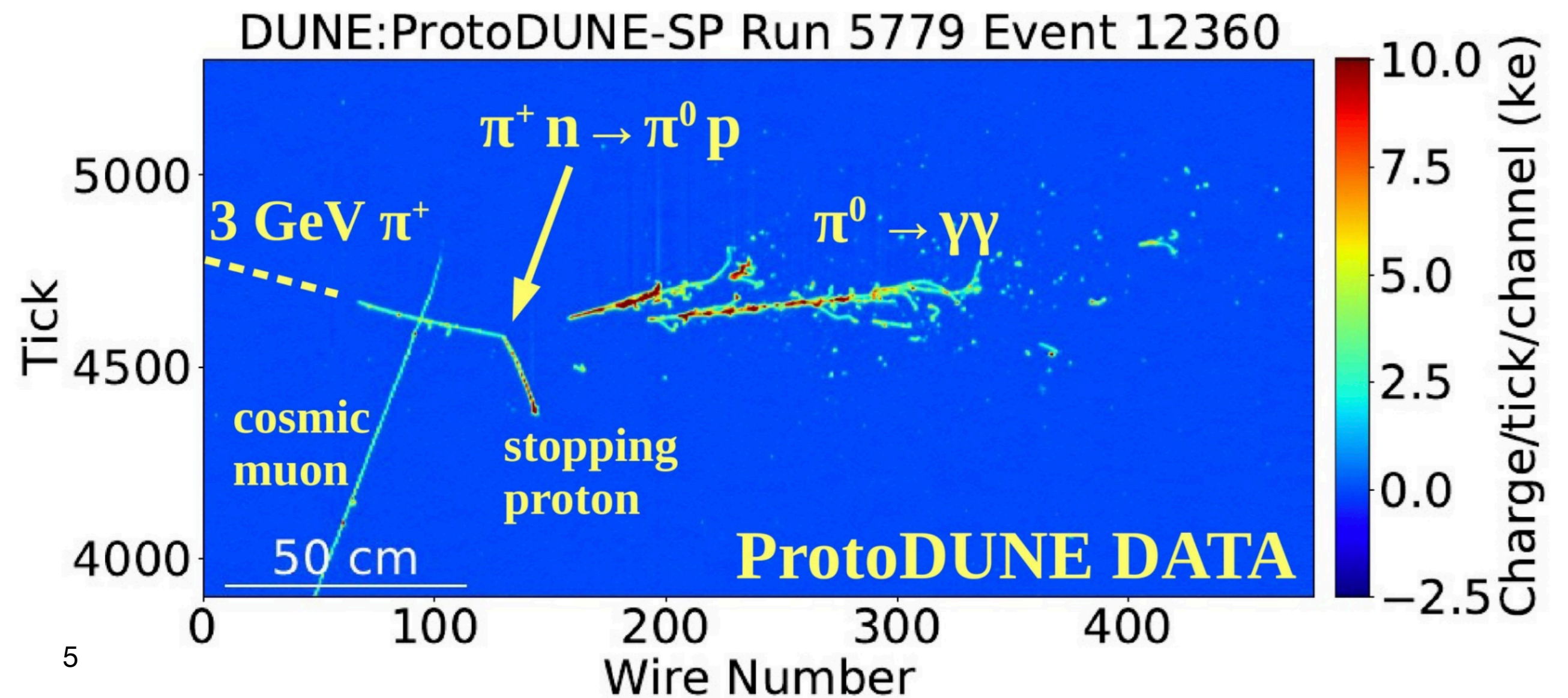


$$P(\nu_\alpha \rightarrow \nu_\beta) = \delta_{\alpha\beta} - 4 \sum_{i>j}^3 \Re(U_{\alpha i}^* U_{\beta i} U_{\alpha j} U_{\beta j}^*) \sin^2\left(\frac{\Delta m_{ij}^2 L}{4E_\nu}\right) + 2 \sum_{i>j}^3 \Im(U_{\alpha i}^* U_{\beta i} U_{\alpha j} U_{\beta j}^*) \sin\left(\frac{\Delta m_{ij}^2 L}{4E_\nu}\right)$$

ν -Interactions in LArTPCs

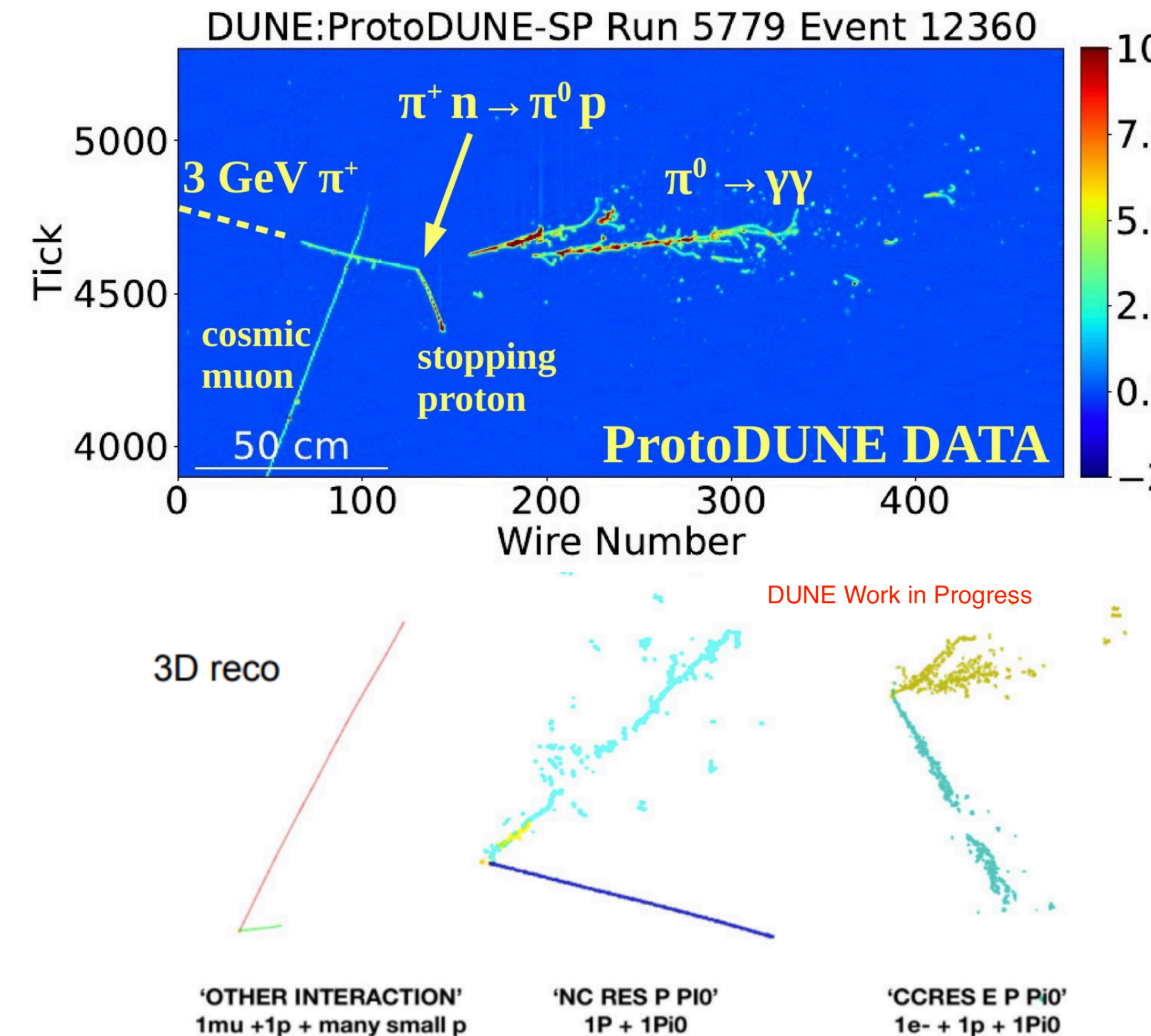


- Broadly track-like or shower-like
- LArTPCs give us exquisite detail
- But mistakes can easily be made!



Issues to deal with

- Neutrino interactions are messy at our energies!
- A decision flow can be something like this :
 - If you identify a μ (usually a long, straight track) $\Rightarrow \nu_{\mu}CC$
 - If you identify an e (shower like) $\Rightarrow \nu_eCC$
 - If you don't see either $\Rightarrow NC$
- But :
 - Sometimes don't see two γ 's from π^0 (one is short and buried in the other), can be confused as e
 - π^{\pm} are often long and straight too, can be confused as μ
 - Other messy stuff
- We need automated tools to tell them apart! Can't sift through each one manually..



Curve Fitting

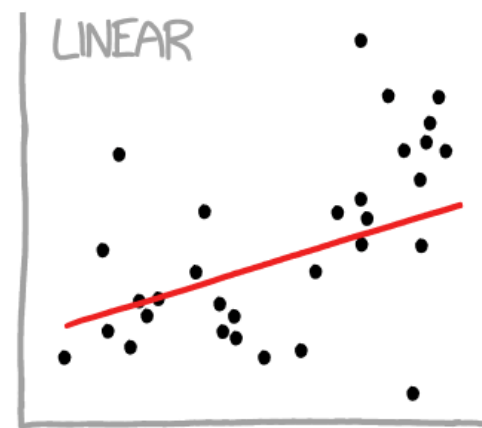
CURVE-FITTING METHODS AND THE MESSAGES THEY SEND

<https://xkcd.com/2048/>

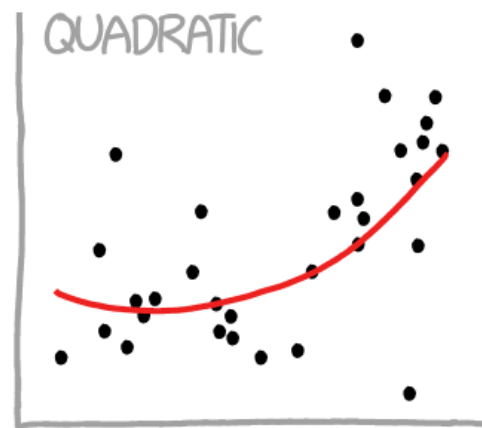
- Teasing out relationships between observables
- Data is usually always noisy
- Often have to make assumptions about what its supposed to look like : linear, quadratic, something fancy etc.
- Not always easy to justify!

Parametric Fitting

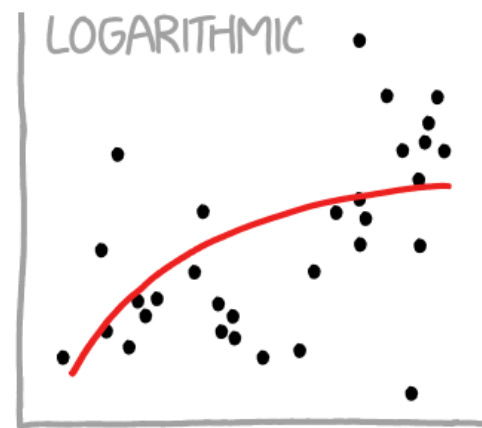
- $y = f(\vec{x}, \vec{\theta})$
 - Eg : $y = ax^2 + bx + c$, where $\vec{\theta} \equiv (a, b, c)$
- Choose $\vec{\theta}$ based on some measure denoting how good/bad the fit is (“loss function”)
 - Eg : $\vec{\theta} : \min |\hat{y} - f(\vec{x}, \vec{\theta})|^2$ (“least square distance”)
- Overfitting :
 - Can be caused by having too many parameters (d.o.f) describing low-dimensional data
 - Fits well to given noisy data but doesn’t generalize!



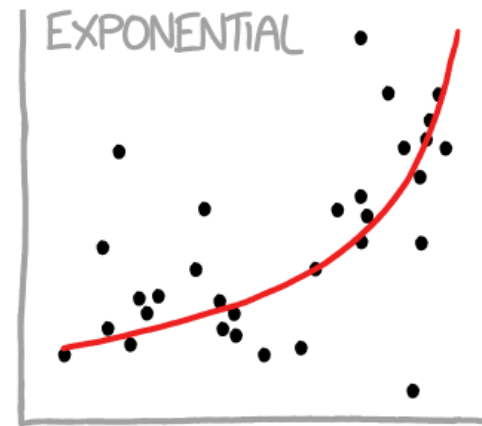
LINEAR
"HEY, I DID A REGRESSION."



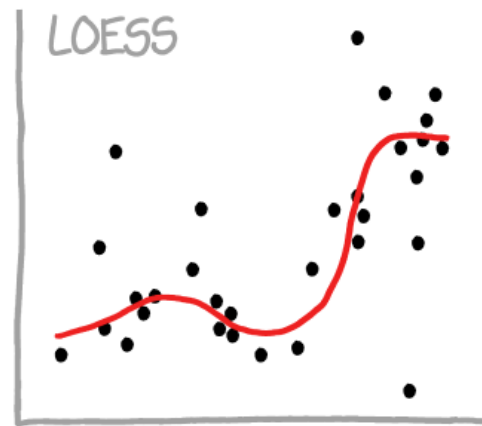
QUADRATIC
"I WANTED A CURVED LINE, SO I MADE ONE WITH MATH."



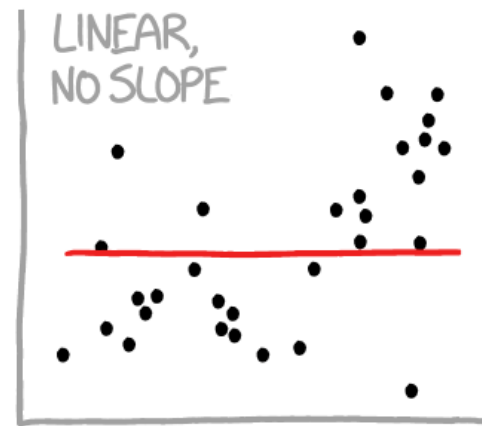
LOGARITHMIC
"LOOK, IT'S TAPERING OFF!"



