Machine Learning in IceCube

Josh Peterson
2024 IceCube Summer School

Outline

- **Machine learning introduction and general methods**
- **Decision Trees**
- **Neural Networks**

Outline

- **Machine learning introduction and general methods**
- Decision Trees
- Neural Networks

Machine Learning (ML)

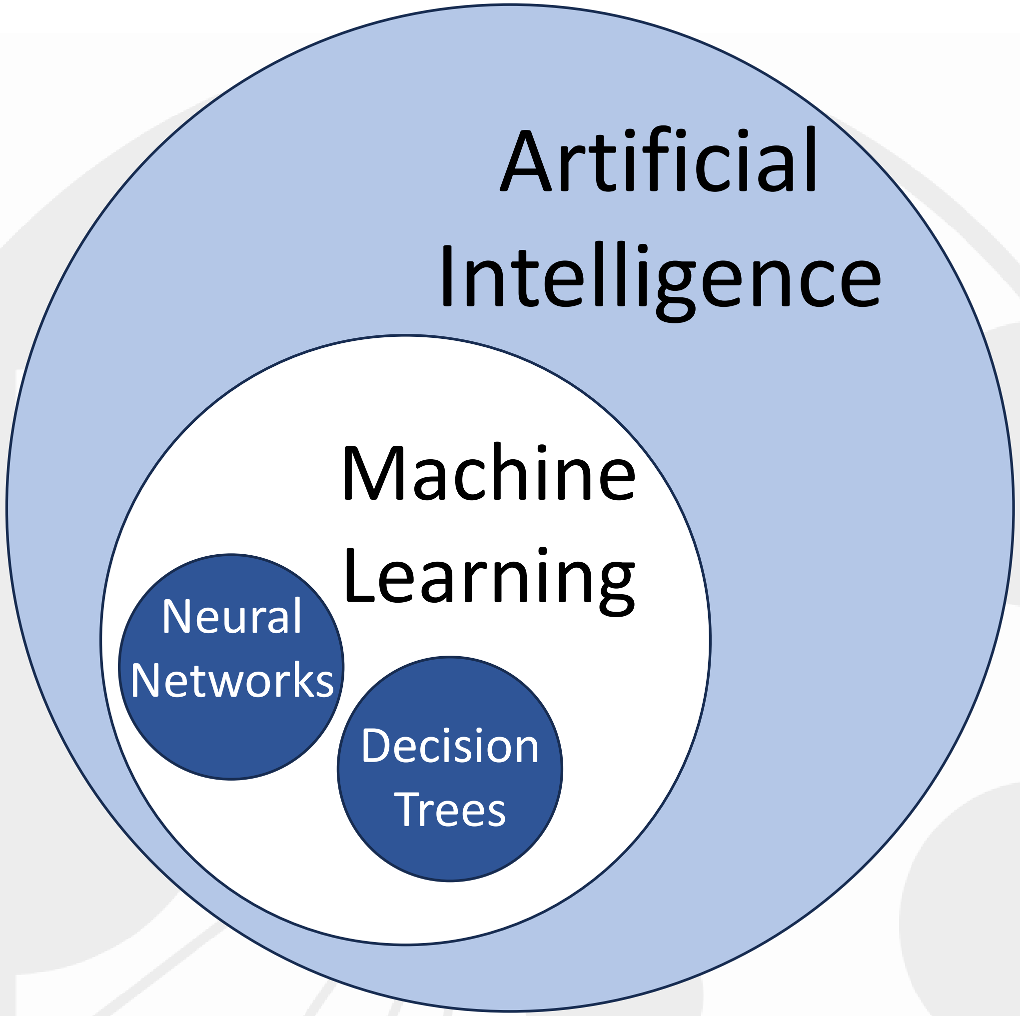
- **A field of artificial intelligence in which you "teach" a computer to perform a task without having to explicitly program how to do the task**

- **A lot of the time, this effectively means optimizing a function on a set of data**
  - **A lot of linear algebra, calculus, and probability**

- **There are many kinds of machine learning algorithms, but today we will focus on two that show up very often in IceCube work:**

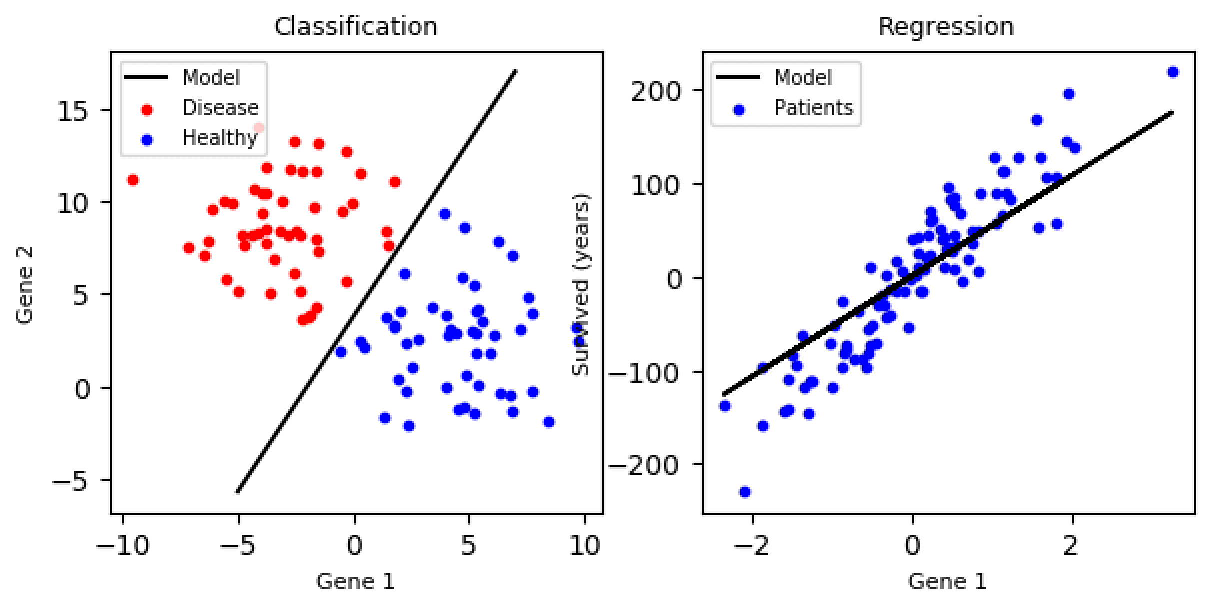
  - **Decision Trees**

  - **Neural Networks**



Tasks

- **A machine learning algorithm learns how to perform a specific task. Here are a couple examples that are common in IceCube:**
  - **Regression:** Find a function such that $f(x_i) = y_i$ for all x_i in the data
    - Energy reconstruction, direction reconstruction
  - **Classification:** Organize x_i into n classes for all x_i in the data
    - Neutrino flavor identification, separating atmospheric muons from muon neutrino events



https://dev.to/petercour/machine-learning-classification-vs-regression-1gn

Types of Learning

- **There are many ways to optimize. Here are a couple common examples:**
  - **Supervised learning:** Provide the ML algorithm the data x_i and the truth y_i
  - **Unsupervised learning:** Provide the ML algorithm with the data x_i, have it learn qualities of the data
- **Supervised learning is the most common method used in IceCube, due to our huge amount of simulation**

Loss Functions

- **A loss function is the metric you use to quantify how well your machine learning algorithm is performing a task**
- **When learning, we minimize the loss**
- **Different loss function are good for different tasks**

