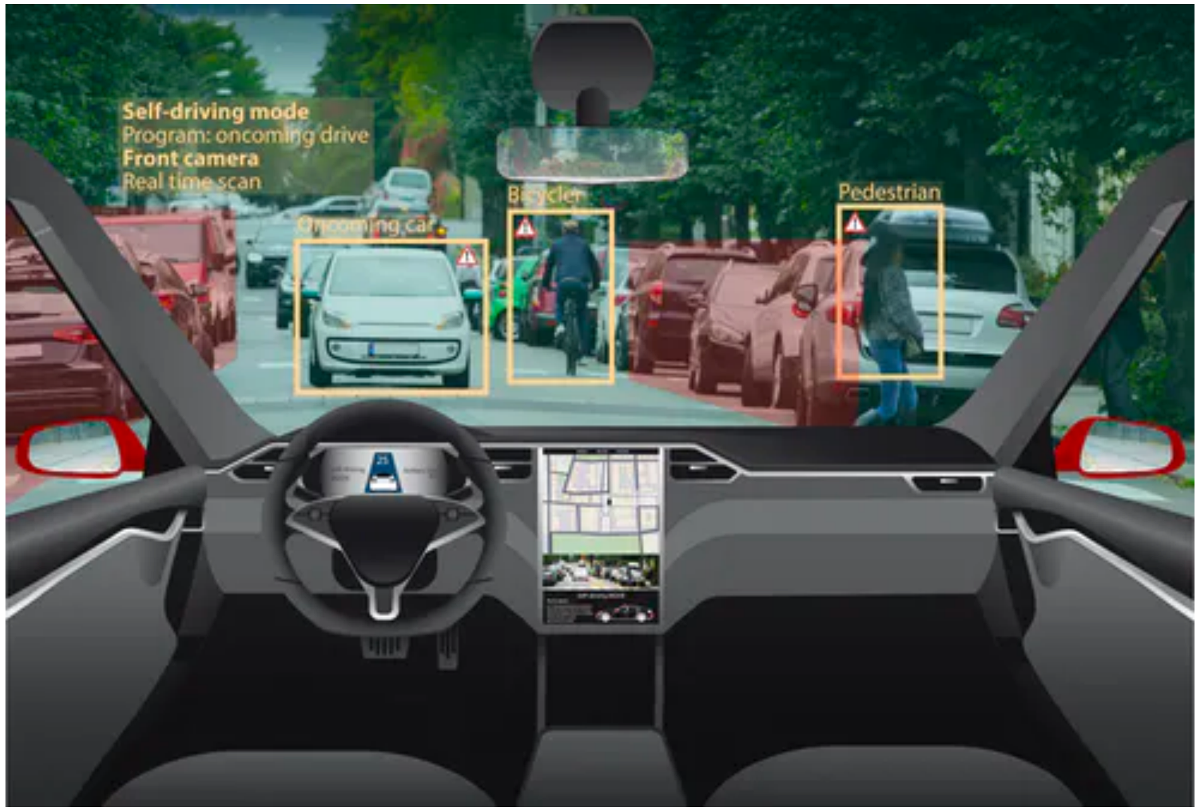
Deep Learning

Lecture 1 NNPSS



Phil Harris
Stolen from Dylan Rankin

Format of these lectures

- First two lectures are going to be hands on
  - We will cover the very basics of deep learning
    - What is going on under the hood
    - How can you build your own NN from scratch
  - Then we will go step by step towards how to train
    - Lecture 1 you learn : How do I train an NN classifier
    - Lecture 2 you learn : How do I train an NN regression

Format of these lectures

- Lecture 3
  - This will cover an overview of what is going on in field
  - We will talk about state of the art uses
    - Graph NNs and friends
    - Anomaly detection
    - Real-time operations
- This last lecture is a survey of the field