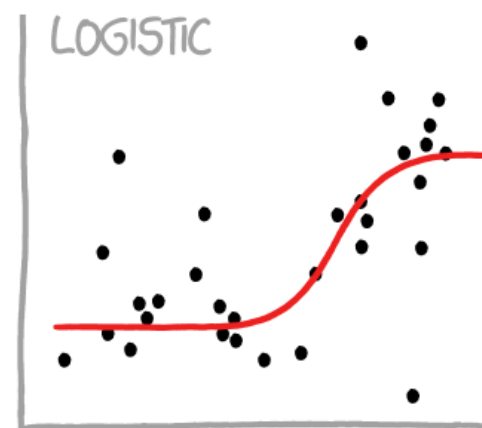
EXPONENTIAL
"LOOK, IT'S GROWING UNCONTROLLABLY!"



LOESS
"I'M SOPHISTICATED, NOT LIKE THOSE BUMBLING POLYNOMIAL PEOPLE."



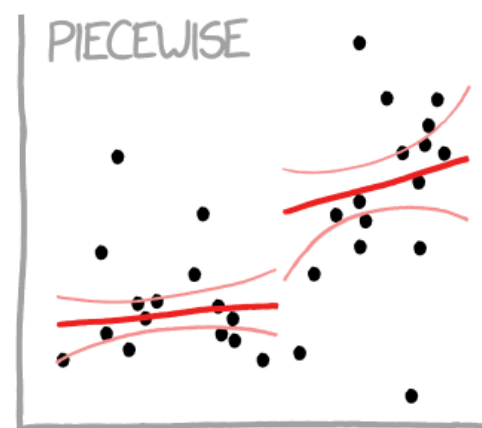
LINEAR, NO SLOPE
"I'M MAKING A SCATTER PLOT BUT I DON'T WANT TO."



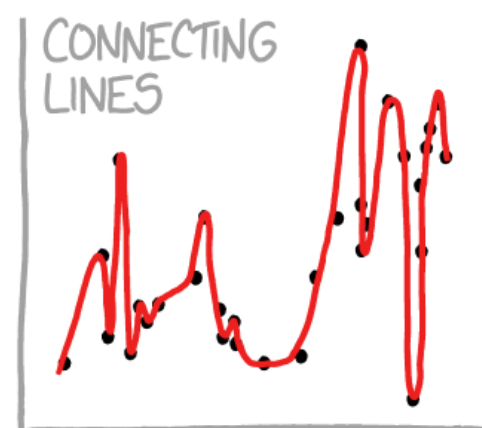
LOGISTIC
"I NEED TO CONNECT THESE TWO LINES, BUT MY FIRST IDEA DIDN'T HAVE ENOUGH MATH."



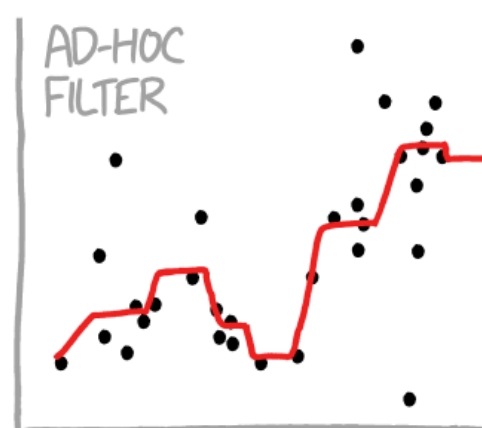
CONFIDENCE INTERVAL
"LISTEN, SCIENCE IS HARD. BUT I'M A SERIOUS PERSON DOING MY BEST."



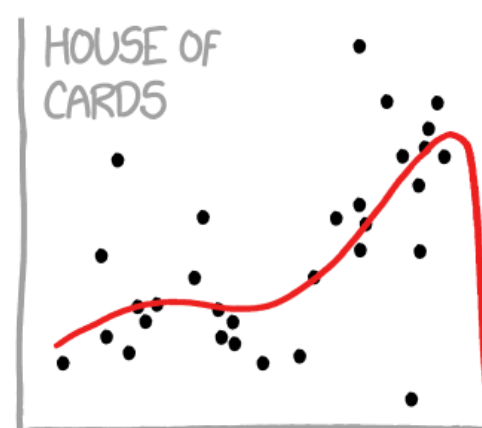
PIECEWISE
"I HAVE A THEORY, AND THIS IS THE ONLY DATA I COULD FIND."



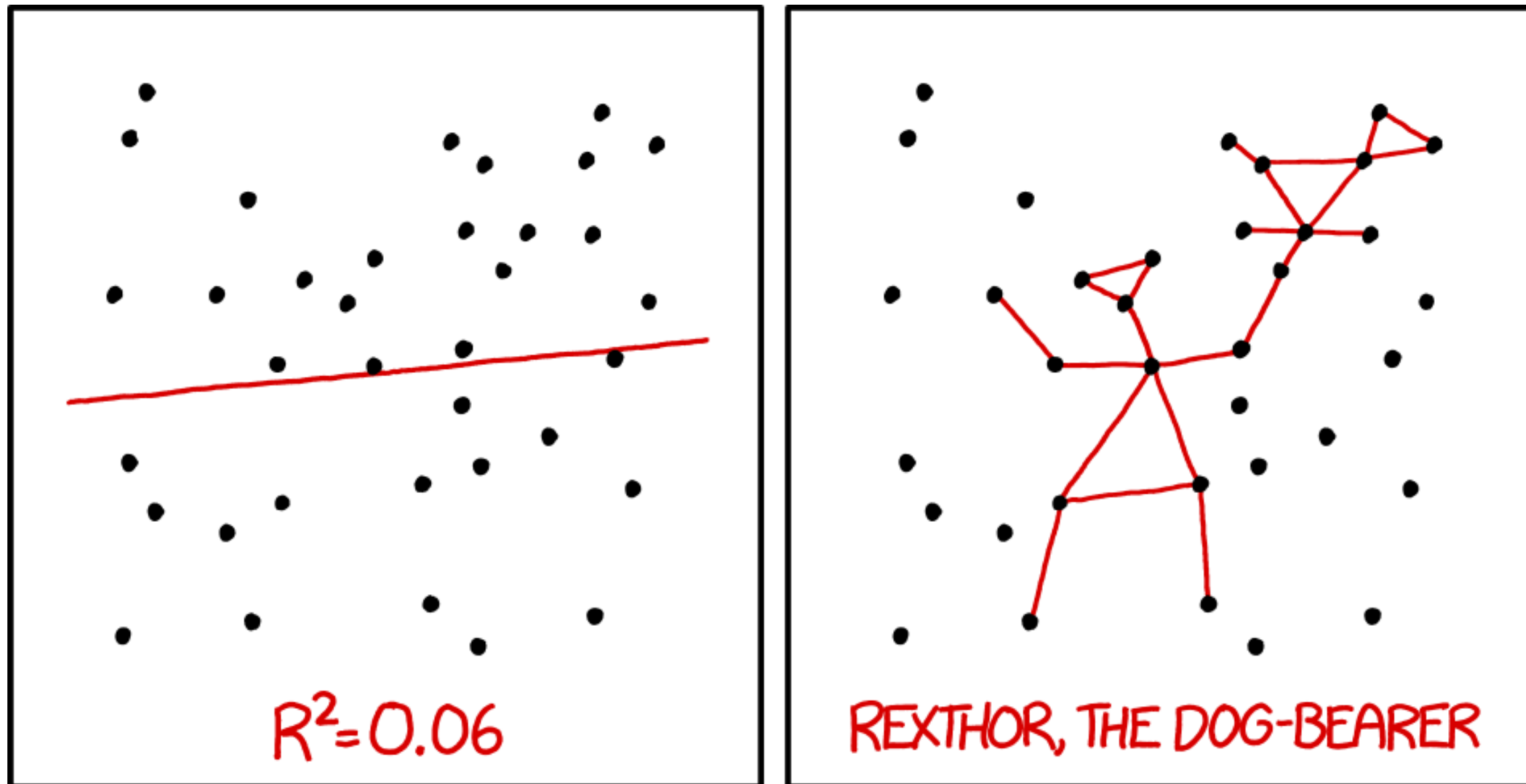
CONNECTING LINES
"I CLICKED 'SMOOTH LINES' IN EXCEL."



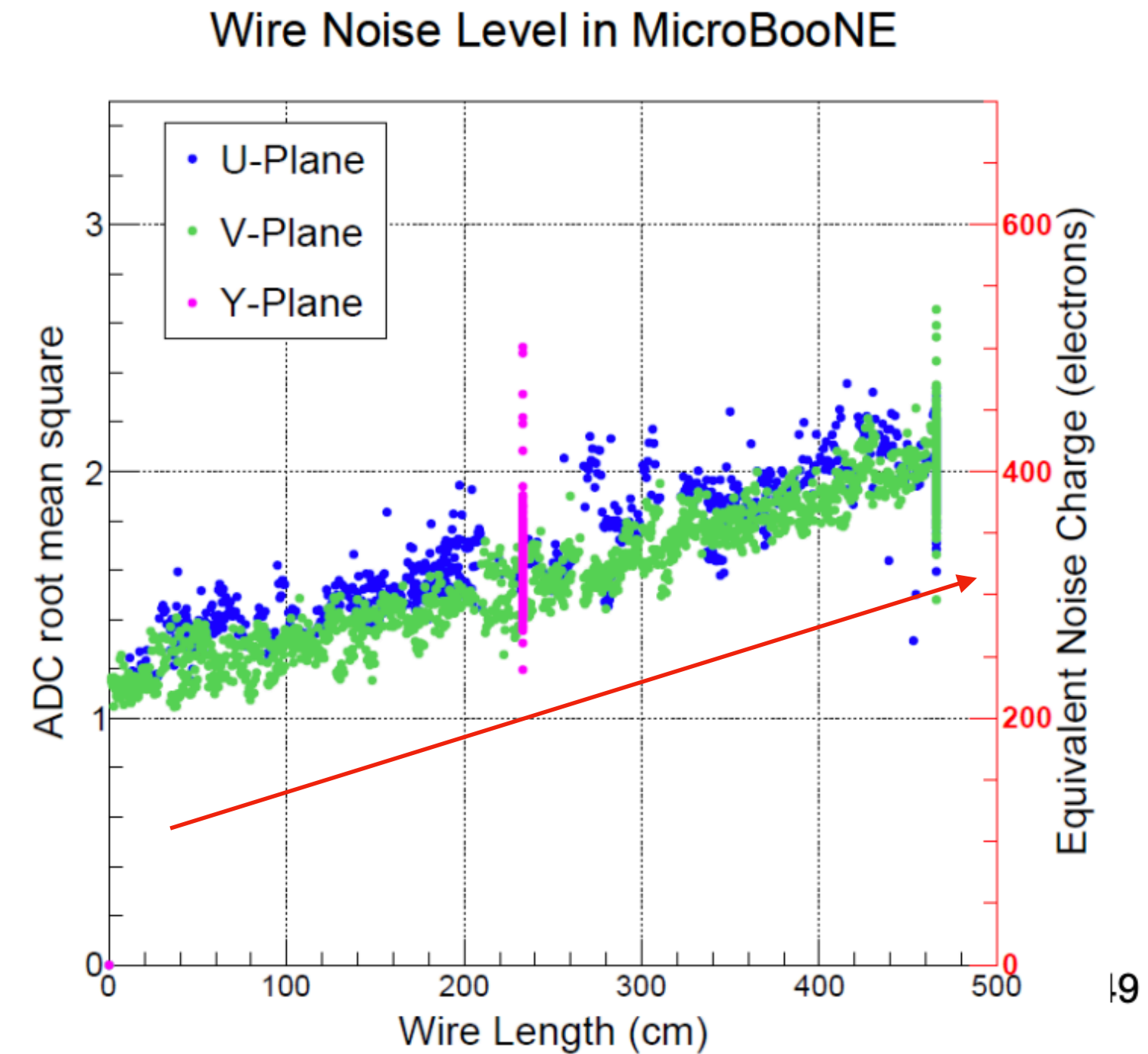
AD-HOC FILTER
"I HAD AN IDEA FOR HOW TO CLEAN UP THE DATA. WHAT DO YOU THINK?"



HOUSE OF CARDS
"AS YOU CAN SEE, THIS MODEL SMOOTHLY FITS THE- WAIT NO NO DON'T EXTEND IT AAAAAA!!"