Regression Loss Functions

Loss	Equation	Property
L2	$\sum_i (f(x_i) - y_i)^2$	**Pretty standard**
L1	$\sum_i \lvert f(x_i) - y_i \rvert$	**Better at characterizing outliers than L2**
Huber	$\begin{cases} 0.5(f(x_i) - y_i)^2 \ for \ \lvert f(x_i) - y_i \rvert < \delta \\ \delta(\lvert f(x_i) - y_i \rvert - 0.5\delta) \ otherwise \end{cases}$	**Combo of L1 and L2**
Beta	$-\sum_i \log(p_{beta}(f_\alpha(x_i), f_\beta(x_i)))$	**Neural network produces a PDF**

Classification Loss Functions

Loss	Equation	Property
Binary Cross Entropy Loss	$N^{-1} \sum_i w_i [y_i \log(f(x_i)) + (1 - y_i) \log(1 - f(x_i))]$	**For 2 classes**
Cross Entropy Loss	$N^{-1} \sum_i -w_{y_i} \log\left(\frac{\exp\left(f\left(x_{i,y_i}\right)\right)}{\sum_c \exp\left(f\left(x_{i,c}\right)\right)}\right)$	**For >2 classes**

Training

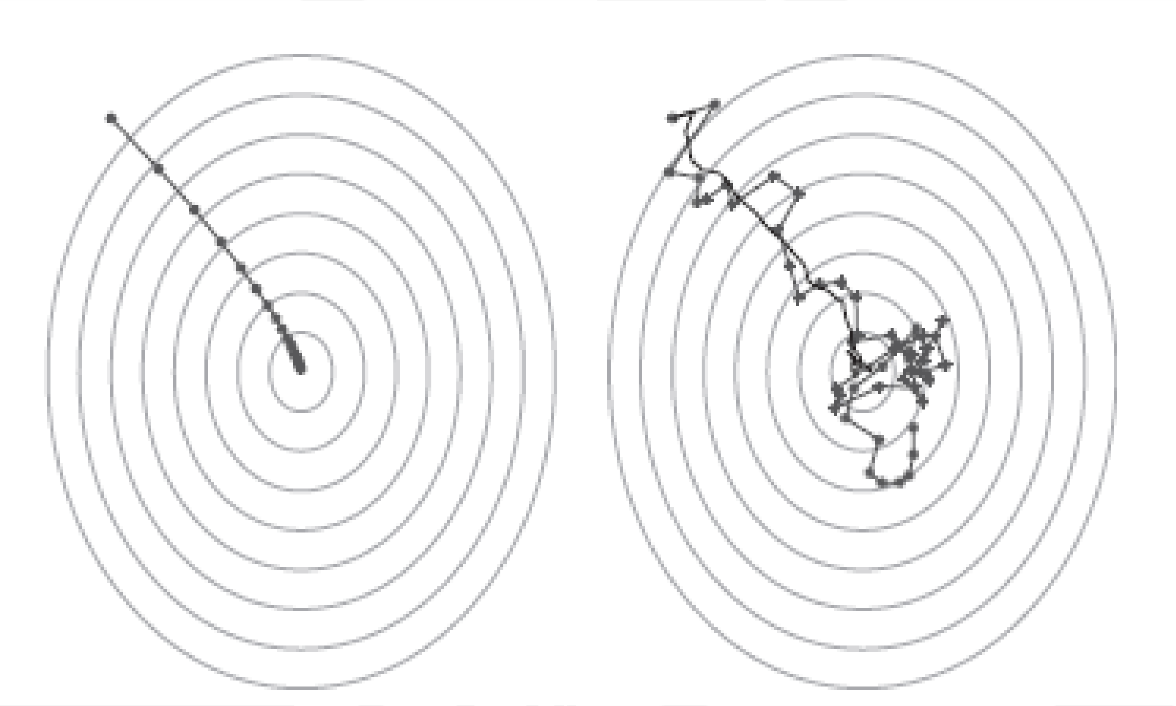
- **“Training” refers to optimizing the algorithm to minimize the loss**
  - Optimization method depends on the ML algorithm being trained

- **For huge amounts of data, stochastic gradient descent is a very common optimization algorithm**

  - General Idea: you can roughly find the correct direction to descend if you use a small batch of your data to compute the gradient

- **Batch: A portion of data to use to calculate the gradient for every step**

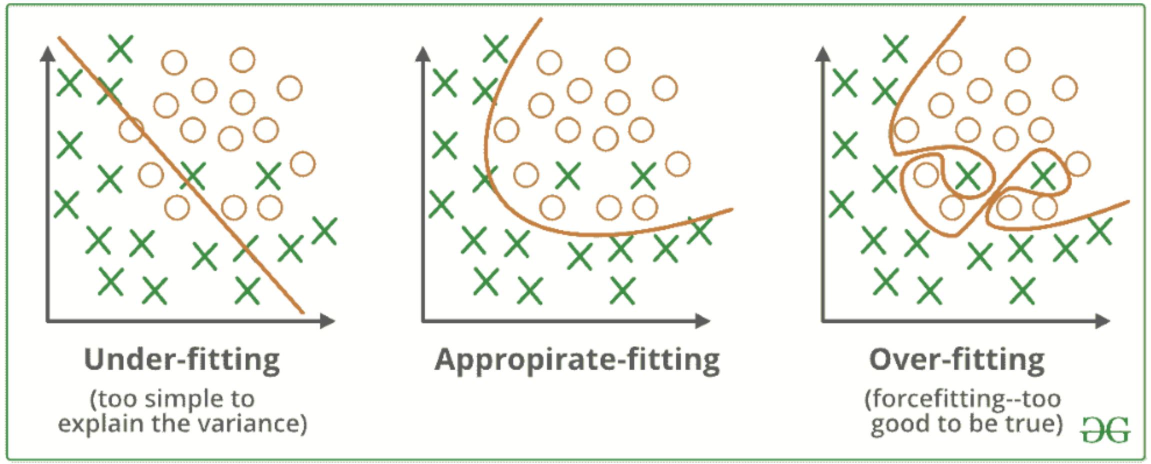
- **Epoch: One use of all data you are using to train**