What Will We Cover Today

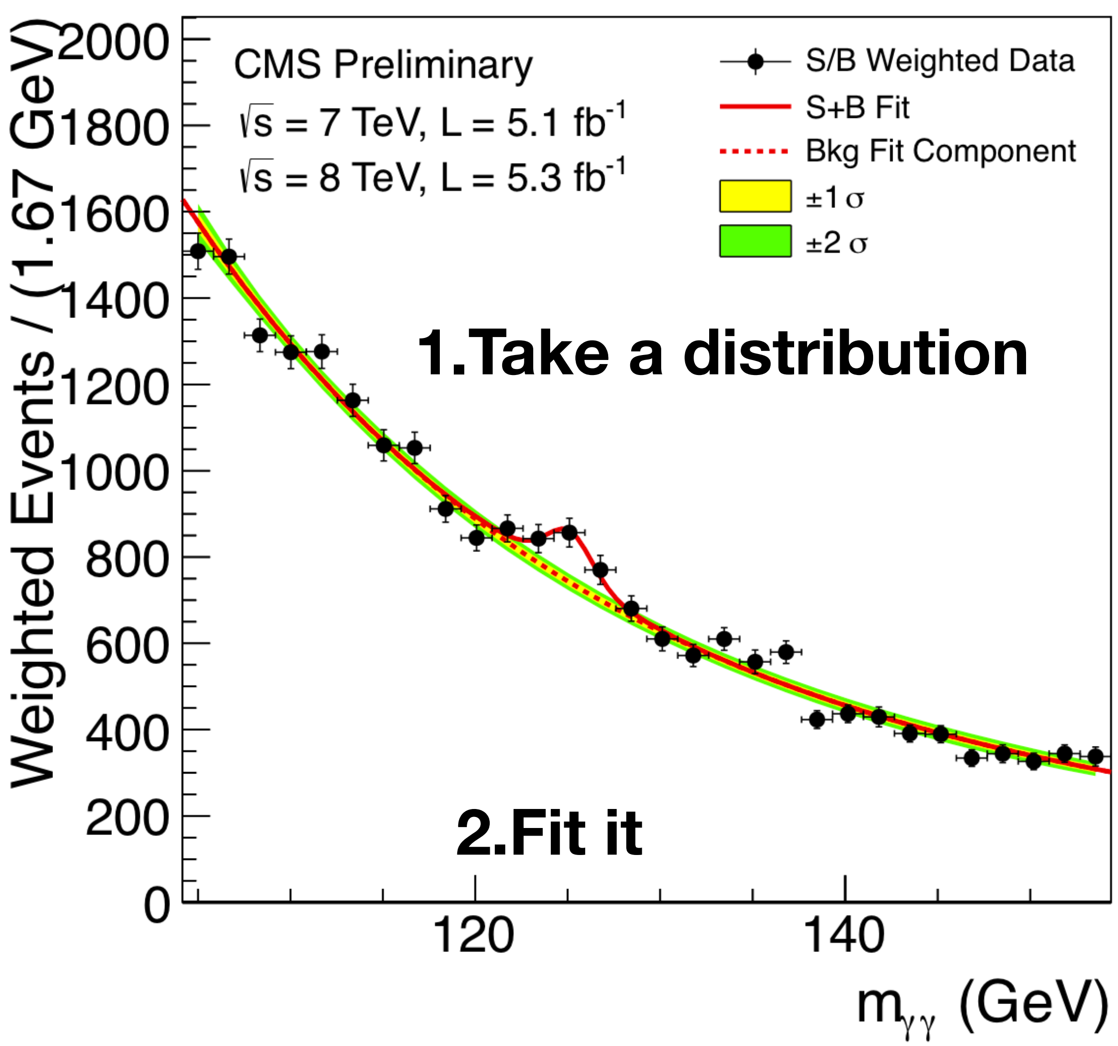
- Slides are straight out of a class I teach on data science+physics
    - This class will be available online in the fall
- What is a neural network?
    - Historical context
    - Why do they work?
- How does a neural network “learn”?
- How can neural networks be designed?

Key Terms

- Supervised Learning
    - Classification
    - Regression
- Unsupervised Learning
    - Clustering
    - Dimensional Reduction
- Architectures
    - Linear Models
        - Perceptron, support vector machine, logistic regression
    - Neural network
- Training
    - Backpropogation
    - (Stochastic) gradient descent

Origins of Machine Learning

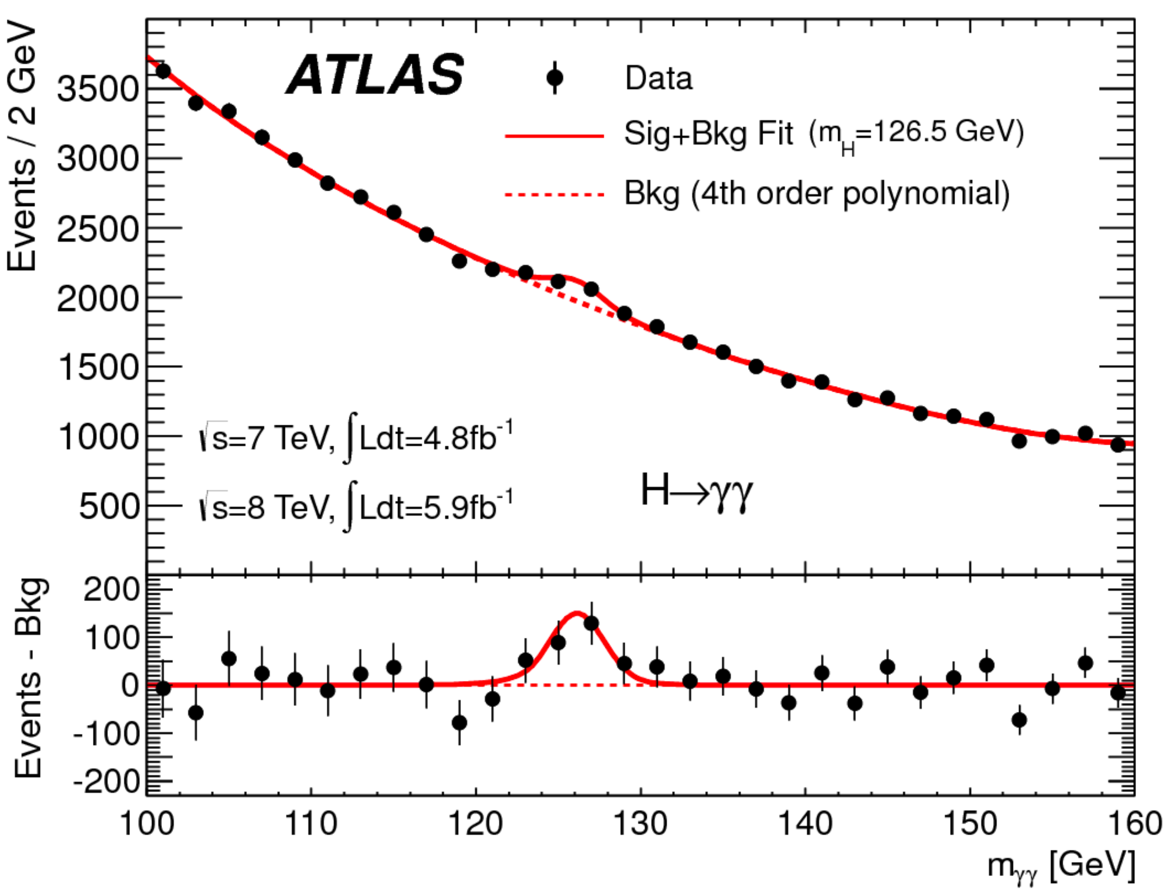
- I assume you are all familiar with fitting