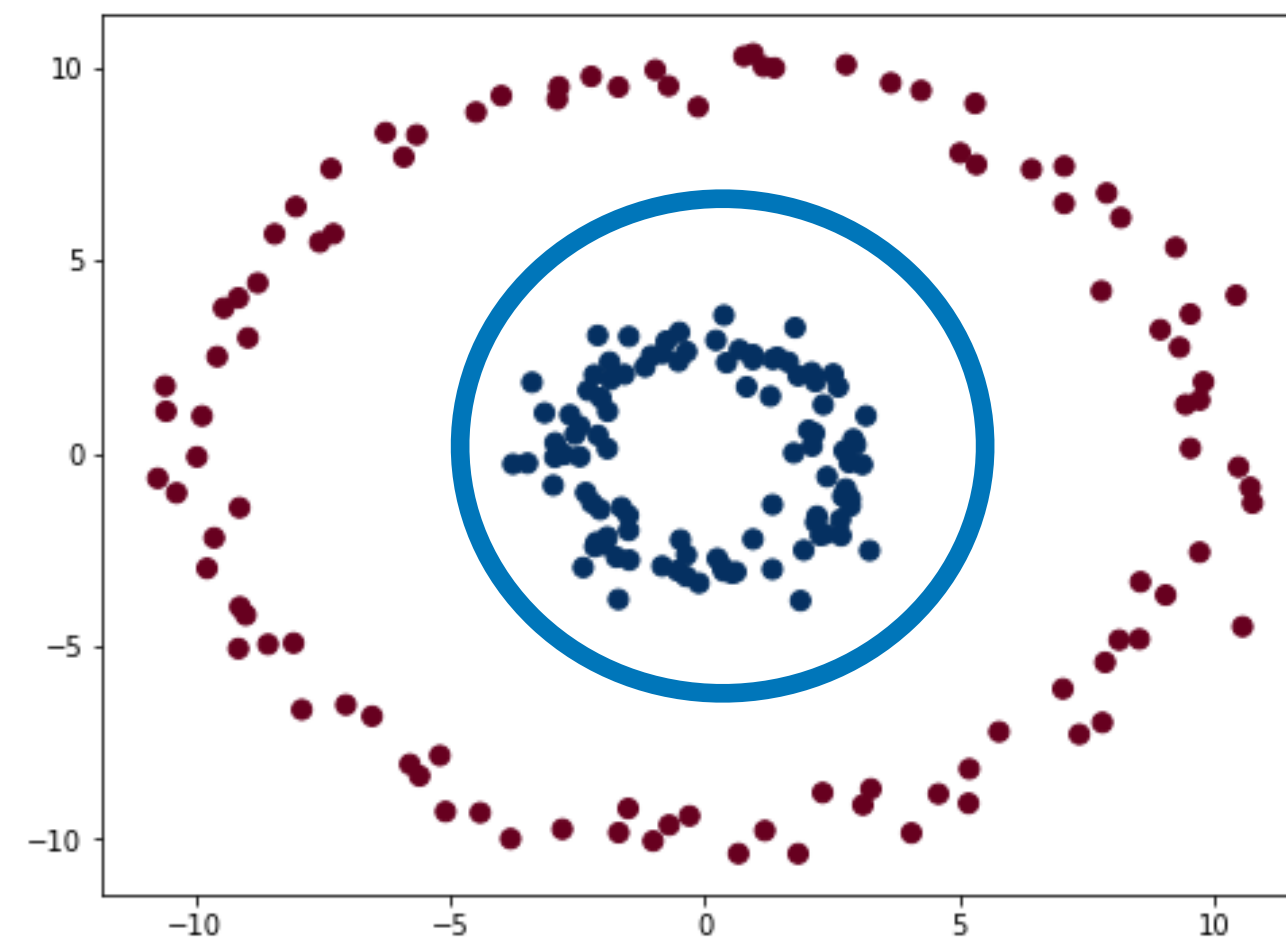
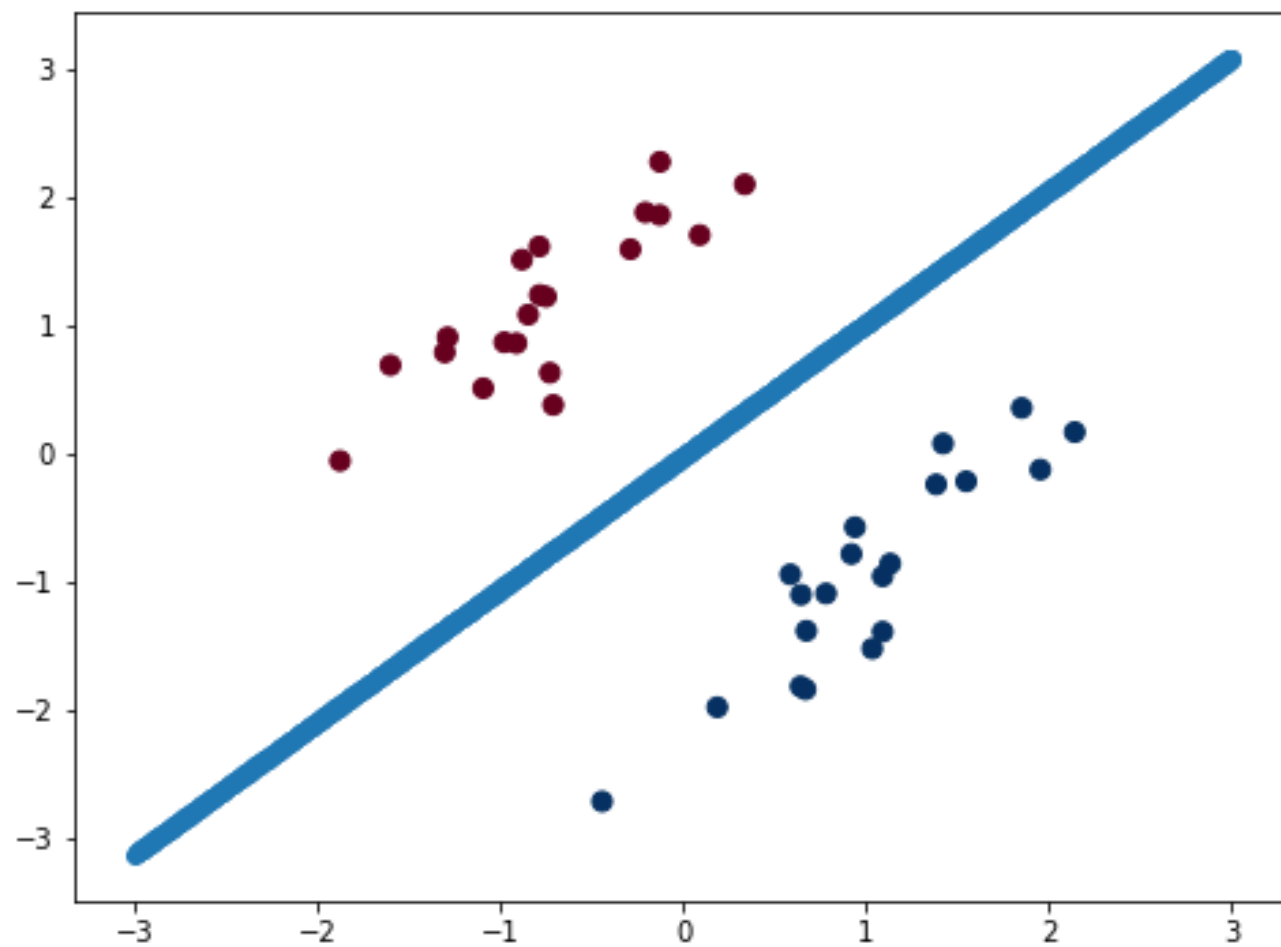
I DON'T TRUST LINEAR REGRESSIONS WHEN IT'S HARDER TO GUESS THE DIRECTION OF THE CORRELATION FROM THE SCATTER PLOT THAN TO FIND NEW CONSTELLATIONS ON IT.



- But how to choose parametric form? Need to carefully assess which one works for given problem under what assumptions
- No reason to believe a parametric form even exists in some cases (esp in higher dimensions)
 - Eg : Predict how movies are going to perform based on various inputs (choice of actors, marketing budget, script etc)
- Sophisticated techniques that are automatically able to “learn” what’s best. Can be very accurate
 - More complexity $\not\Rightarrow$ automatically better! Still prone to overfitting, biases, bad assumptions etc

Regression vs Classification

- Regression - Predict a continuous variable, for eg. the energy of the neutrino that interacted in the detector
 - Parametric methods : Find $\vec{\theta} : \min |\hat{y} - f(\vec{x}, \vec{\theta})|^2$: Choice of f can be pre-determined or automatically “learnt”
 - Complex problems require complex f , often very non-linear
- Classification - Predict one of many possible discrete labels, for eg. ν_e CC , ν_μ CC, ν_τ CC, ν_x NC
- Classification is also curve-fitting in a way!

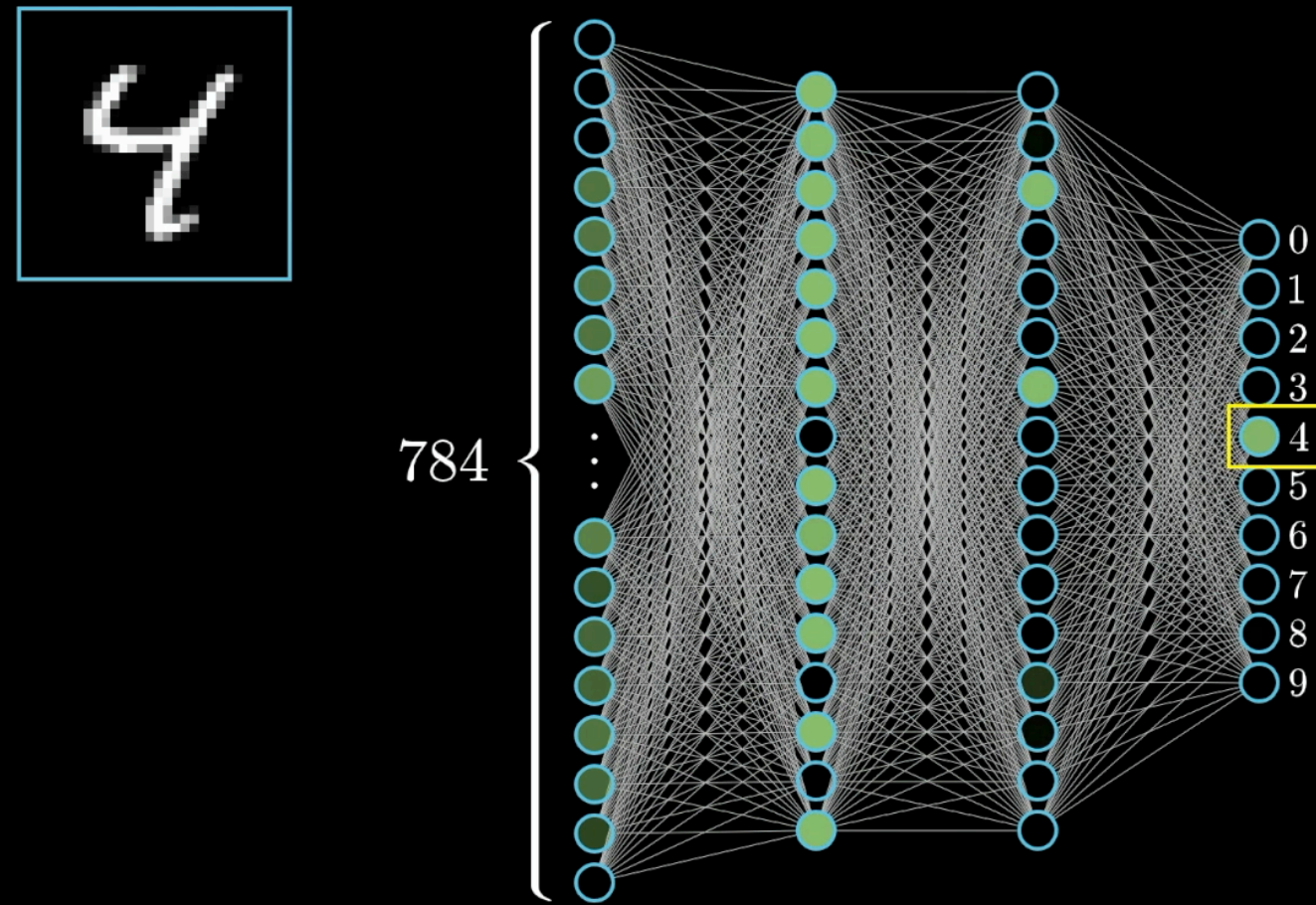


- Now, we want f and $\vec{\theta}$ that best discriminates between data coming from different labels
 - “Decision boundary”
- Essentially a regression problem for the probability of given data to belong to one label or the other
- Again, f can be relatively simple or highly complicated depending on context
- Involves another loss-function that minimizes “probability error”, can be least-squares as before

Neural Networks

Plain vanilla

(aka “multilayer perceptron”)



- Essentially devices that can spit out arbitrarily complex functions in many dimensions
- Network of “neurons” to mimic structure of human brain
- Consider for eg, input 28x28 (=784 pixels) image where each pixel has a number b/w (0, 1) denoting how bright that pixel is
- The neurons (1 for each pixel) in the first (“input”) layer can just be “brightness” values in that pixel

- For a neuron in the 2nd layer, calculate its response as :