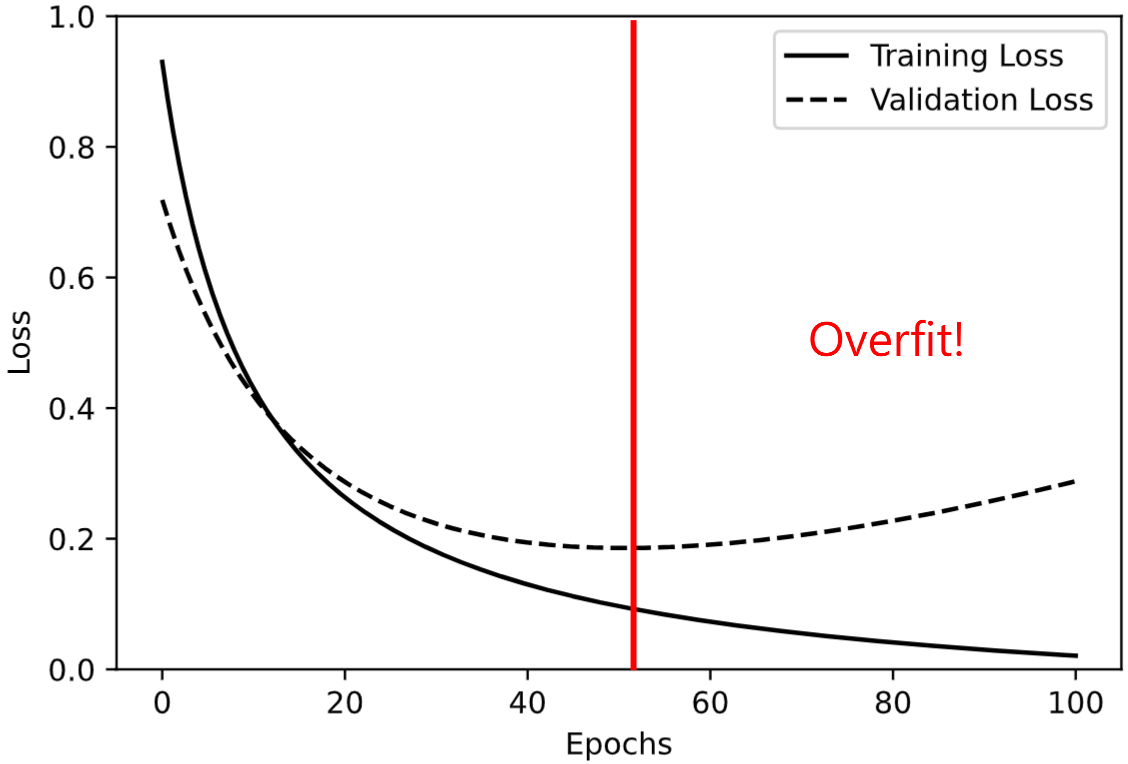
https://www.researchgate.net/figure/A-plot-of-the-gradient-descent-algorithm-left-and-the-stochastic-gradient-descent_fig1_303257470

Overfitting

- **Generally, we want our ML algorithm to learn things about the population from which our data sample comes from**
- **Overfitting occurs when the ML algorithm learns features that are specific to the sample of data it was trained on**
- **To prevent this, you could separate your data into the following sets:**
    - **Training set:** The data that you optimize the ML algorithm with
    - **Validation set:** Used at the end of each epoch to evaluate performance
    - **Test set:** Used to evaluate performance of the ML algorithm once training is completed
- **You generally should stop training when the validation loss achieves a minimum**

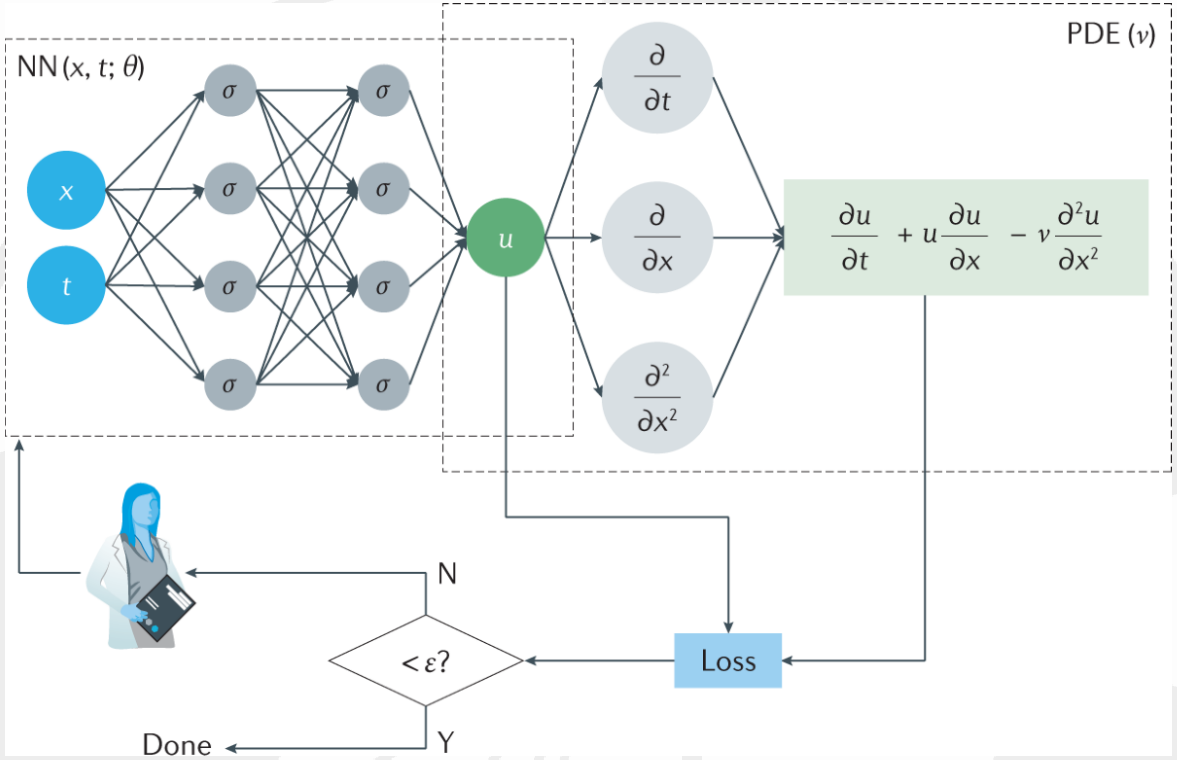


https://www.geeksforgeeks.org/underfitting-and-overfitting-in-machine-learning/



Regularization

- **If the ML algorithm is learning in an undesirable way, you can use regularization to alter the optimization**
- **Total Loss = Standard Loss + Regularizing Term**
- **Some examples**:
  - **L1/L2 regularization:** Apply the L1/L2 loss to the weights of the model. Good for overdetermined systems to prevent overfitting
  - **Early stopping:** We've seen this before!
  - **Physics informed neural networks:** Add loss terms that enforce specific physics