3. Change physics forever

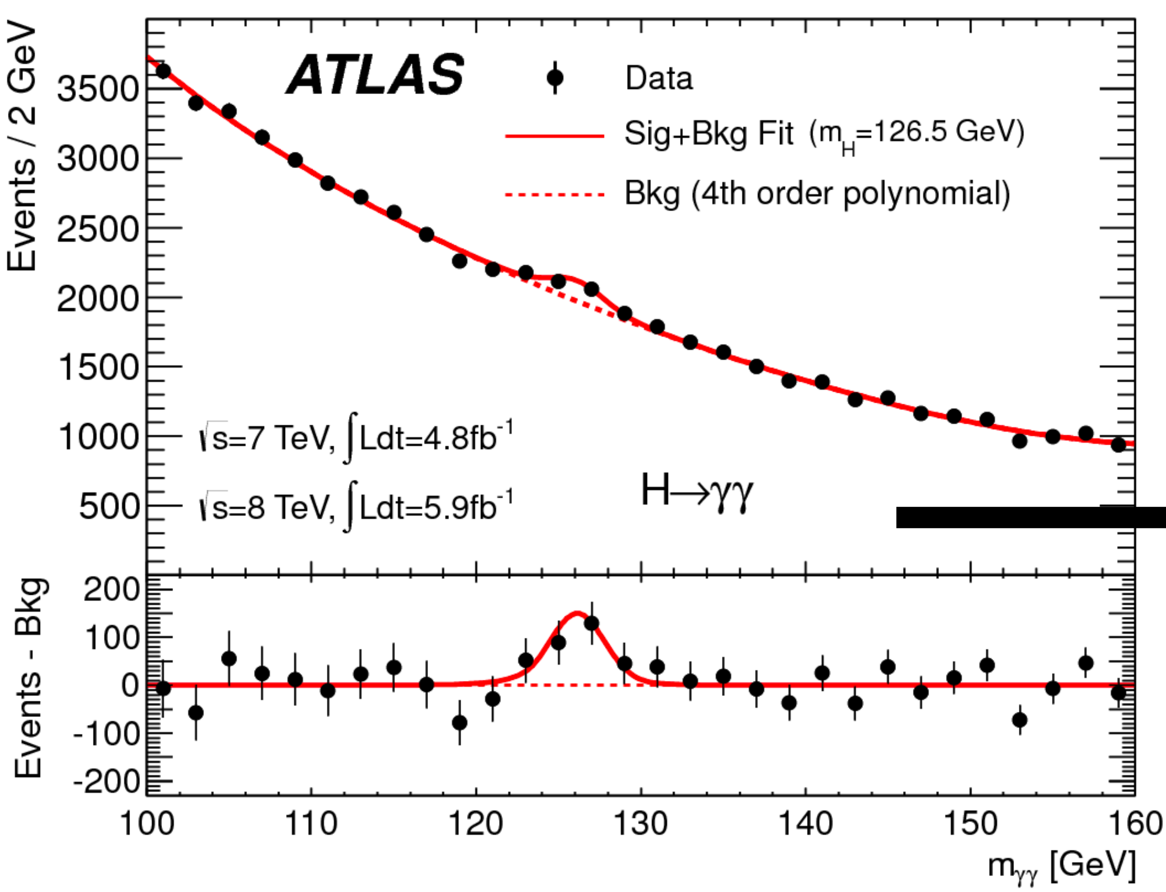
What is the right fit function?