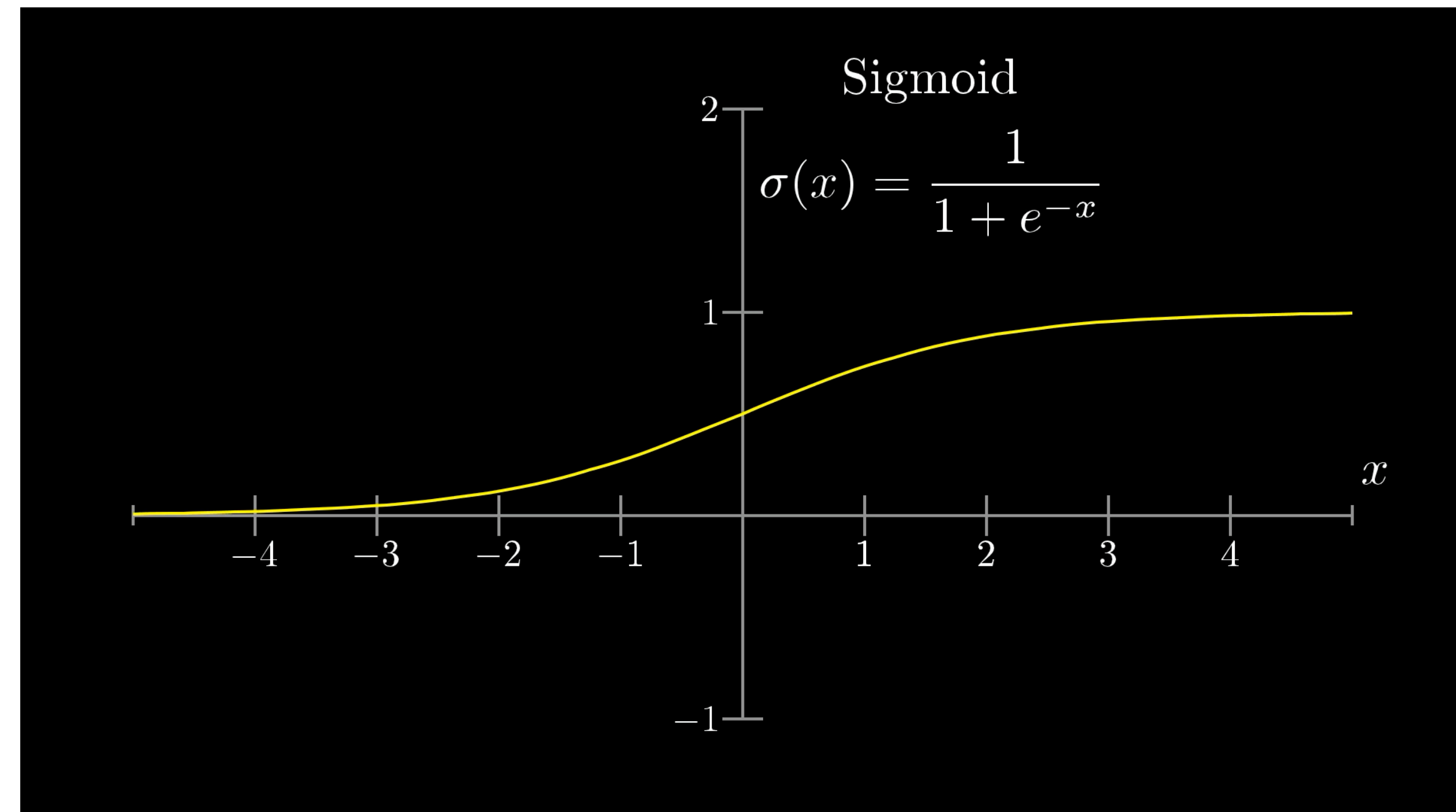
- $a_0^{(1)} = \sigma(w_{0,0}a_0^{(0)} + w_{0,1}a_1^{(0)} + w_{0,2}a_2^{(0)} + \dots w_{0,n}a_n^{(0)} + b_0)$

- Where $a_i^{(0)}$ represents the value of the i^{th} neuron in the 0^{th} layer

- $w_{0,i}$ represents the strength of the connection between $a_0^{(1)}$ and $a_i^{(0)}$ (“weights”)

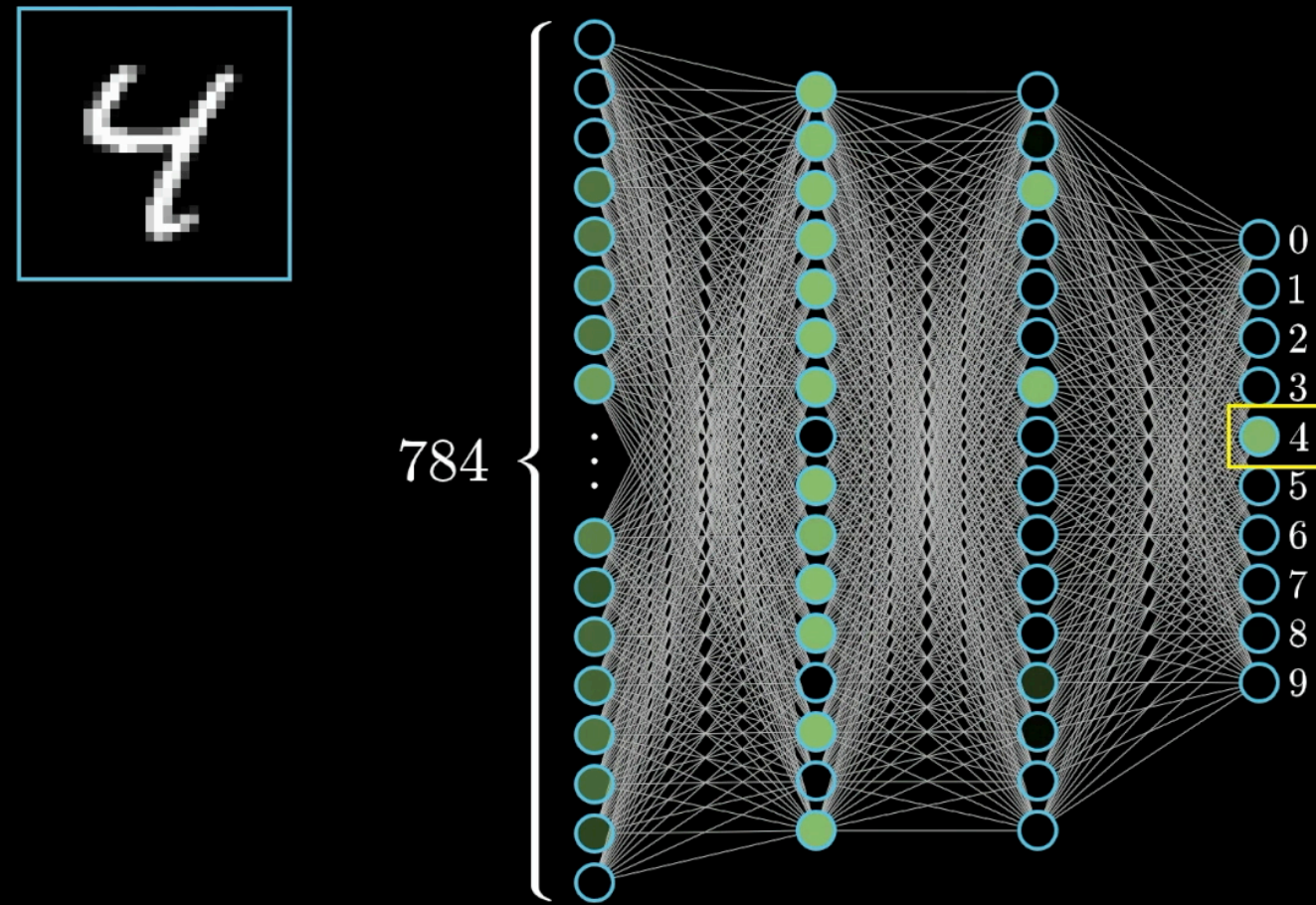
- b_0 is a bias parameter

- σ is a so-called activation function, designed to ensure values in each neuron are within a certain range, for eg between (0, 1) typically for classification



Plain vanilla

(aka “multilayer perceptron”)



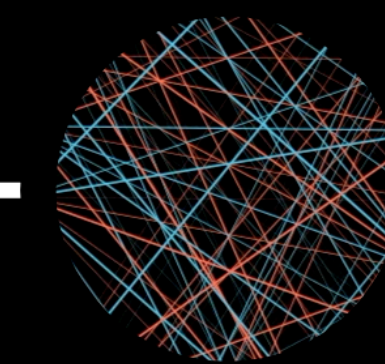
- In our eg, the final (“output”) layer has 10 neurons
 - 1 for each digit we want to predict (0-9)
- Lets assume 2 hidden layers, each with 16 neurons and image w/ 784 pixels

- $\Rightarrow (784 \cdot 16 + 16 \cdot 16 + 16 \cdot 10) = 12960$ w parameters (“weights”)
- $\Rightarrow (16 + 16 + 10) = 42$ b parameters (“biases”)
- Total = 13002 parameters
- Find w and b values such that we get the best predictions
- Curve fitting in 13002 dimensions!

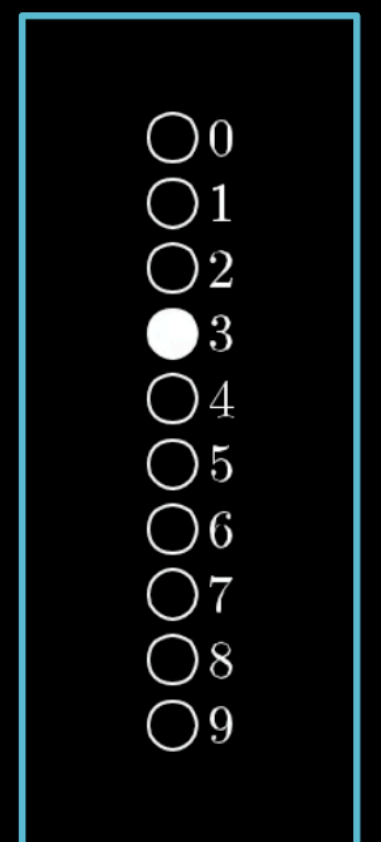
Neural network function



784 inputs



13,002 weights
and biases

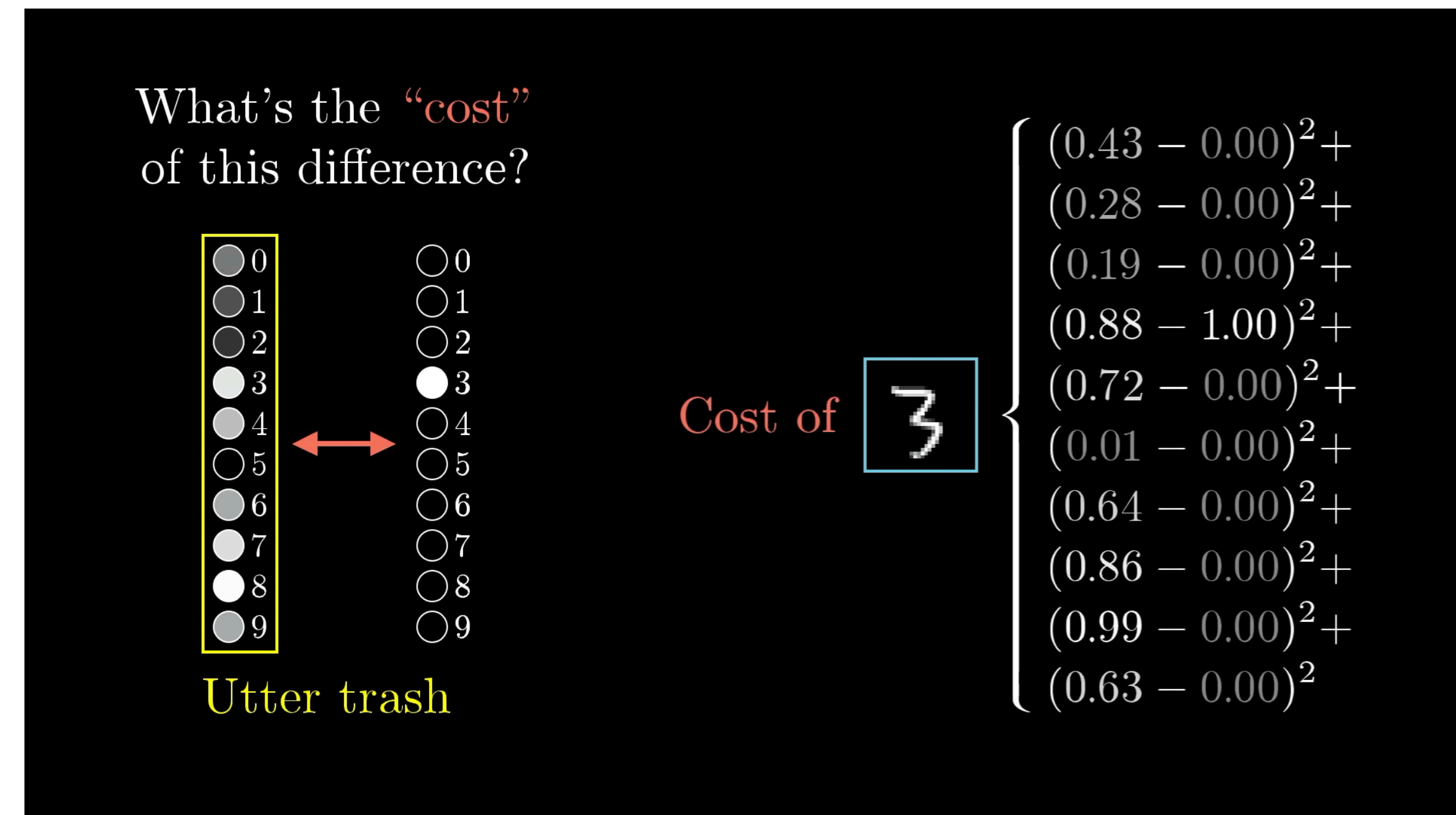
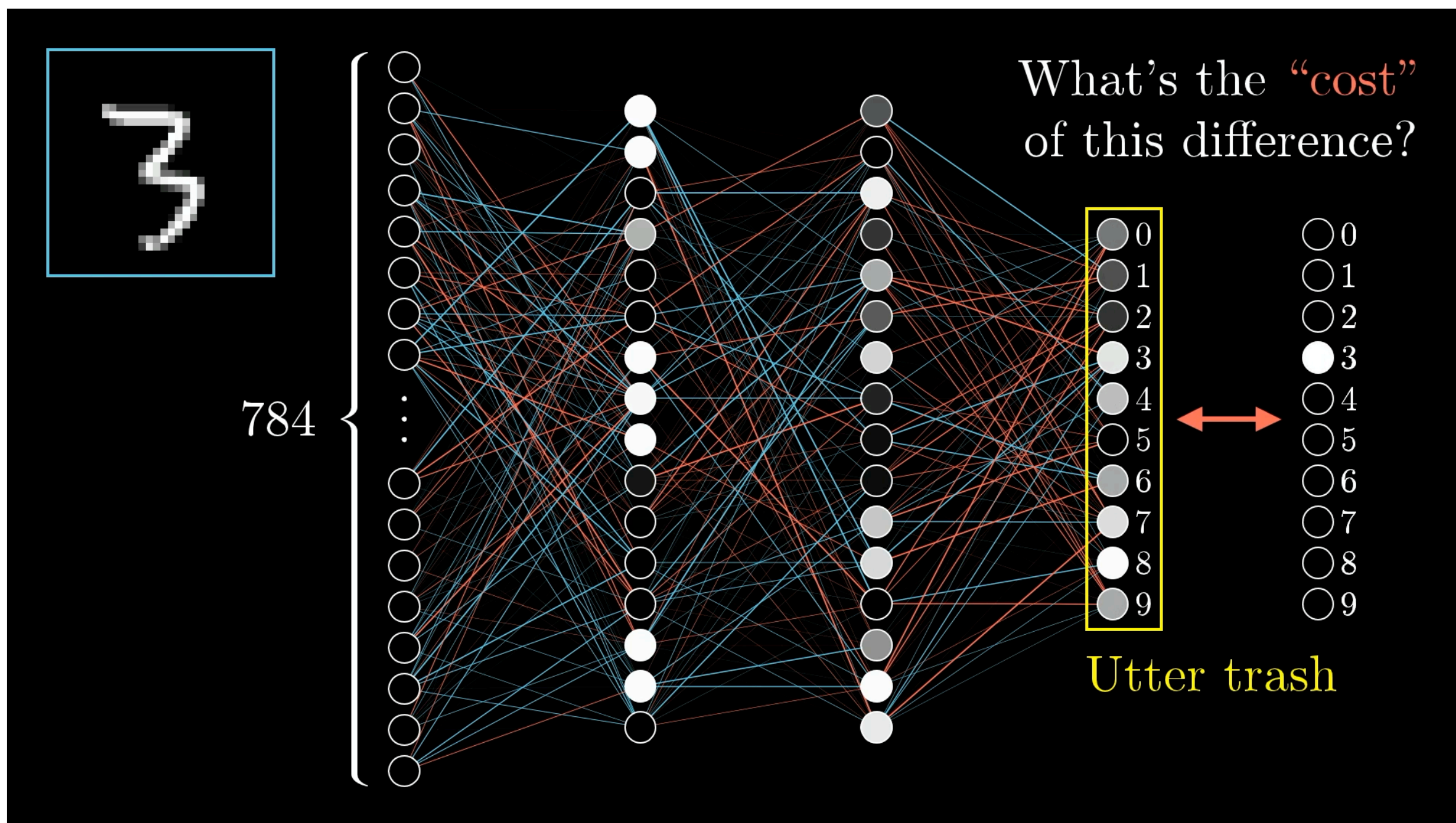