https://www.nature.com/articles/s42254-021-00314-5

Hyperparameters

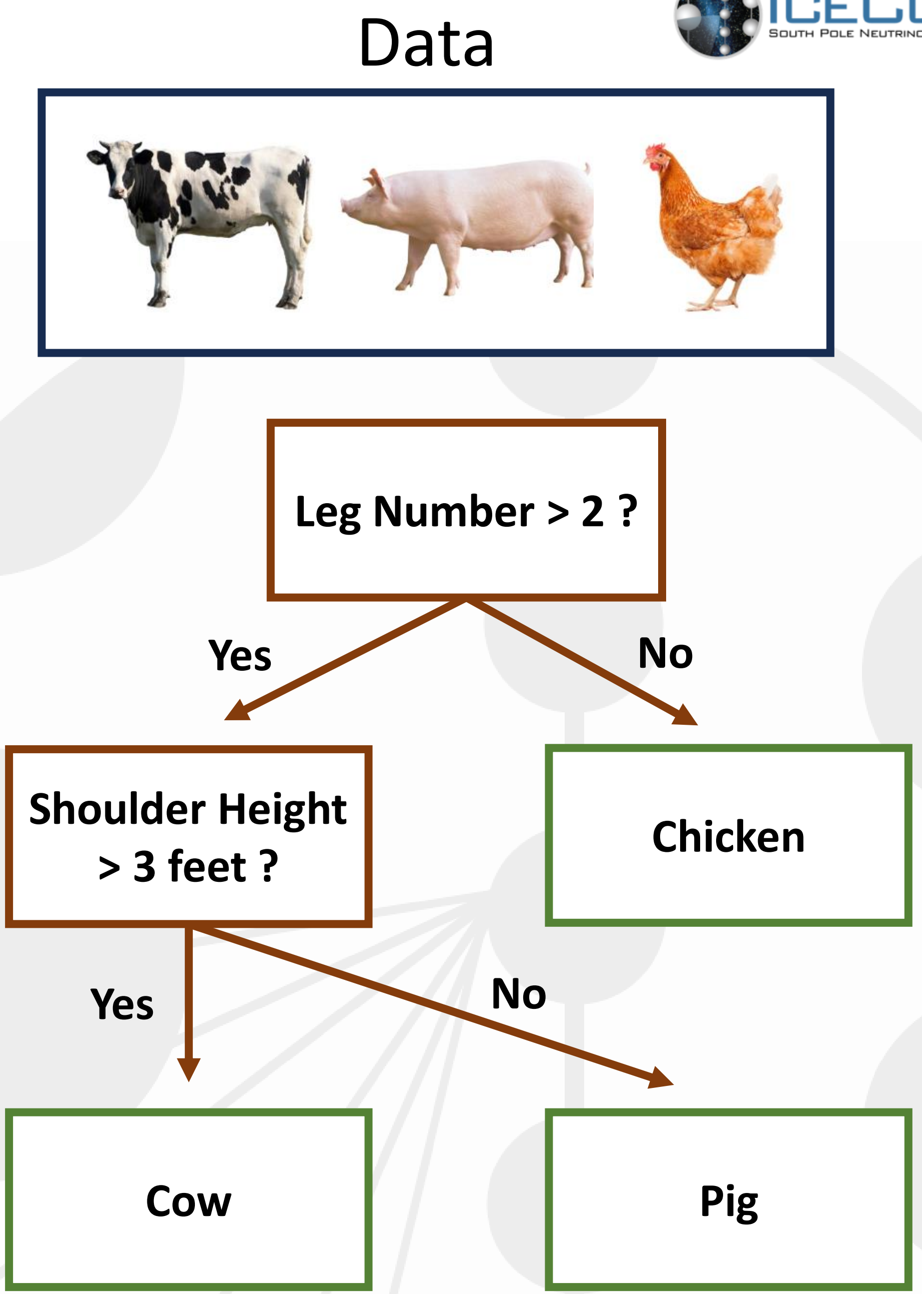
- **These are parameters that affect the training and performance of the ML algorithm that are not optimized (learning rate, hidden layer size, tree depth, regularization constant, etc.)**
- **There is no go-to method for determining hyperparameters**
    - Could use a genetic algorithm or cross validation for hyperparameter selection
    - If the training is fast enough / the number of hyperparameters is small enough one can do a grid scan
    - Can hand-tune hyperparameters until the desired performance is achieved

Outline

- Machine learning introduction and general methods
- **Decision Trees**
- Neural Networks

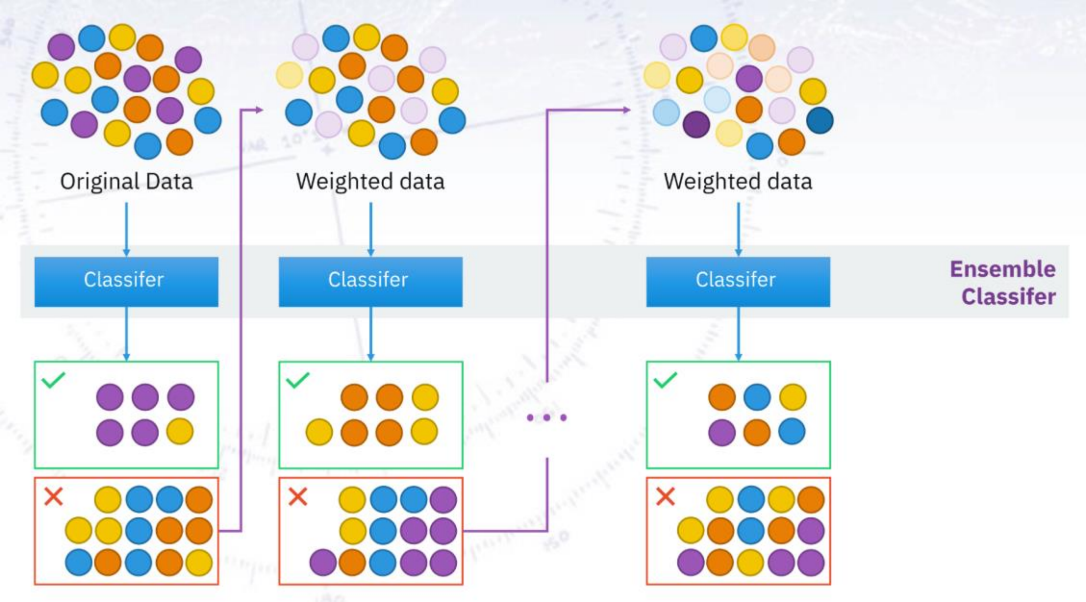
Decision Trees

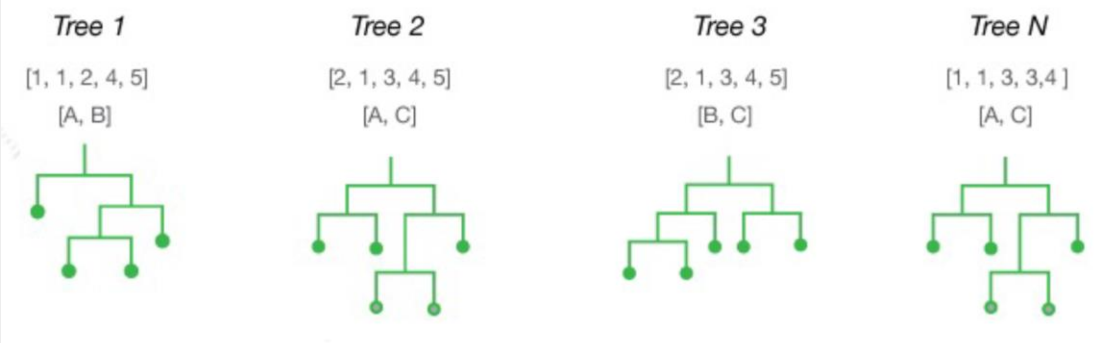
- **A series of binary decisions to give something a label**
- **Nodes: Data is sorted based on a binary criteria**
  - The variables used in a node and the cuts are what is learned
  - Chosen via information gain
- **Leaves: A terminal node representing the class, probability, or value assigned to the input**
- **Can be used for classification or regression tasks**



Decision Trees

- **Boosted Decision Trees: Iteratively train trees on data weighted by error of the linear combination of previous trees**
- **Random Forests: Produce many uncorrelated decision trees with bootstrapping and then combine their outputs with an ensemble method**
- **Good for when you want to use high level information (previous reconstructions, for example)**
- **Very fast to train**
- **Python supports multiple packages for training decision trees: scikit-learn, XGBoost**