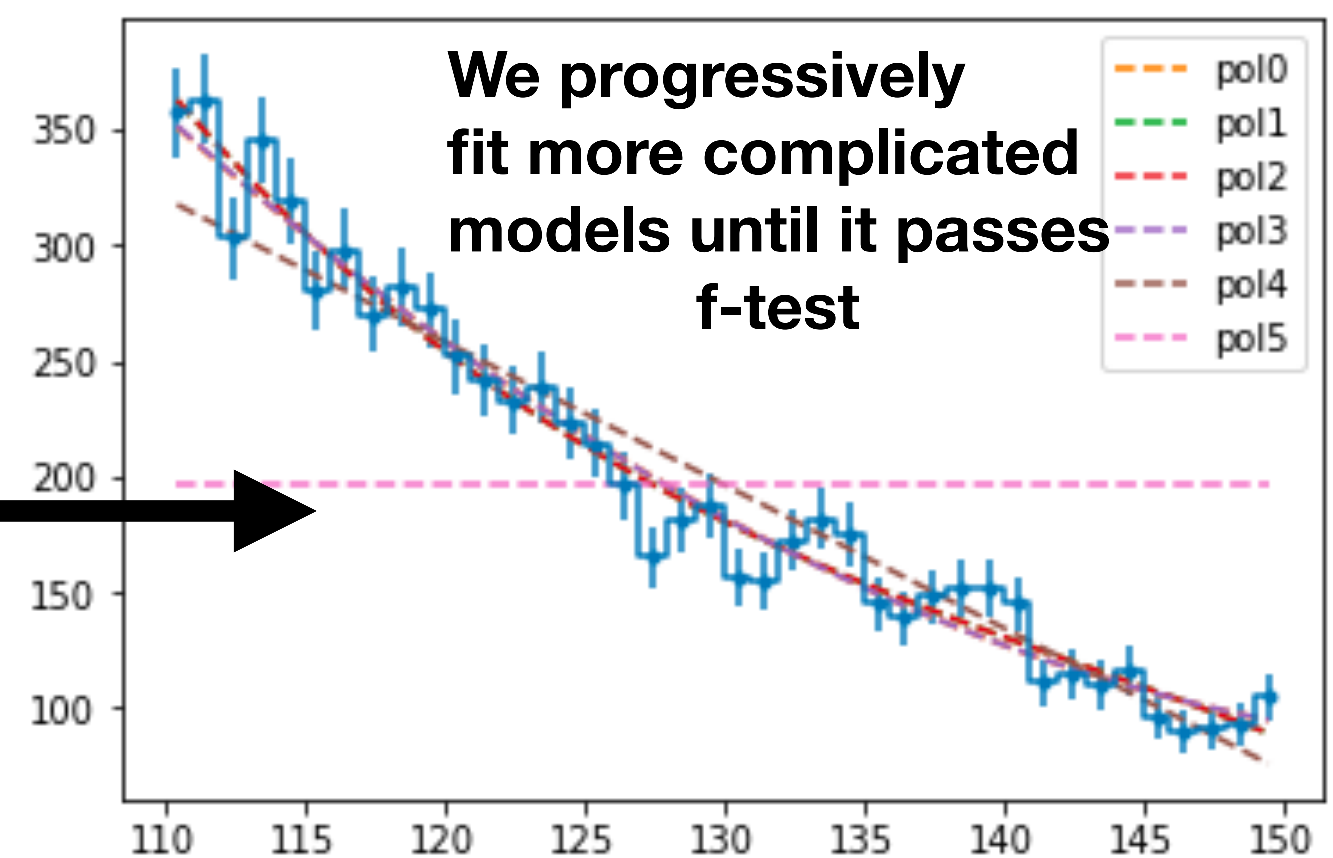
- When we are trying to fit for the Higgs boson data?
  - We need a signal model and a background model
  - How do we determine the right backtround model?



What is the right fit function?

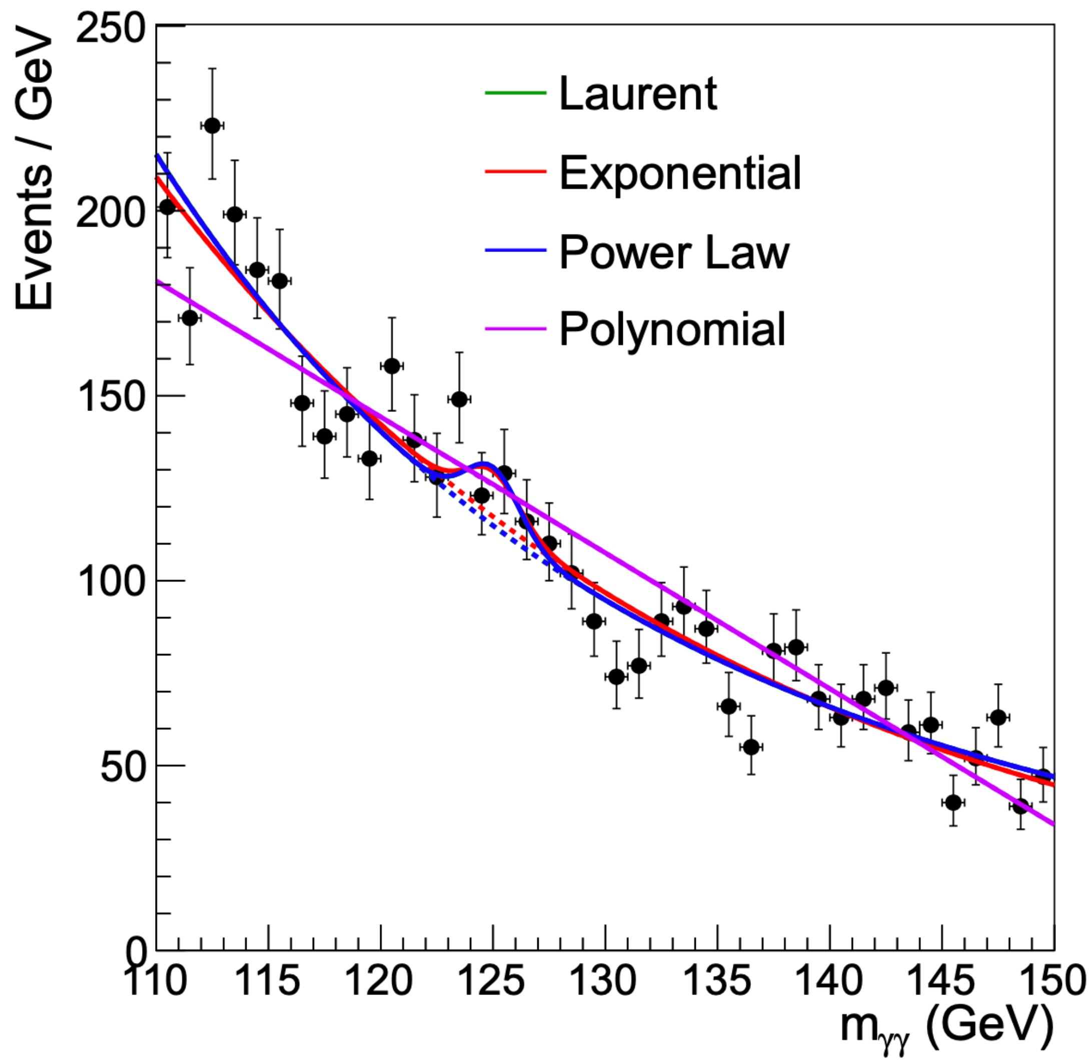
- When we are trying to fit for the Higgs boson data?
  - We need a signal model and a background model
  - How do we determine the right backtround model?