10 outputs

Training

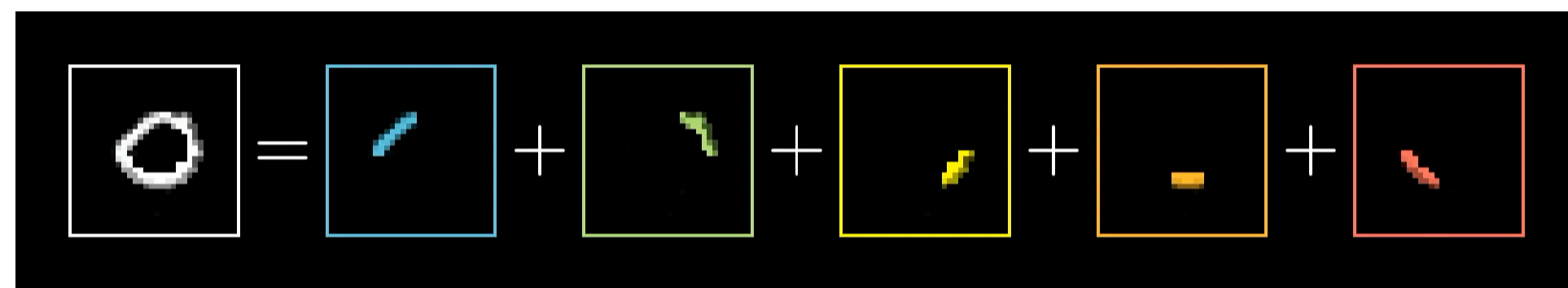
- Process of finding best w and b values referred to as “training the network”
- What do we mean by “best”?
 - For some w and b , each neuron in output layer contains value between (0, 1)
 - A probability measure denoting how confident the network is about the input image corresponding to that label
 - Can define loss function exactly as before (“least-squares”) as

$$L(w, b) = \sum_{i=0}^9 (\hat{y}_i - o_i(w, b))^2 \text{ and minimize this}$$



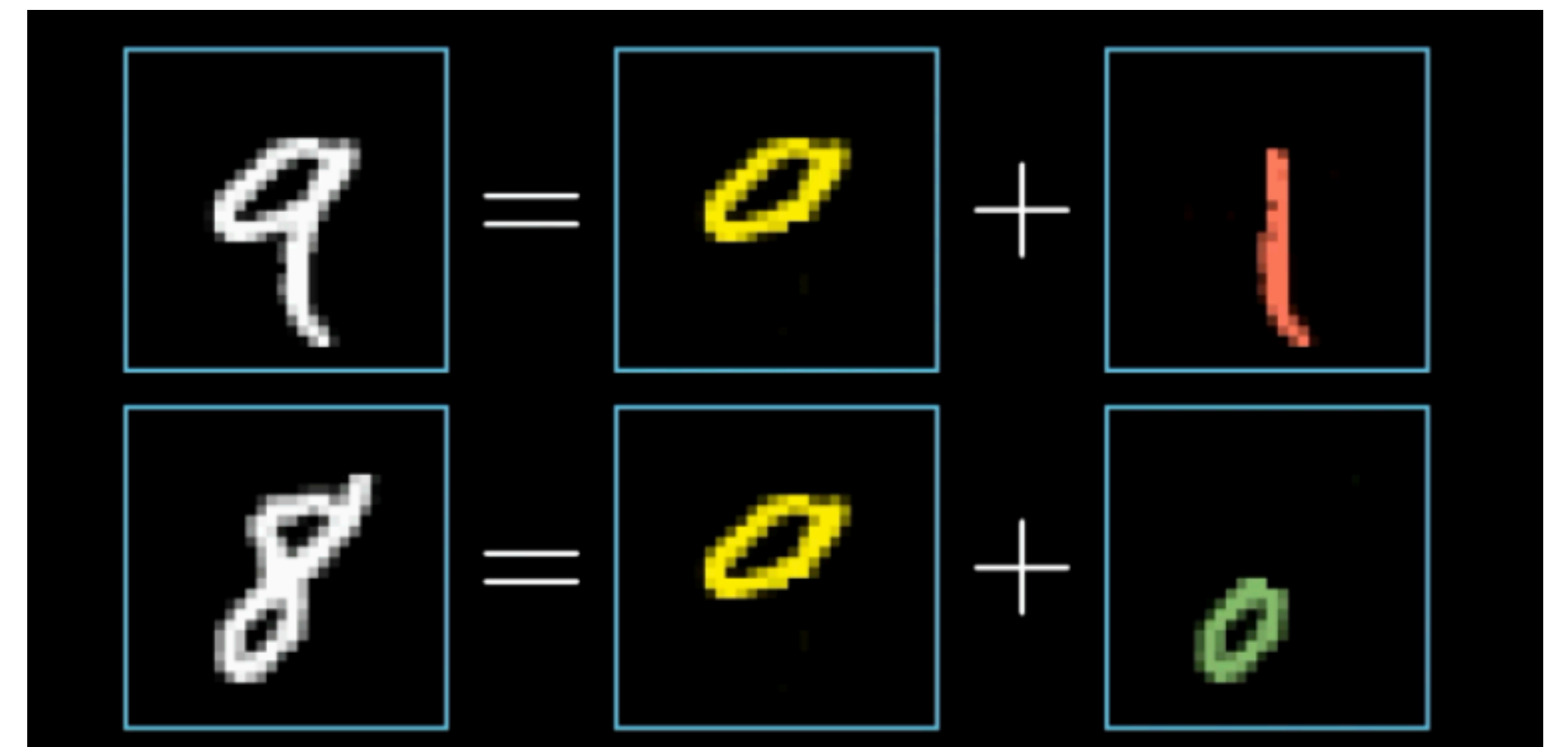
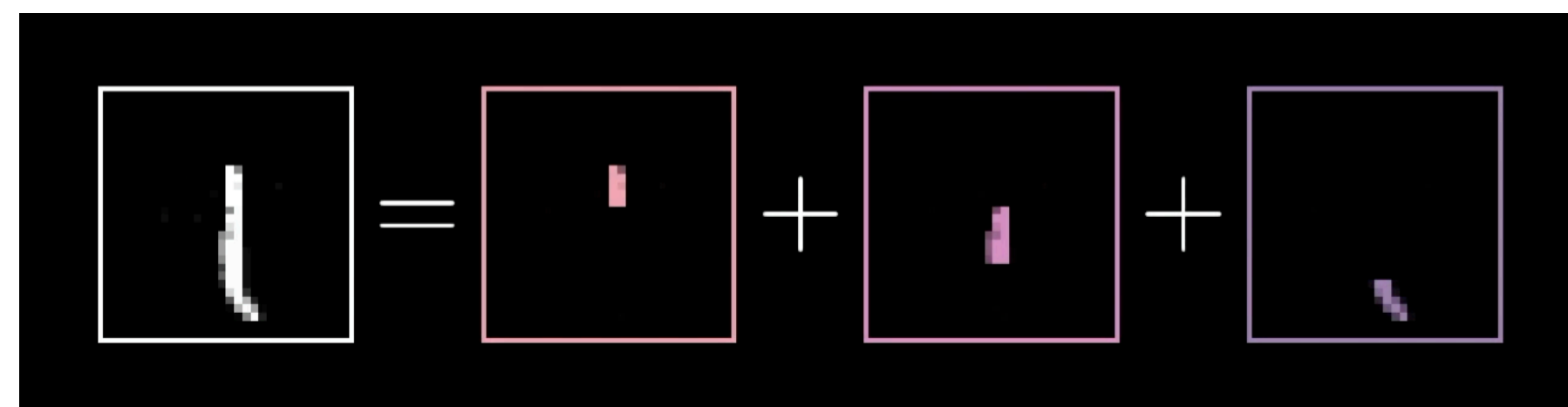
Why so complex?

- Our brains can easily recognize patterns/digits even if the images were a bit fuzzy
- But its a hard problem. We need ~13000 parameters to be able to describe arbitrary decision boundary shapes
- Could imagine the neural network decision flow as :
 - 2nd layer of neurons detect edges of image where pixels are bright
 - 3rd layer of neurons combines these edge pixels in various shapes, for eg loops or lines
 - Final layer might try to correlate number of loops or lines to the actual digits, for eg, 2 loops 0 lines => label “8” etc
- As we feed in more data during the training phase to optimize the loss function, network can get better and better at figuring out these patterns