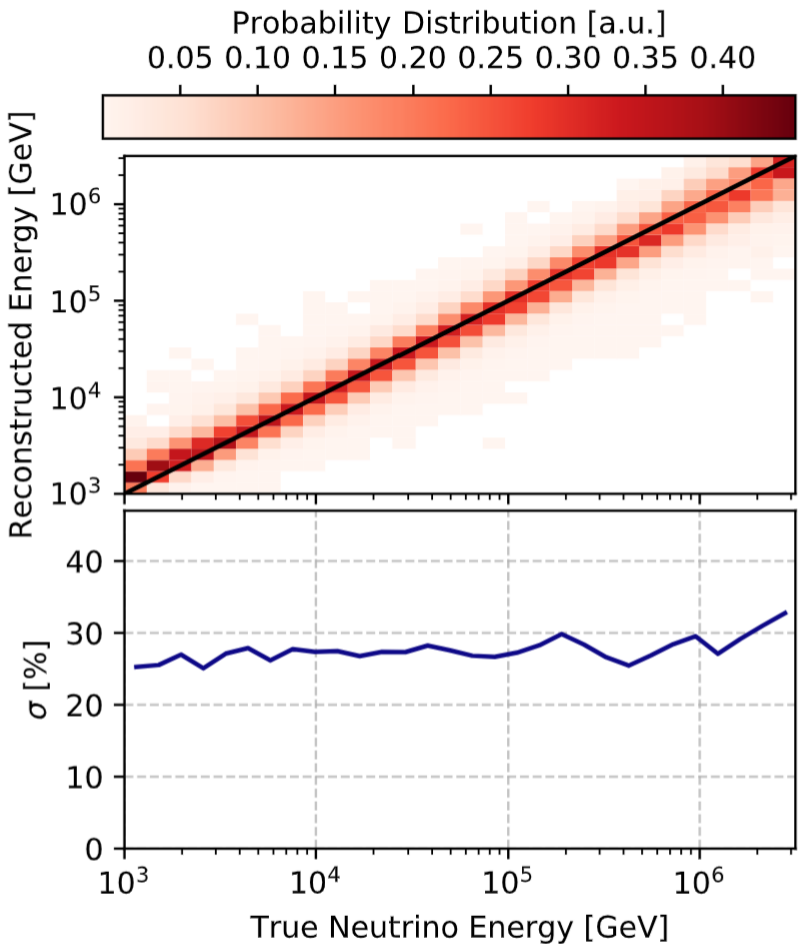




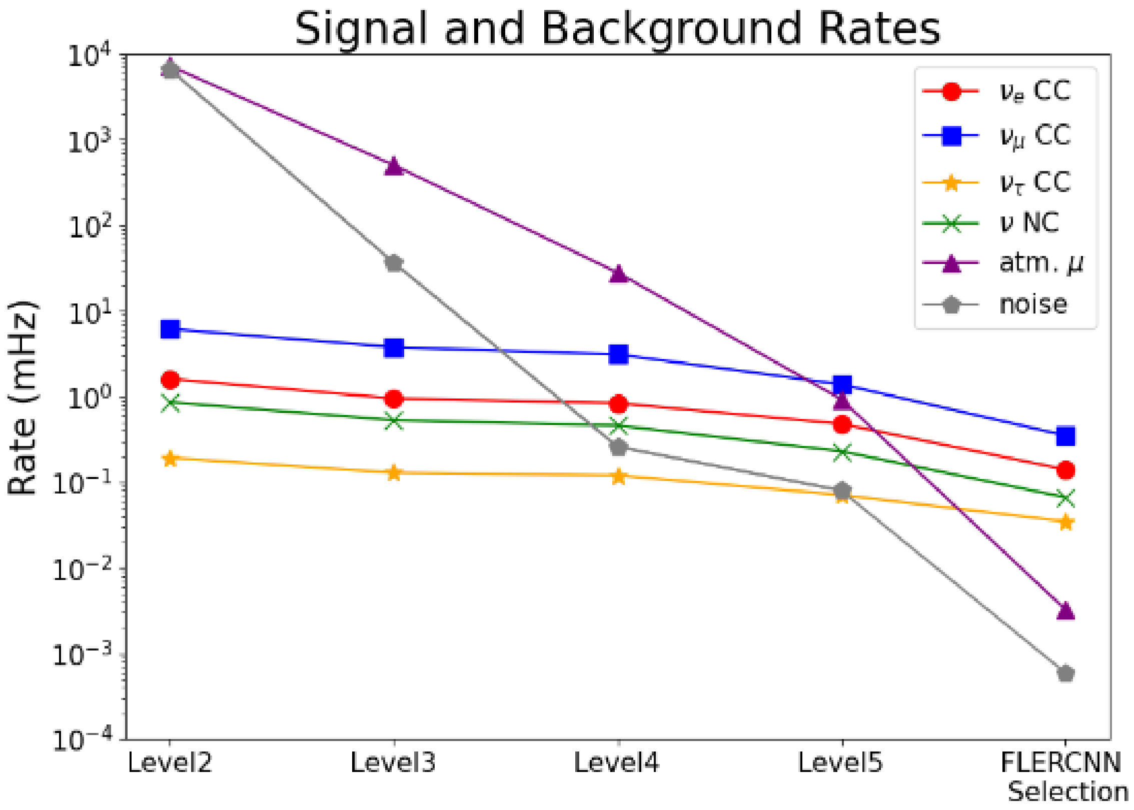
https://www.nbi.dk/~petersen/Teaching/ML2022/Week1/ML2022_DecisionTrees_XGboost.pdf