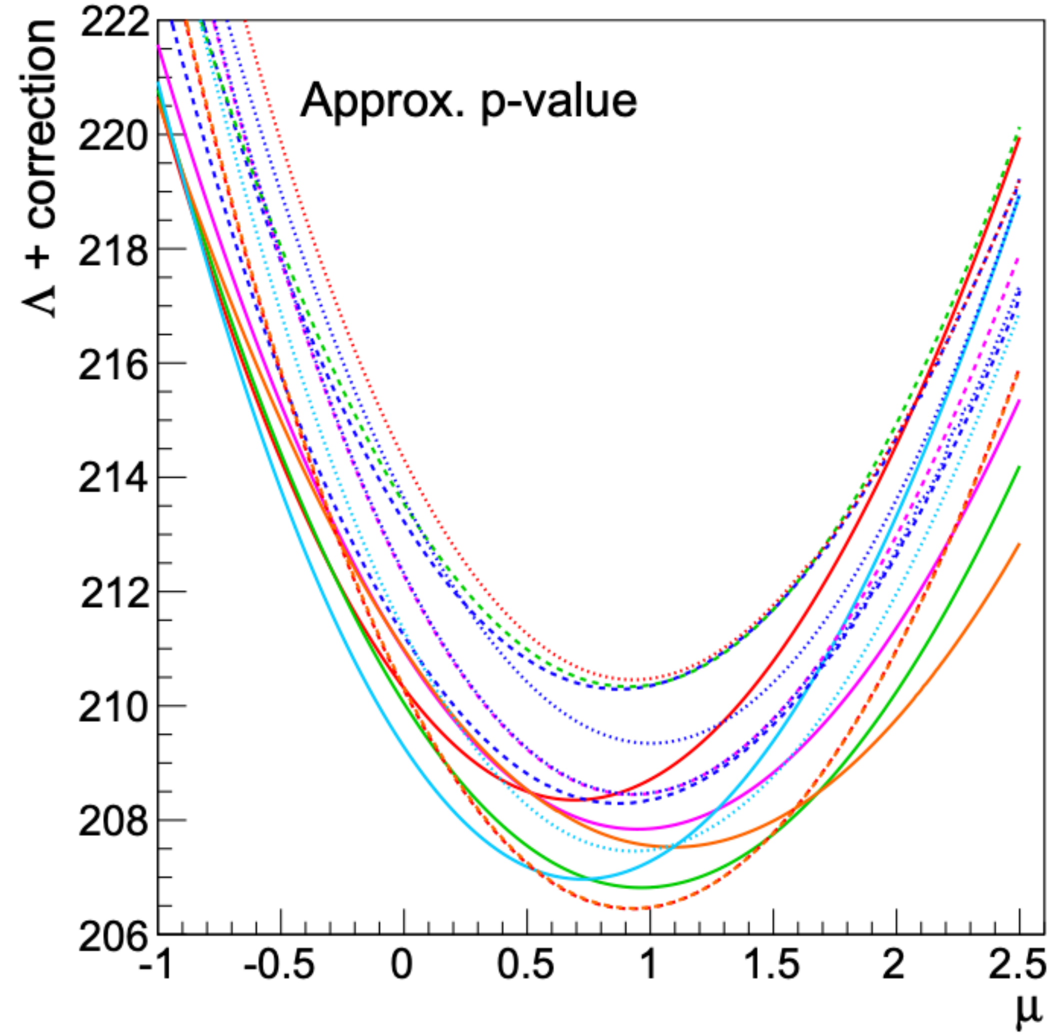




Building a Model

- Throwing a barrage of functions at the problem

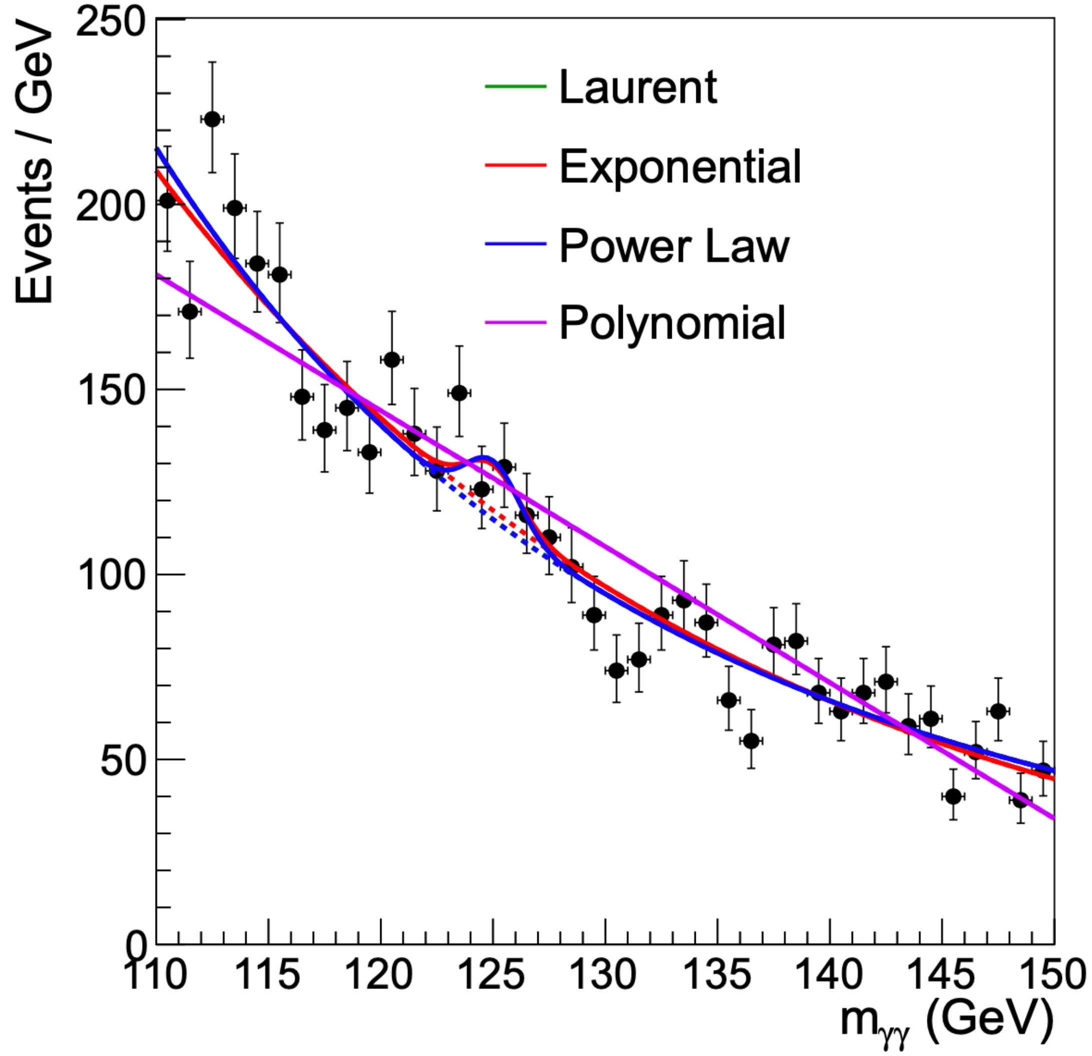