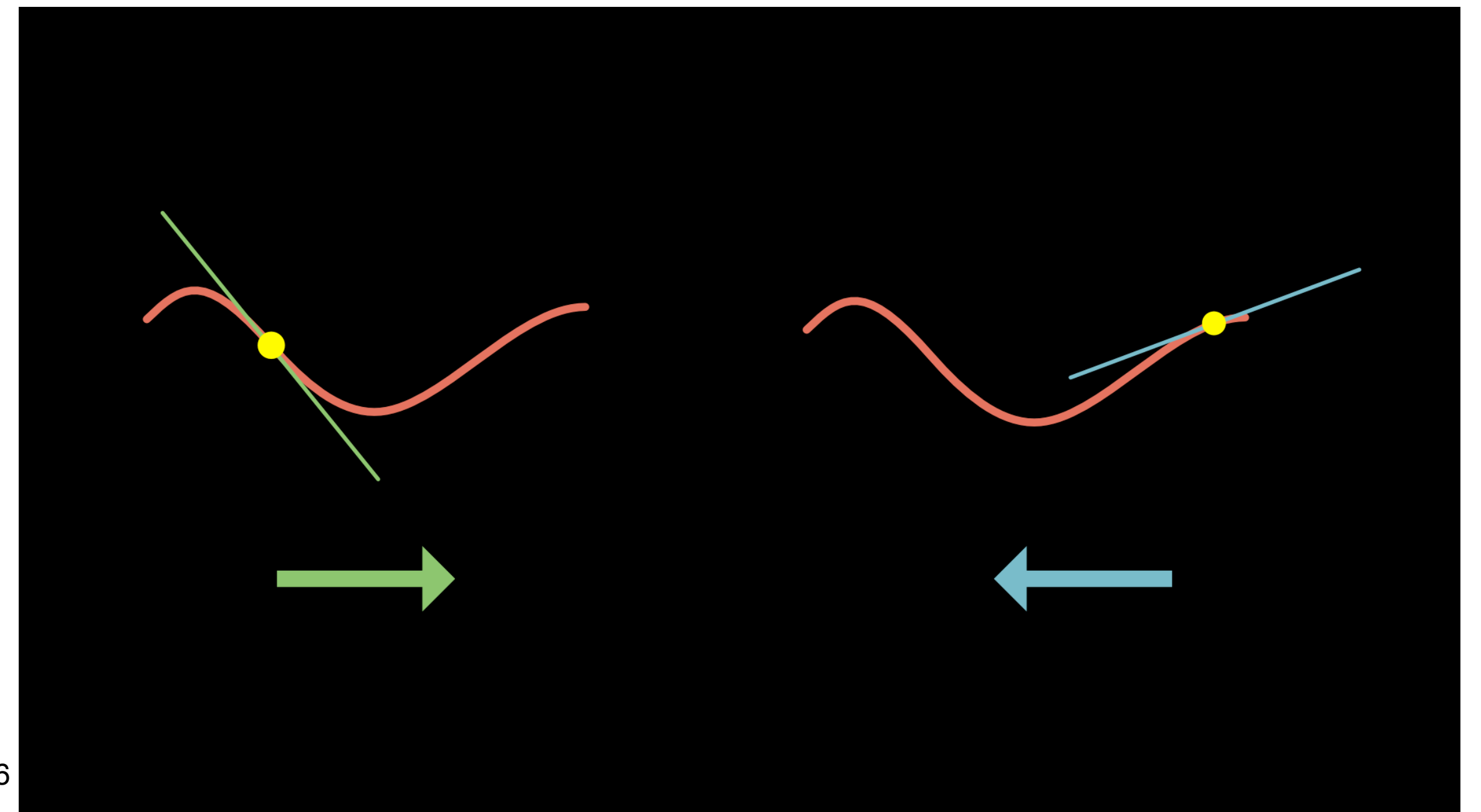
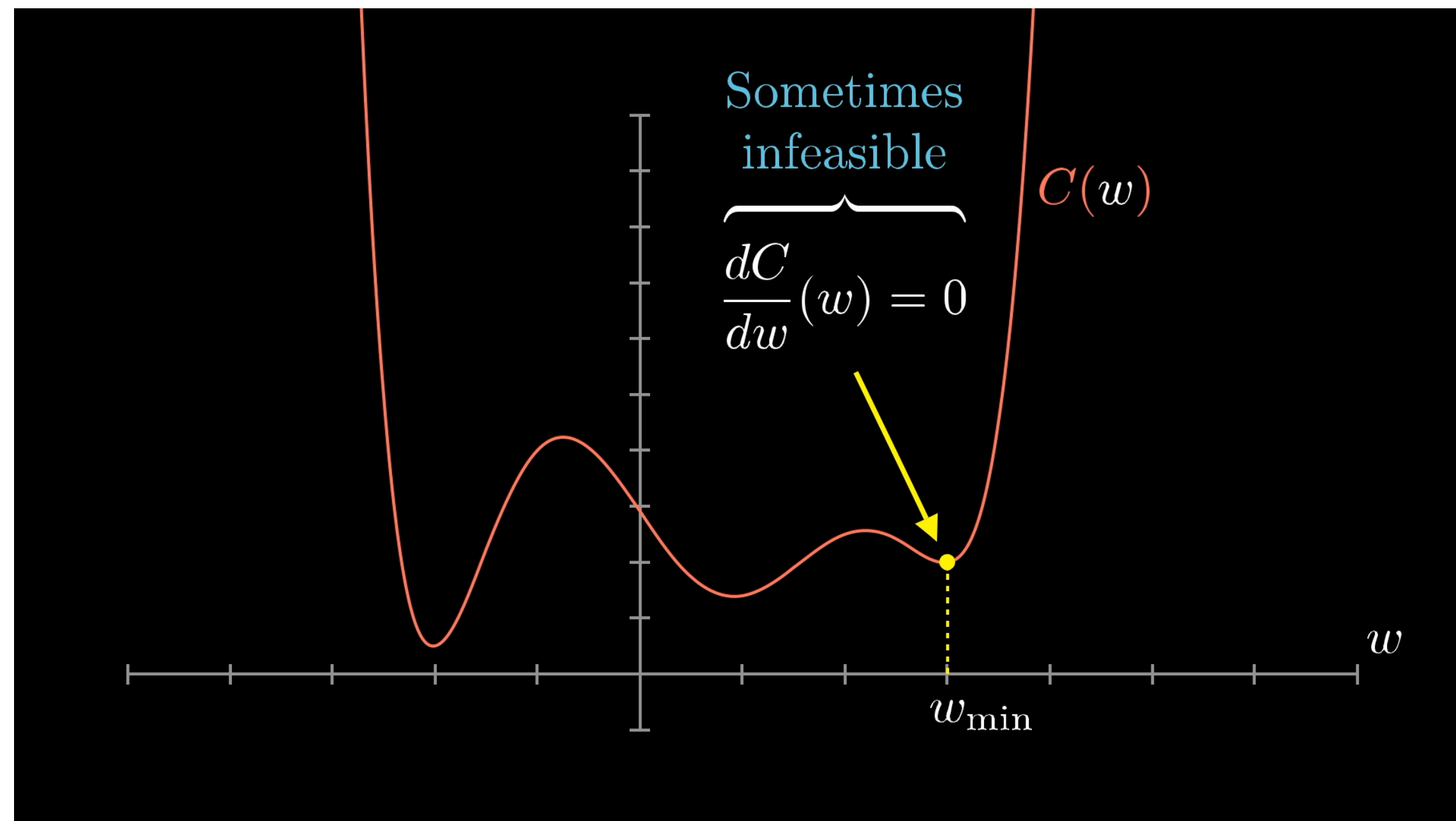
+

=



Gradient Descent

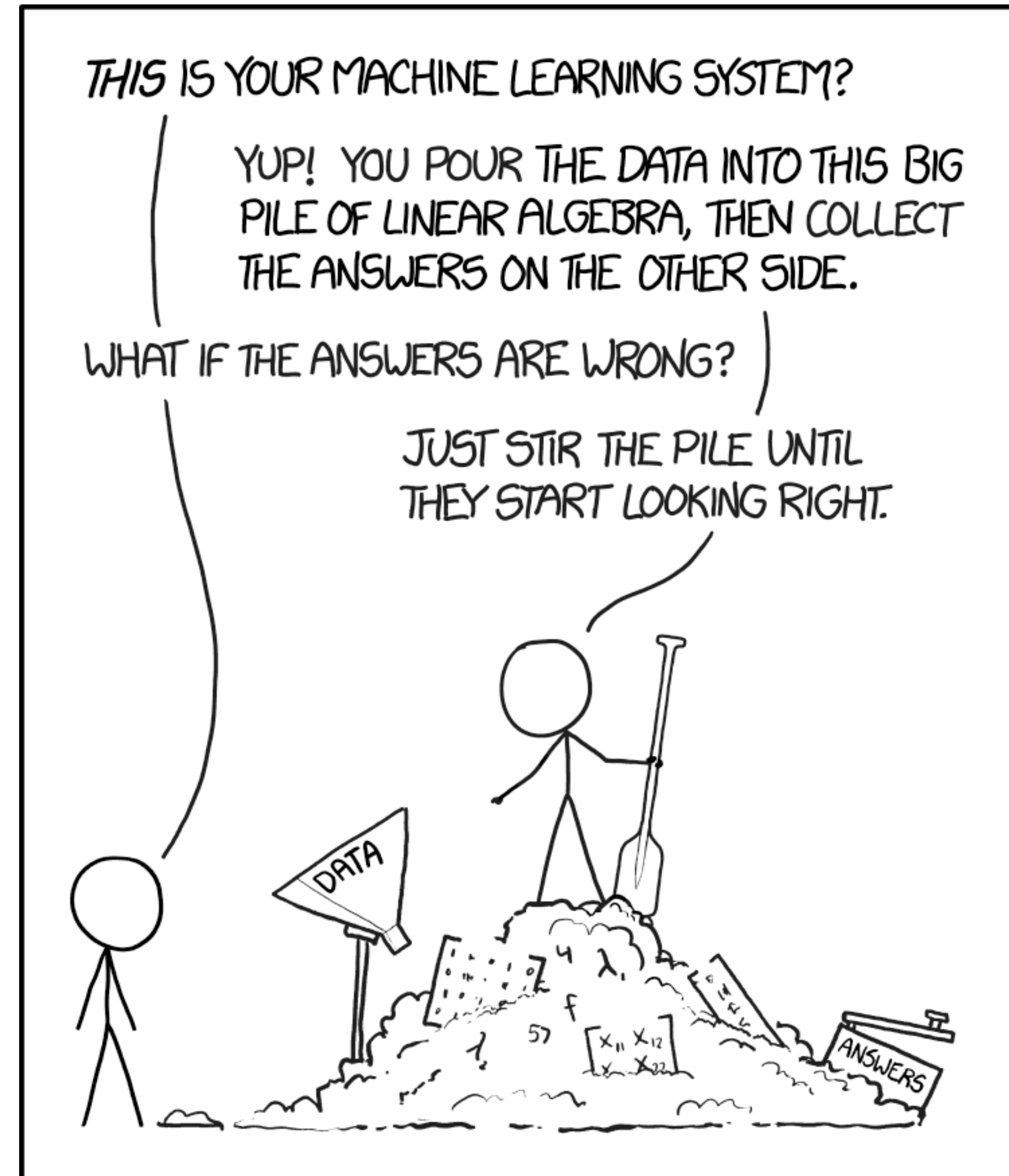
- Algorithm to minimize loss-function, $L(w, b)$ and find best w, b
- $L(w, b)$ has 13,002 input parameters but outputs a single number
- To find minimum, use Gradient descent — “ball rolling down a hill”
- Non-convex optimization : not guaranteed to find global minimum
 - Try starting the ball at different starting points. Also “stochastic GD” — try to make ball jump across valleys



What's the takeaway

<https://xkcd.com/1838/>

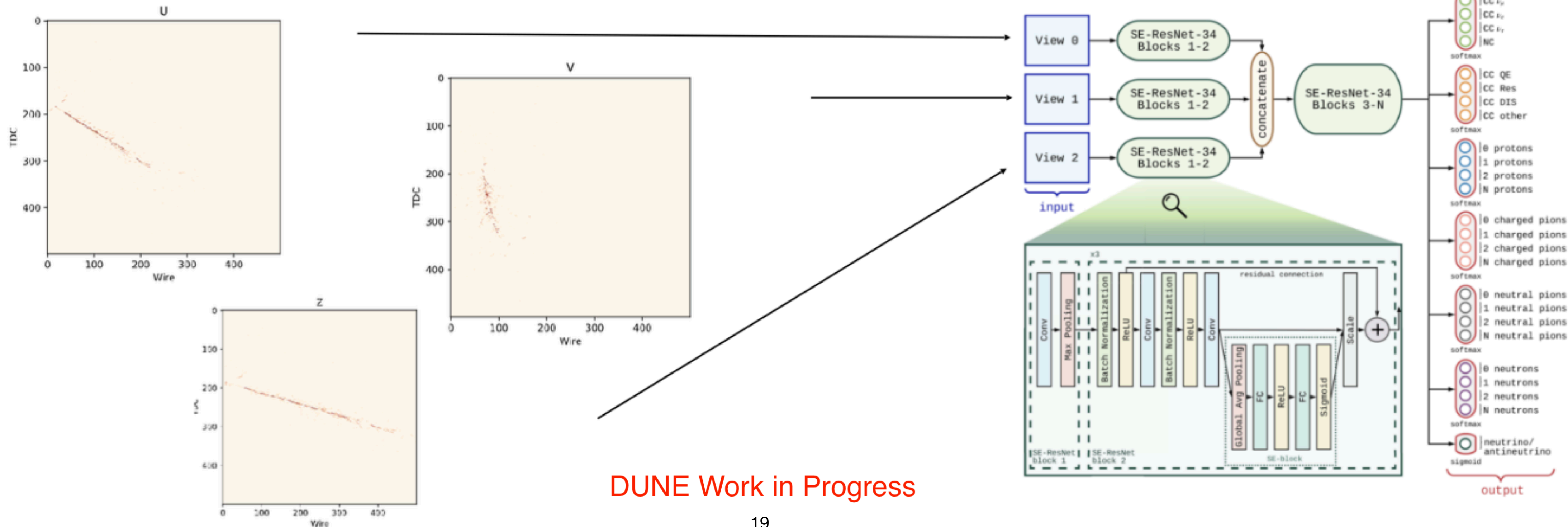
- Forced to deal with complexity
 - Large dimensionality, unknown parametric relationships
- Overfitting : network starts to predict based on spurious features it learns from training data
 - For eg, dependence on handwriting style — “bias”
 - But its hard to know what it learns ~Black box
- Don't assess network performance on training dataset
- Always keep a fraction of data separate and then see how trained network performs (“Test dataset”)
 - If training errors are very different from test errors, we may have issues
- Also good practices, shuffling dataset before training, cross-validation, i.e change up training and test data for different iterations
- Playing around with GD parameters etc