Decision Tree Examples

Energy Reconstruction



OscNext Event Selection

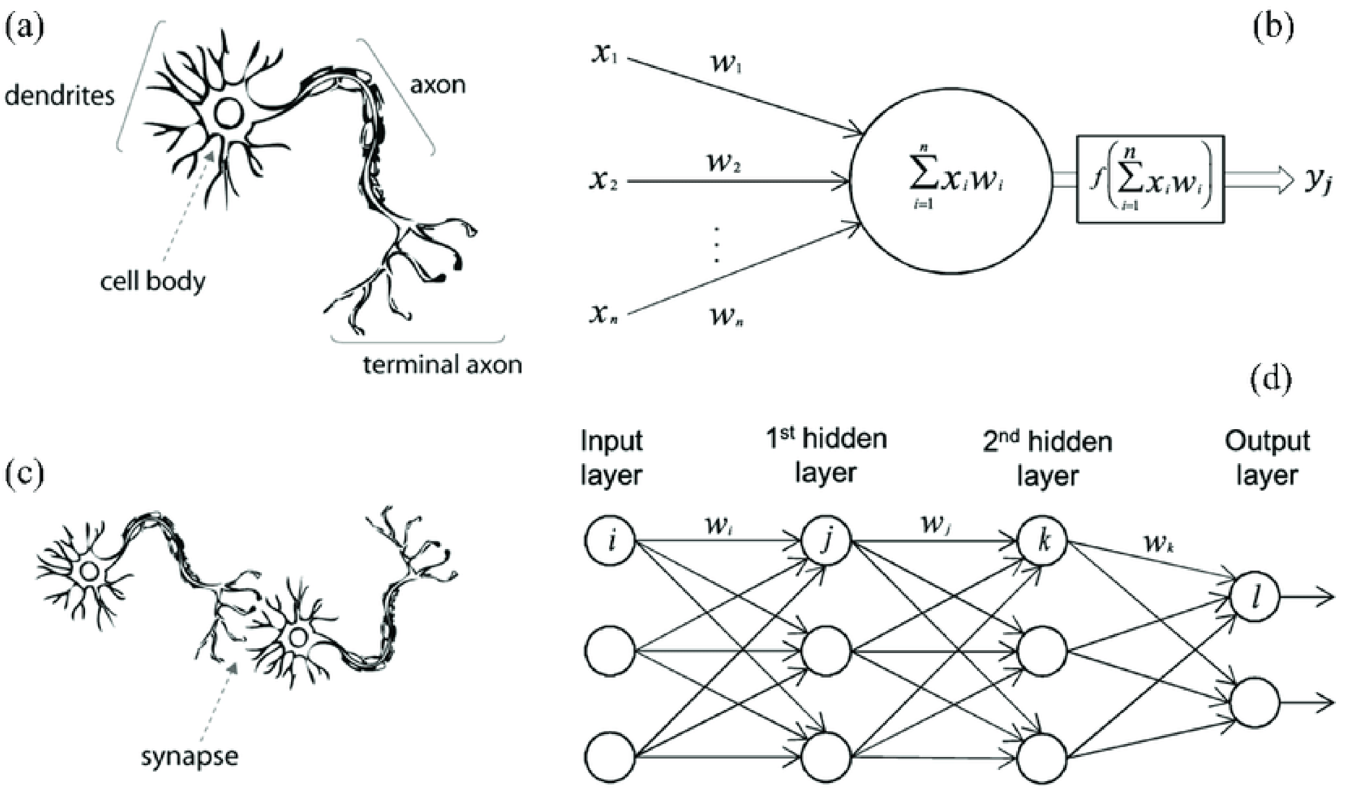


Outline

- Machine learning introduction and general methods
- Decision Trees
- **Neural Networks**

Neural Networks

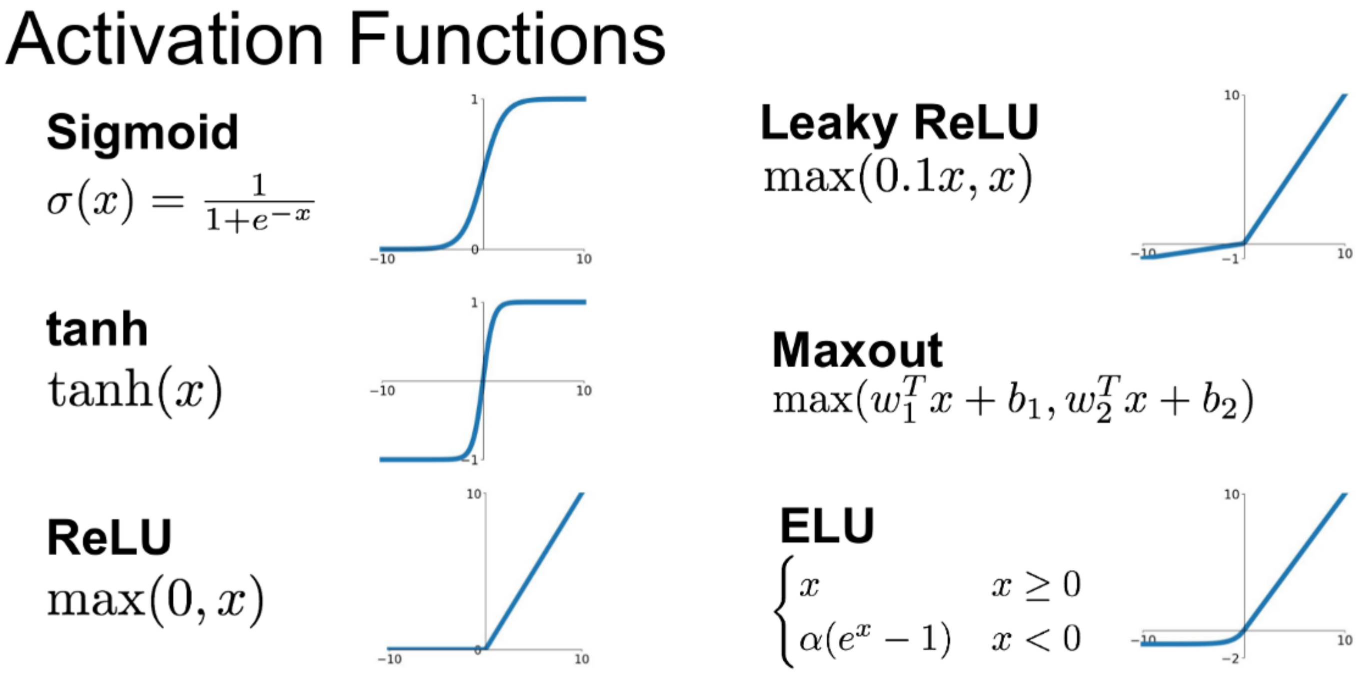
- **An artificial brain**
- **Series of neurons and connections**
    - Each neuron uses the output of all connected neurons from a previous layer as input
    - Each connection has an associated weight
    - Linear combination of inputs is passed to a non-linear activation function
- **The weights and biases are what are learned**



https://www.researchgate.net/figure/A-biological-neuron-in-comparison-to-an-artificial-neural-network-a-human-neuron-b_fig2_339446790

Activation Functions

- **An activation function is used to produce non-linearity in the neural network**
- **Tend to be chosen to have an "on/off" type behavior, like a neuron**
- **For deep neural networks, ReLU is a good first choice to use**
    - Does not have saturation / second order gradient issues



https://medium.com/@shrutijadon/survey-on-activation-functions-for-deep-learning-9689331ba092

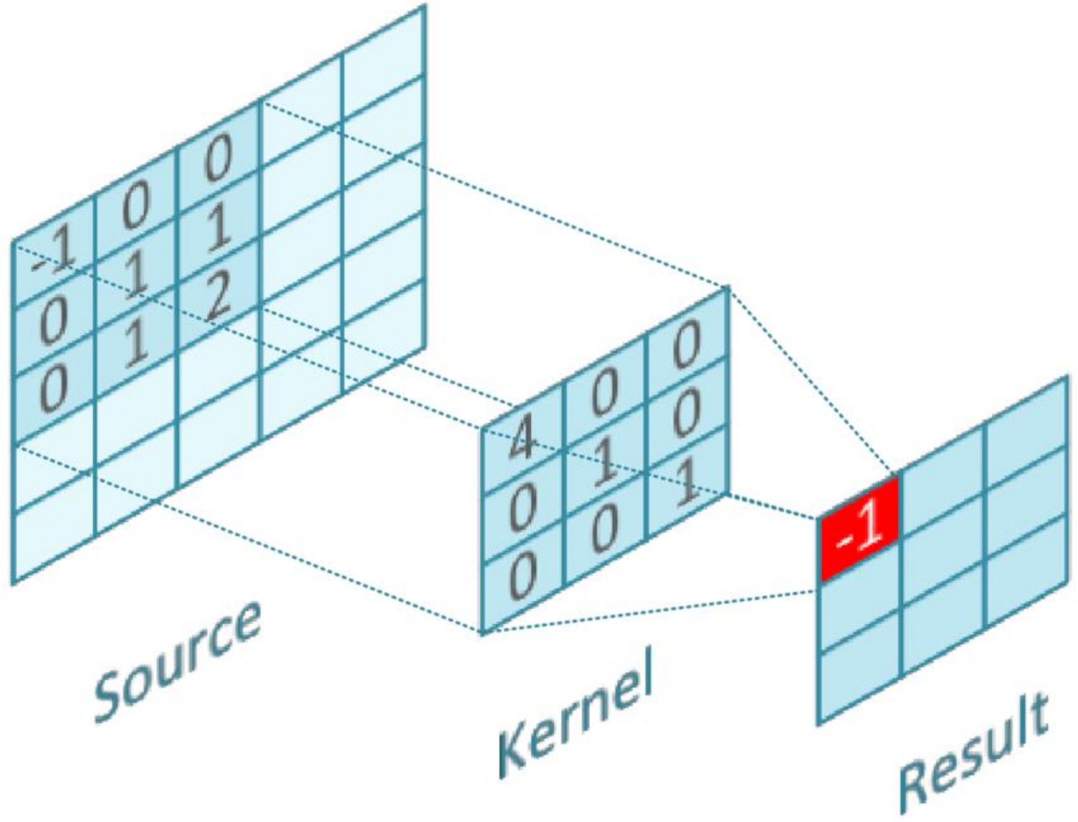
Neural Network Architectures

- **There are many kinds of neural networks, they are usually designed to exploit different features of data**