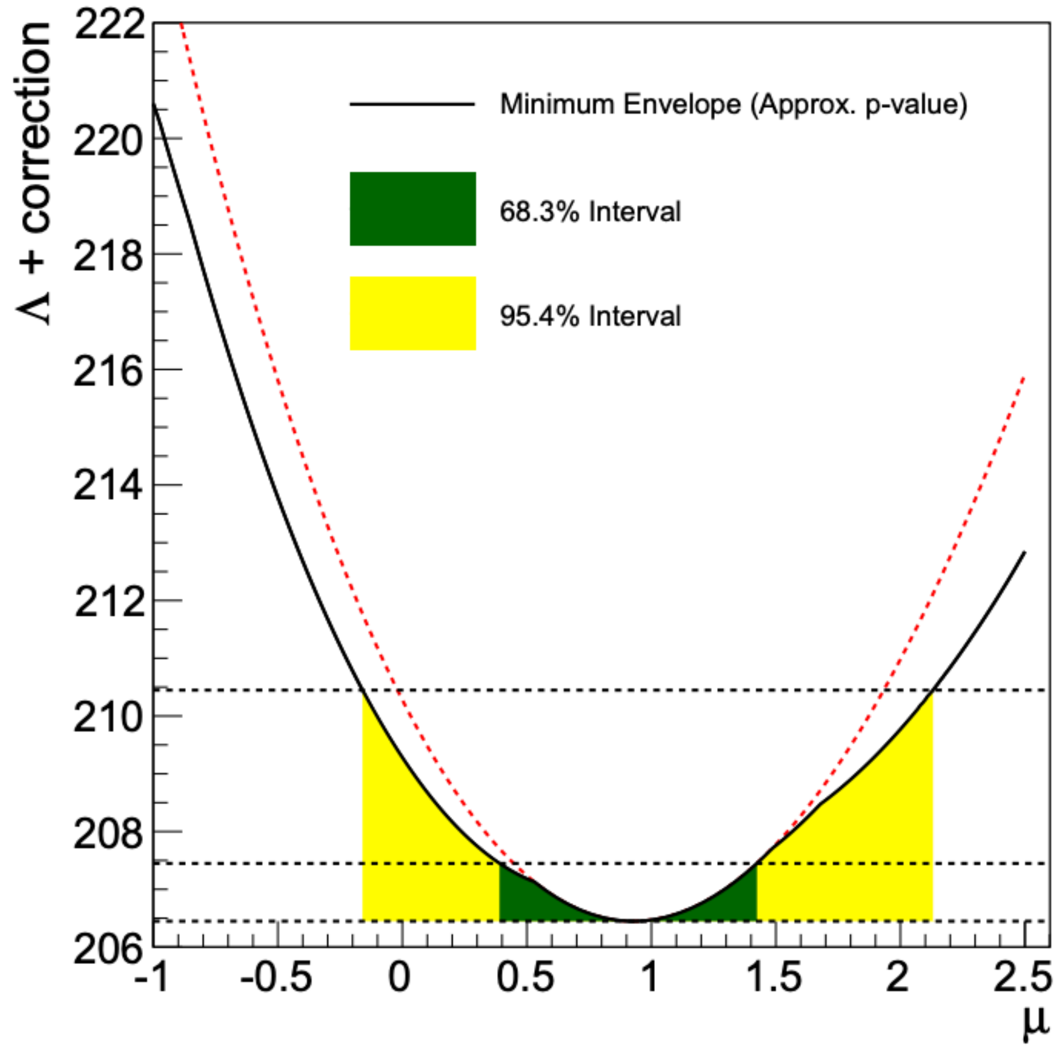


We can try a whole library of functions
The likelihood we get translates to our fit