Neutrino Flavor Tagging

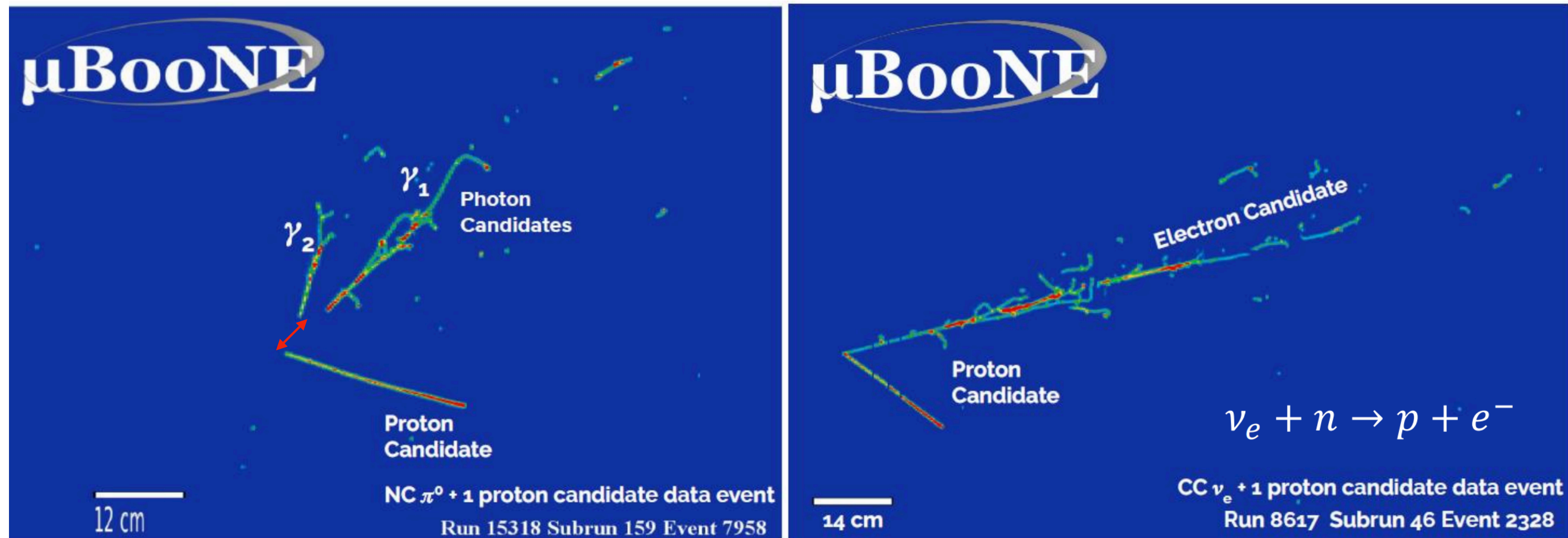
- Classification problem
- We use “deep neural networks”, particularly a brand called convolutional neural networks
 - Trained on ~6 million total events across ν_e CC, ν_μ CC, ν_τ CC, ν_x NC
 - ~22 million parameters ($\gg 13000!$), “softmax” activation function to squish neurons to b/w (0, 1)
 - Trained for a week using Nvidia GPU clusters. Actually, most modern GPUs can handle these kinds of payloads but sometimes need more than 1
 - Three input images, each $500 \times 500 = 250,000$ pixels
- Output is a score b/w (0, 1) for each of 4 flavor labels : ν_e CC, ν_μ CC, ν_τ CC, ν_x NC. Also has scores for other features of interaction

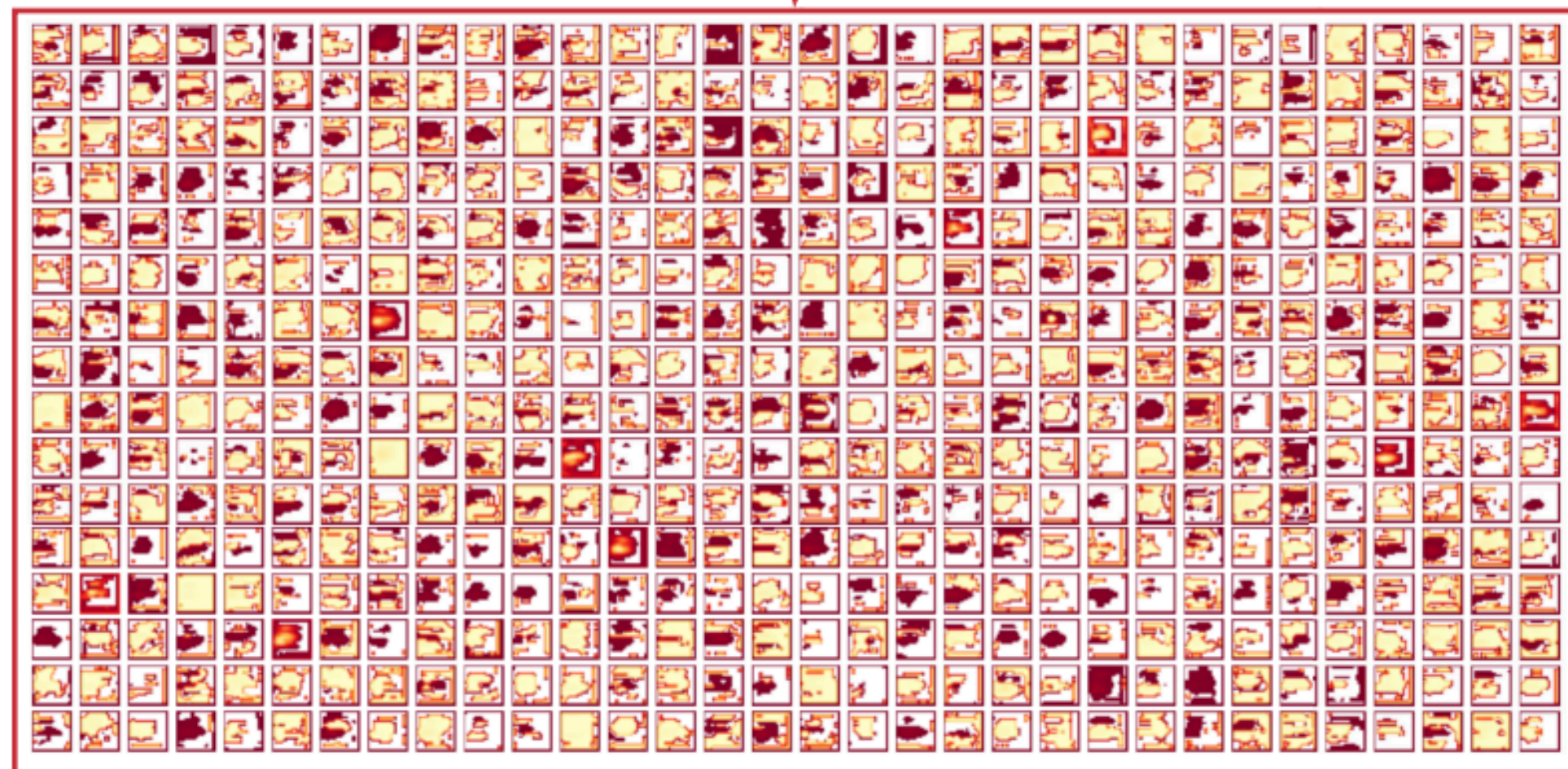
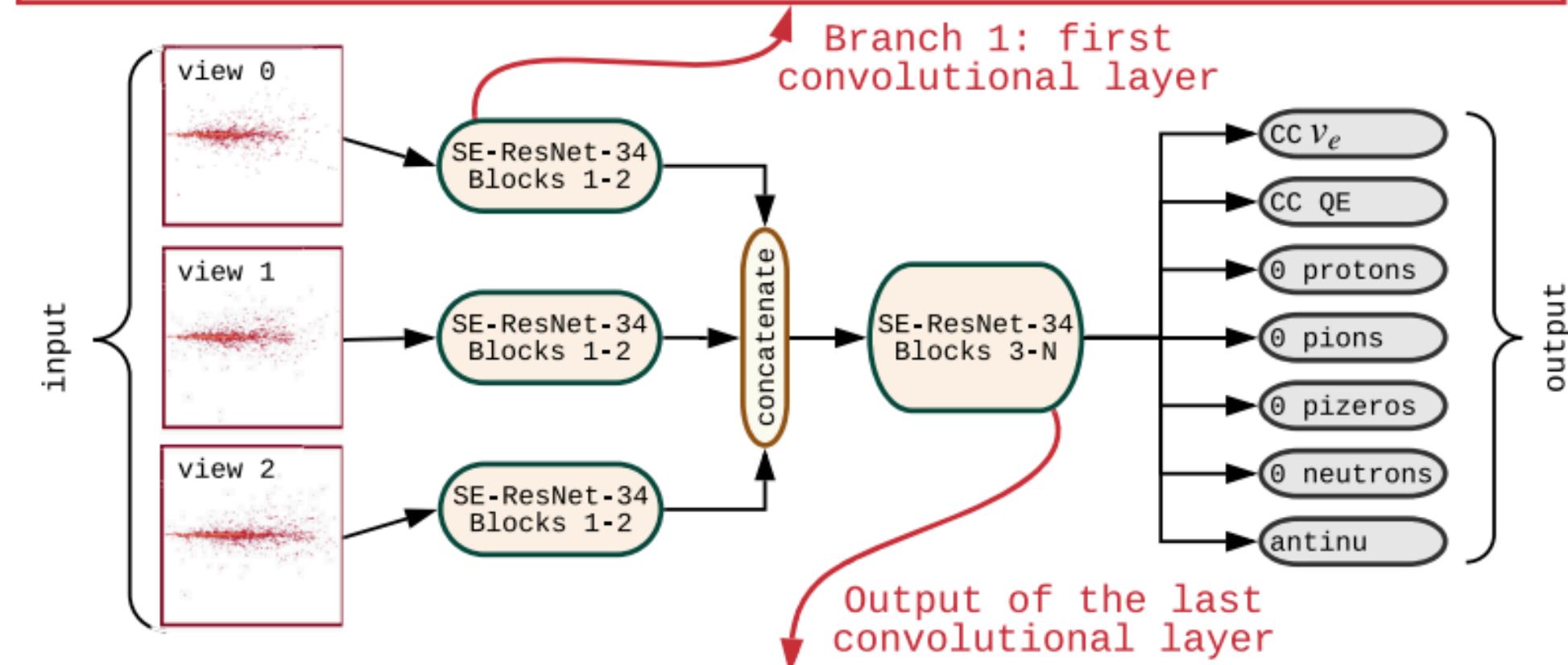
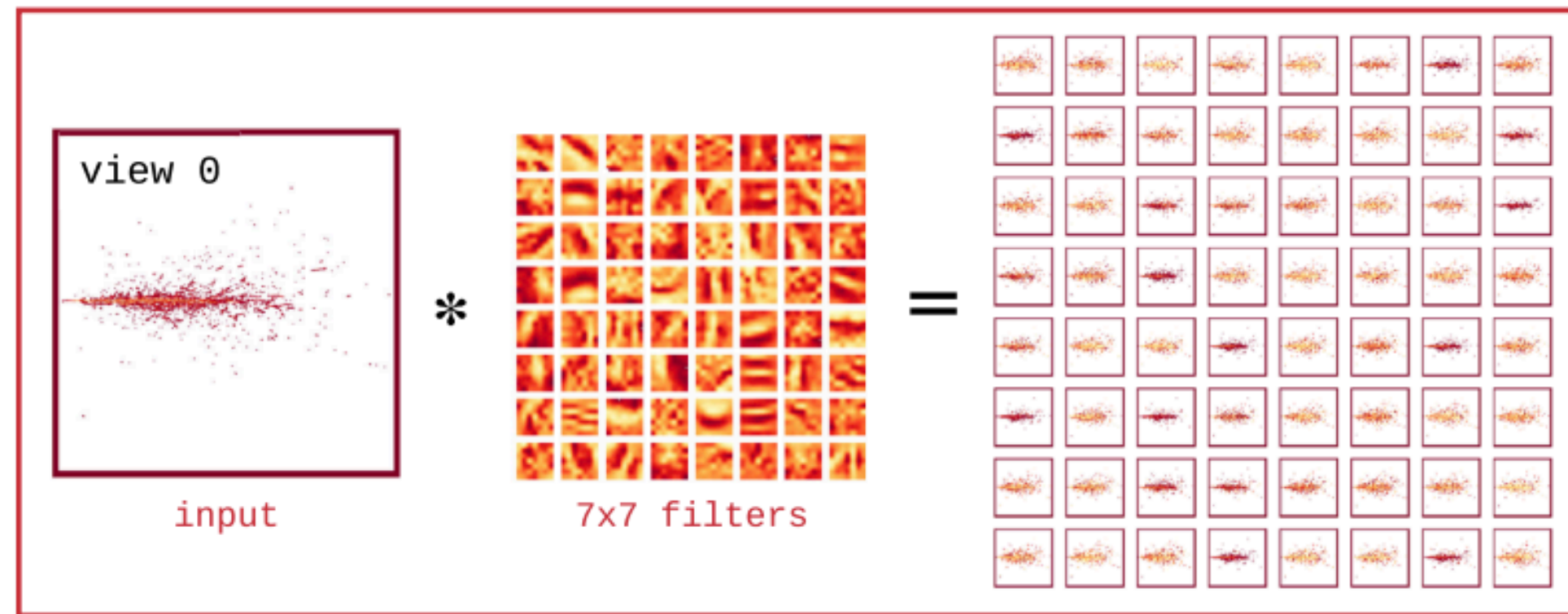
$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^K e^{z_j}}$$