- **Multilayer Perceptron**
  - The simplest neural network (shown in slide 17)

- **Convolutional Neural Networks (CNNs)**
  - Good for uniform data with translational symmetry

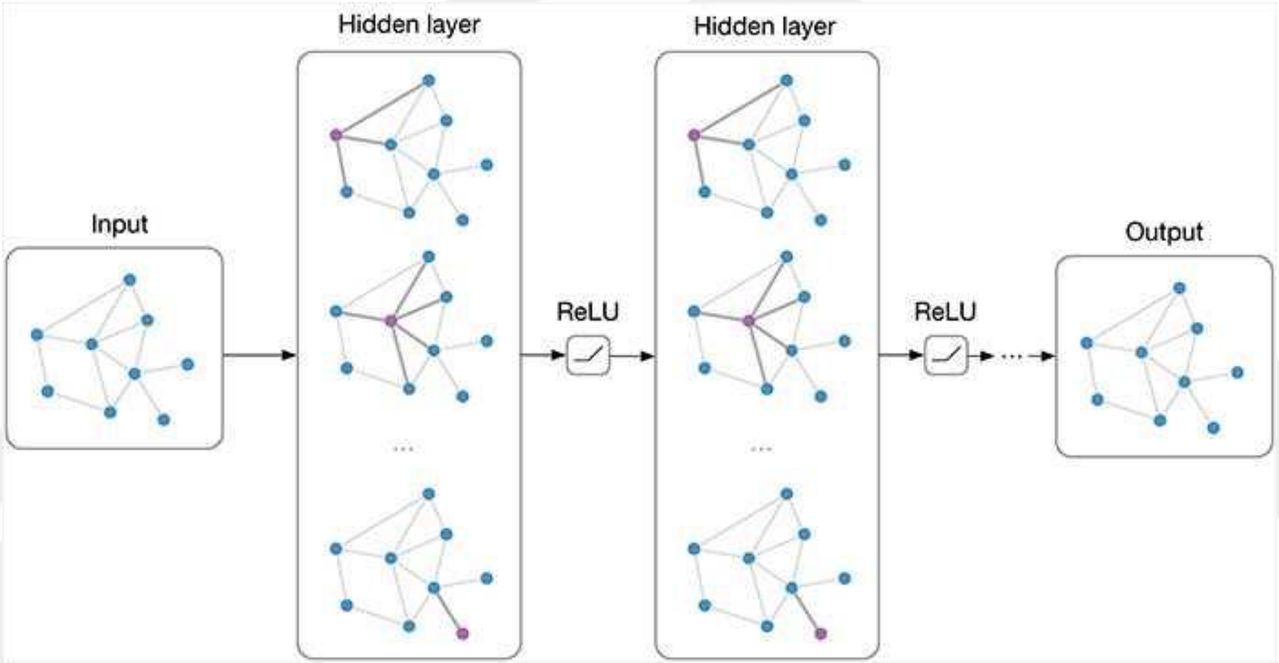
- **Recurrent Neural Networks (RNNs)**
  - Good for data with a specific sequence



https://www.researchgate.net/figure/A-valid-convolution-of-a-5x5-image-with-a-3x3-kernel-The-kernel-will-be-applied-to_fig5_322505397

Neural Network Architectures

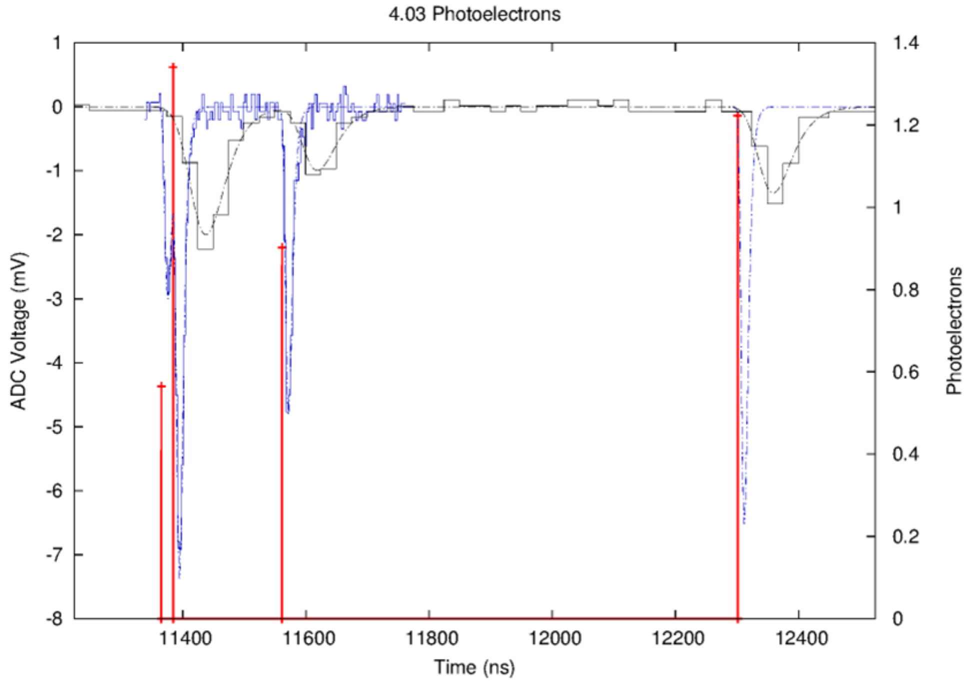
- **More advanced architectures:**
- **Graph Neural Networks (GNNs)**
  - Good for data that can be naturally described as a graph (point cloud data, for example)
- **Transformers**
  - Better version of RNN, CNN