Building a Model

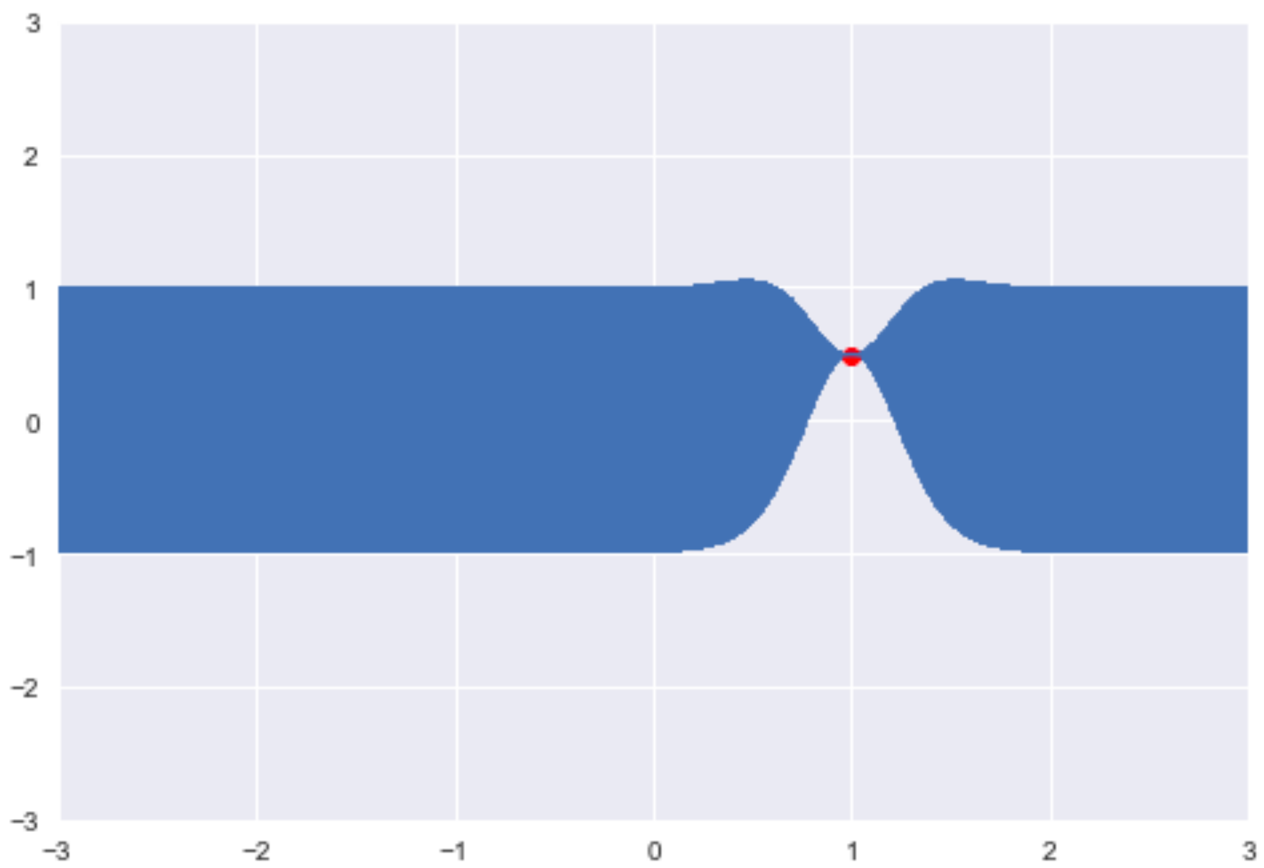
- Throwing a barrage of functions at the problem