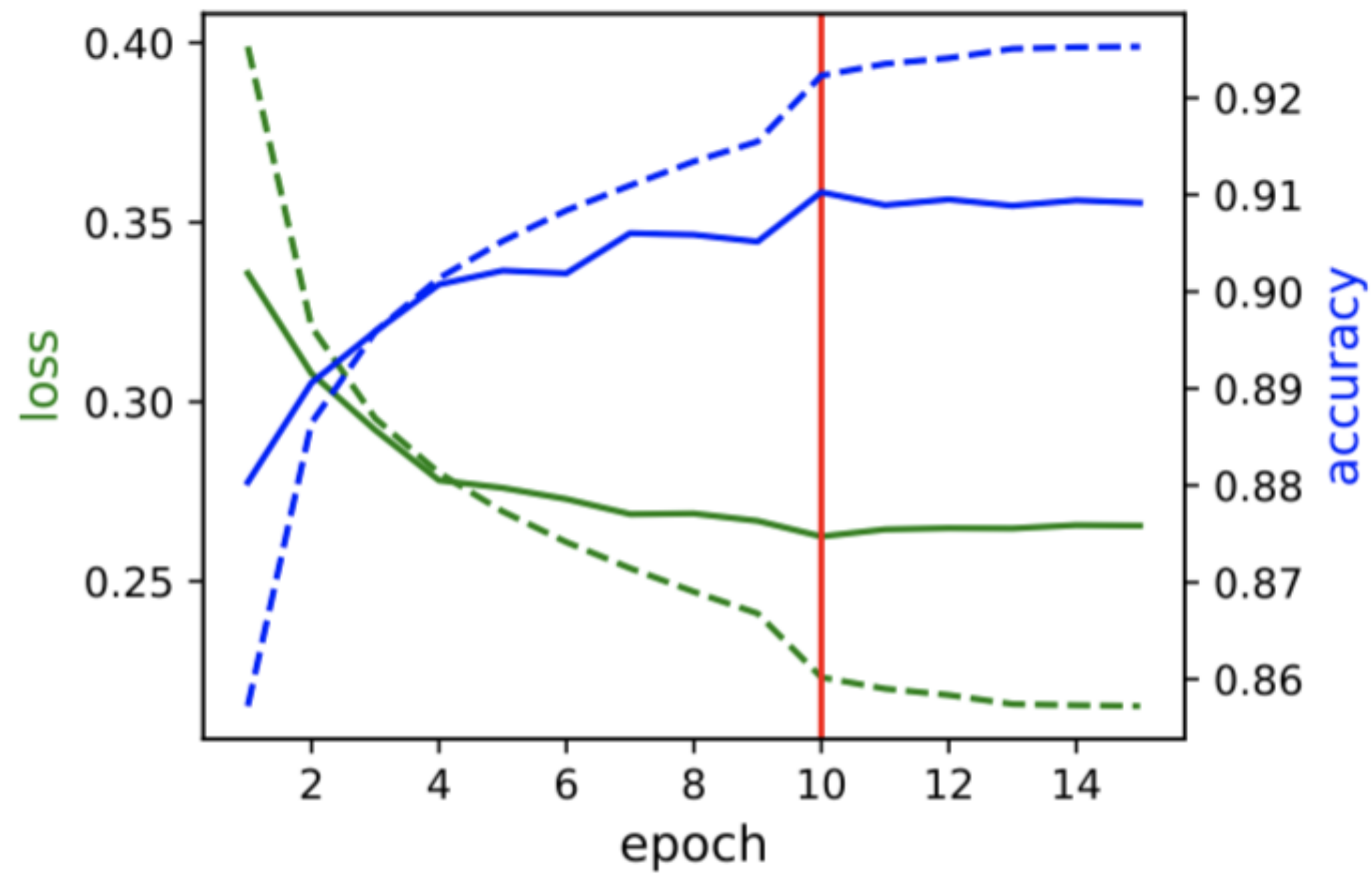
Why Deep?

- Previously, traditional approaches involve using set of human-engineered features as input, even to a shallower neural network
 - Examples of useful features for our problem : number of showers, gap from vertex, number of tracks
- Deep neural networks are able to figure these out themselves and also able to catch features we may have missed
 - We get more accurate networks, but possibly at the cost of not knowing as much about what its doing.
 - Validation is key to building trust!

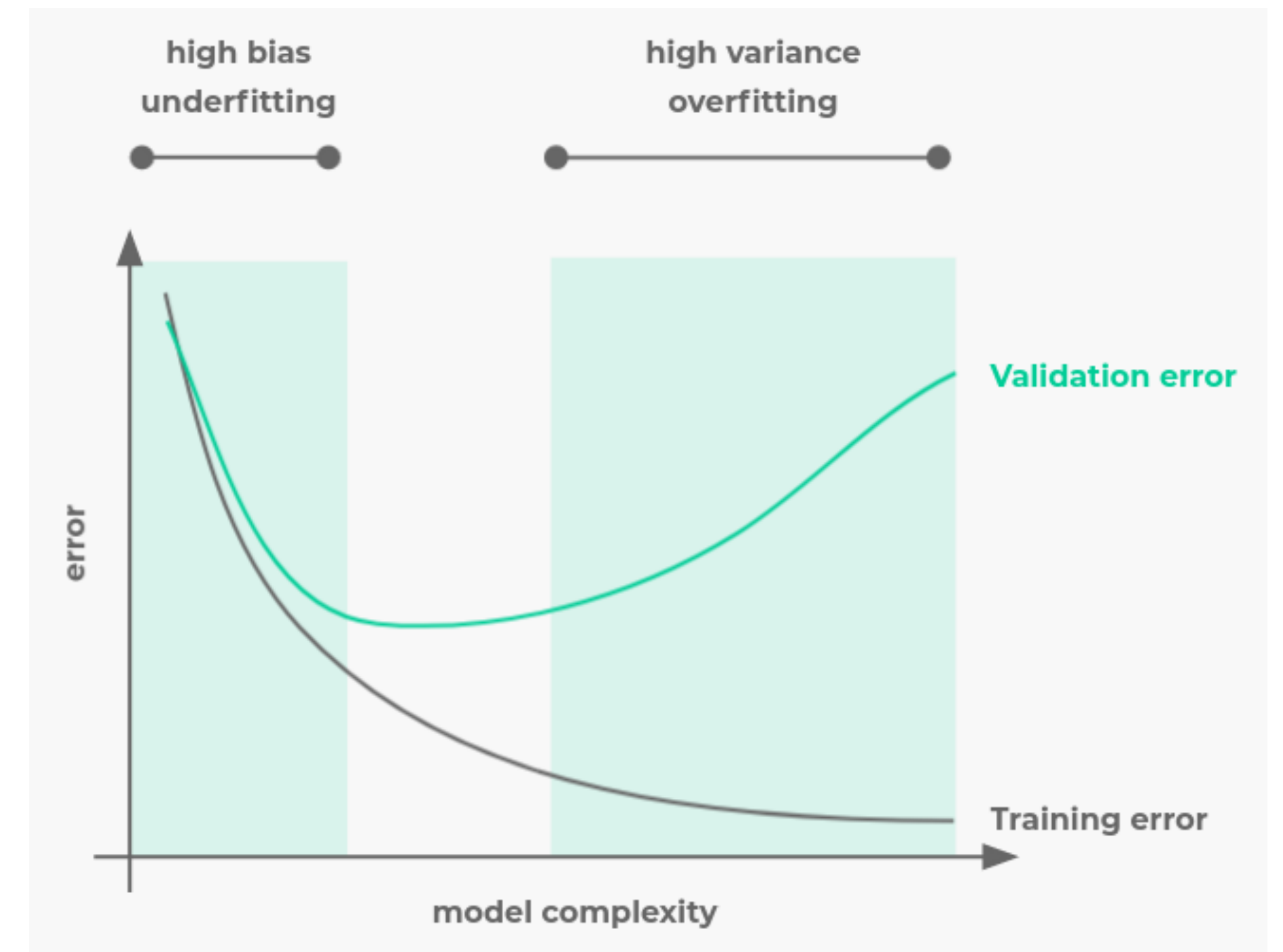




- Extracted features end up looking a bit like this
 - Pretty abstract, hard to interpret
- Training process is actually done over multiple iterations
- Works something like this :
 - Split dataset into 90% “training”, 10% “test”. Don’t touch test dataset until after all the training is done
 - For each iteration (“epoch”), further split “training” into 80% actual training, 10% validation dataset after shuffling randomly
 - Evaluate loss function in each iteration for different images
 - Tune w , b using Gradient Descent
 - Once all the 80% dataset is exhausted, evaluate result on validation dataset (“accuracy”, “loss”)
 - Now re-shuffle the 90% again into a different training and validation set and start all over again



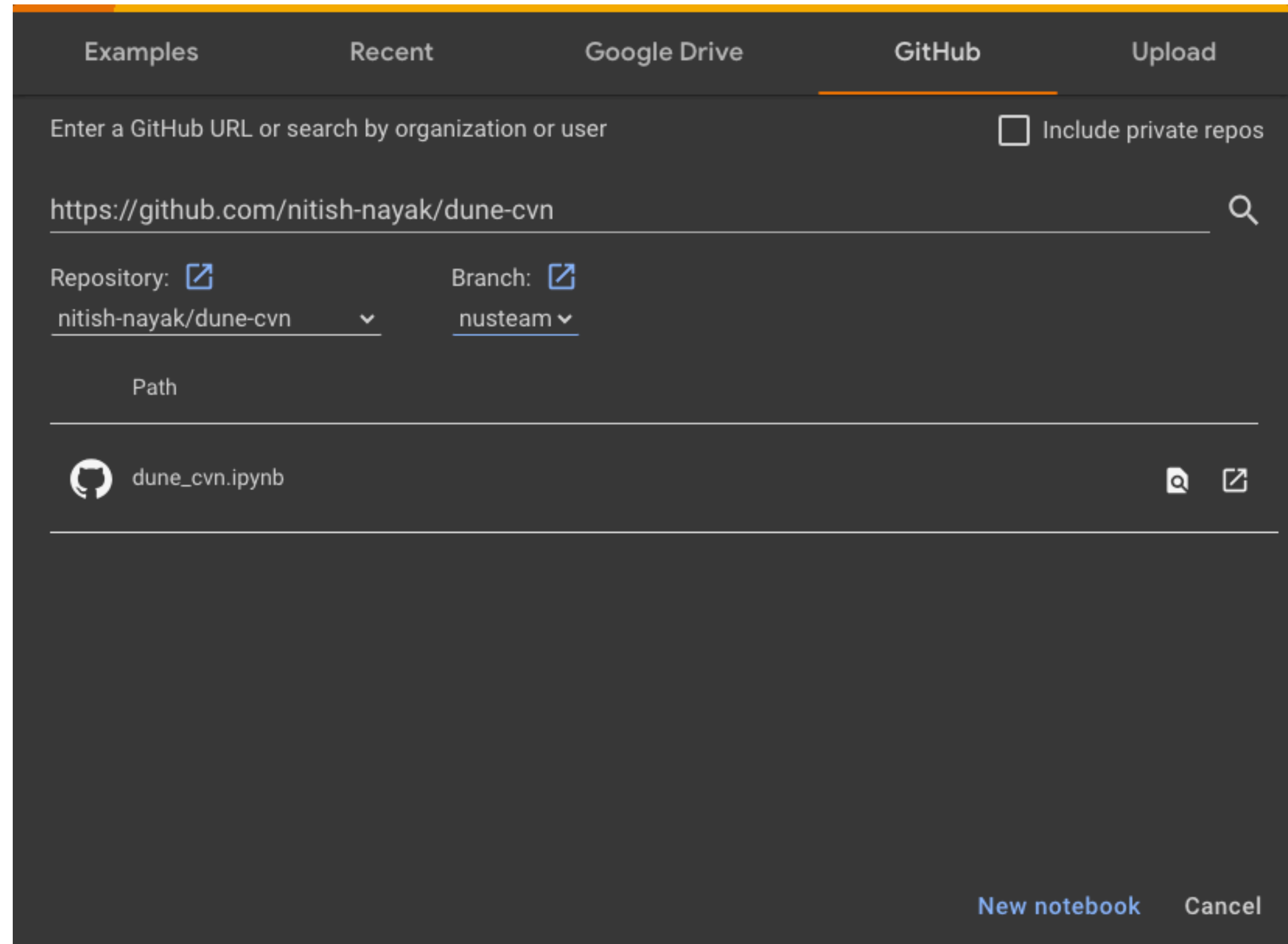
Doesn't look like this!



- Dashed is on training dataset, solid is on validation dataset
- For now, we stop training when things stop improving after a few epochs/iterations
- Notice also that it flatlines — this is a good sign that there's no overfitting!
- If we're confident the training went well, we can then look at the test dataset

Lets test things out!

- As a demo, we will test out the CNN network used in [<https://journals.aps.org/prd/abstract/10.1103/PhysRevD.102.092003>]
- We can do all this on the browser itself, hopefully without having to install anything
 - Go to <https://colab.research.google.com/>
 - In the pop-up, go to the GitHub tab and type in url : <https://github.com/nitish-nayak/dune-cvn> w/ the “nusteam” branch
 - You should see two .ipynb files (Python notebooks)
 - Select “dune_cv.n.ipynb”



Further Reading/Homework

- Borrowed heavily from 3Blue1Brown's excellent neural network explainer :
 - <https://www.3blue1brown.com/topics/neural-networks>
 - Youtube playlist : https://www.youtube.com/watch?v=aircAruvnKk&list=PLZHQObOWTQDNU6R1_67000Dx_ZCJB-3pi
- Other references :
 - <https://towardsdatascience.com/a-visual-introduction-to-neural-networks-68586b0b733b>
 - <https://towardsdatascience.com/artificial-neural-networks-for-total-beginners-d8cd07abaae4>
 - <https://towardsdatascience.com/conv-nets-for-dummies-a-bottom-up-approach-c1b754fb14d6>
 - https://www.youtube.com/watch?v=YRhxdVk_sls
- Build and train your own CNN :
 - [https://github.com/josephlee94/intuitive-deep-learning/blob/master/Part%202:%20Image%20Recognition%20CIFAR-10/Coding%20Companion%20to%20Intuitive%20Deep%20Learning%20Part%202%20\(Annotated\).ipynb](https://github.com/josephlee94/intuitive-deep-learning/blob/master/Part%202:%20Image%20Recognition%20CIFAR-10/Coding%20Companion%20to%20Intuitive%20Deep%20Learning%20Part%202%20(Annotated).ipynb)
 - <https://medium.com/intuitive-deep-learning/build-your-first-convolutional-neural-network-to-recognize-images-84b9c78fe0ce>