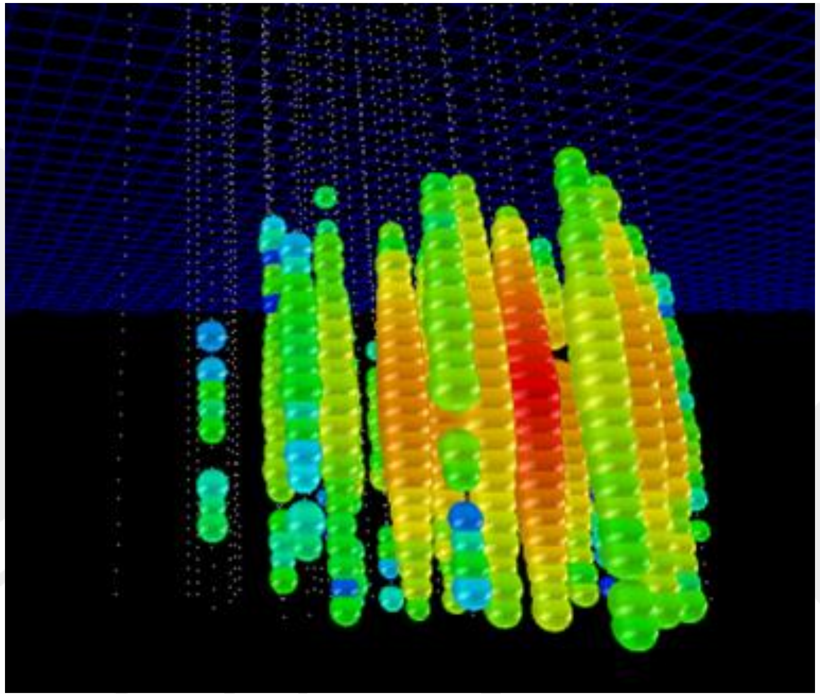


https://theaisummer.com/Graph_Neural_Networks/

Neural Networks

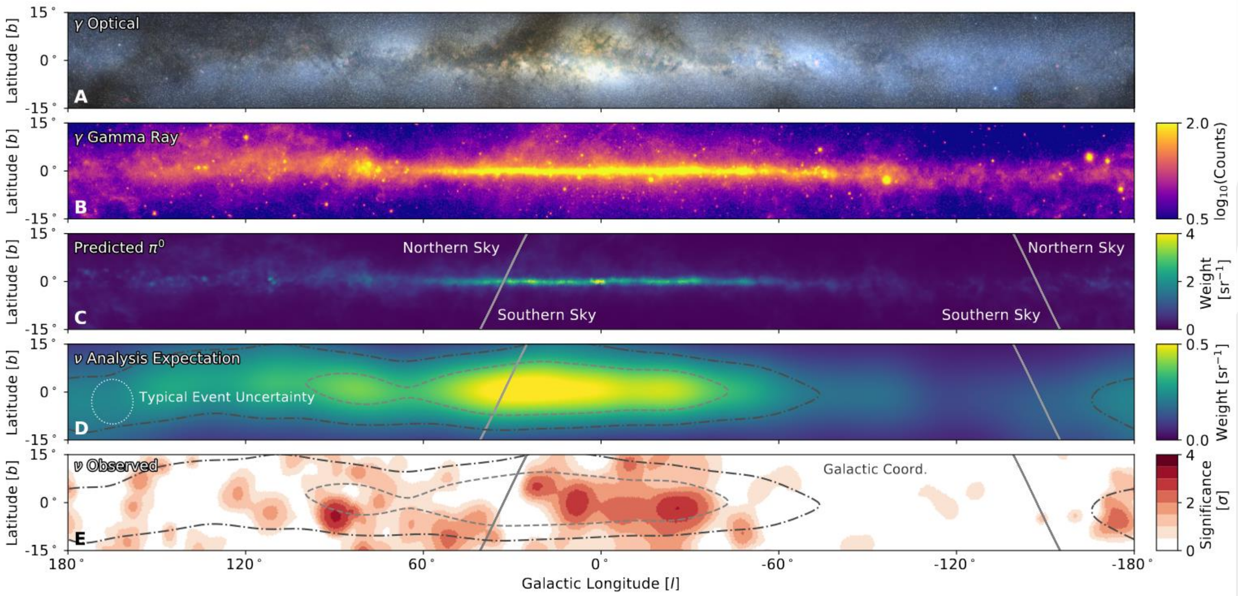
- **Good for low level data / more complex data structures (photon hits, voltage waveforms, etc.)**
- **Can pick up on features of the data that may be missed by traditional reconstruction methods**
- **There are many packages that support training and using neural networks**
  - **Tensorflow / Keras**
  - **PyTorch**
  - **GraphNet**
- **For large amounts of data or large models a GPU may be needed for training in a reasonable amount of time**



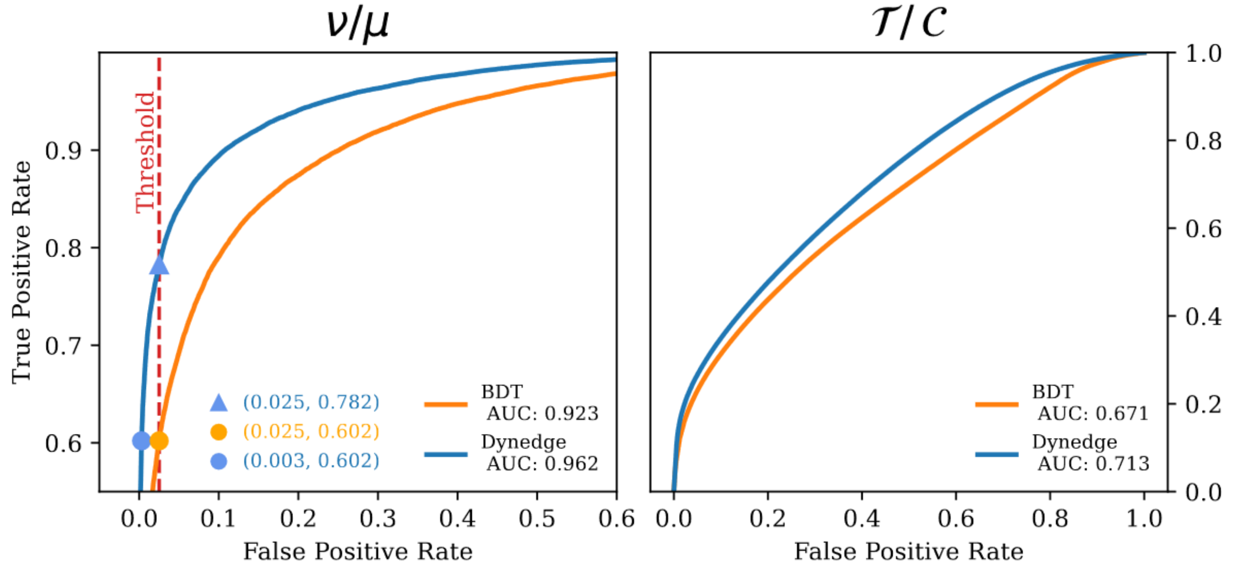


Examples in IceCube

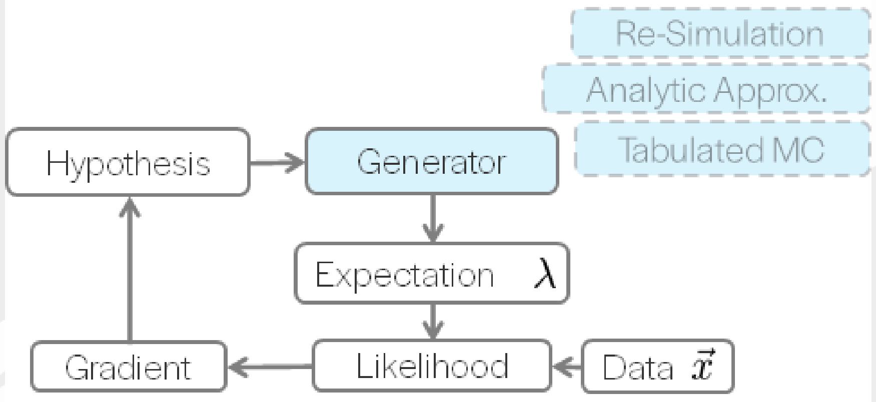
DNN for cascade direction reconstruction



Dynedge (GNNs)



Light Yield Generative CNN



Questions?