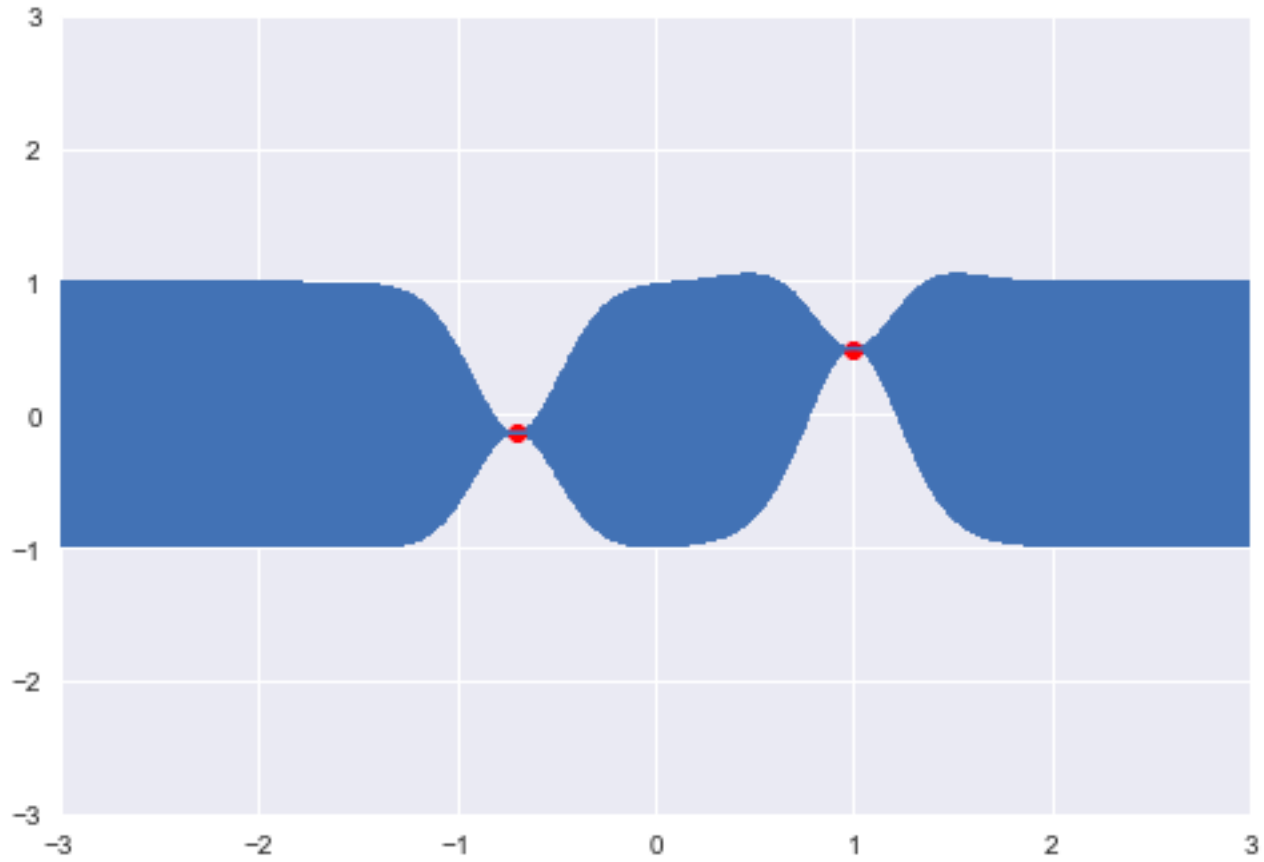
We can try a whole library of functions
The likelihood we get translates to our fit

Splines+GaussianProcesses



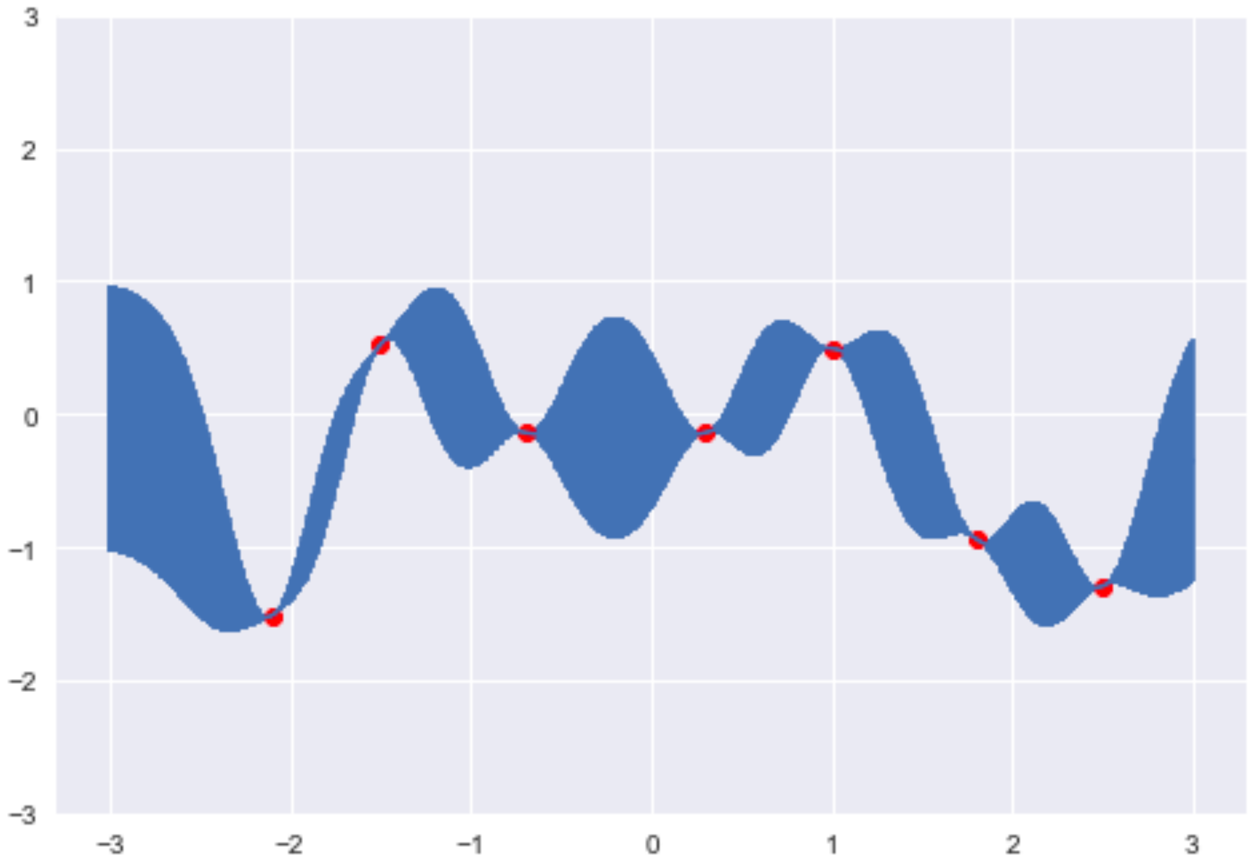
- Gaussian processes allow us to build function choice from the data

Splines+GaussianProcesses



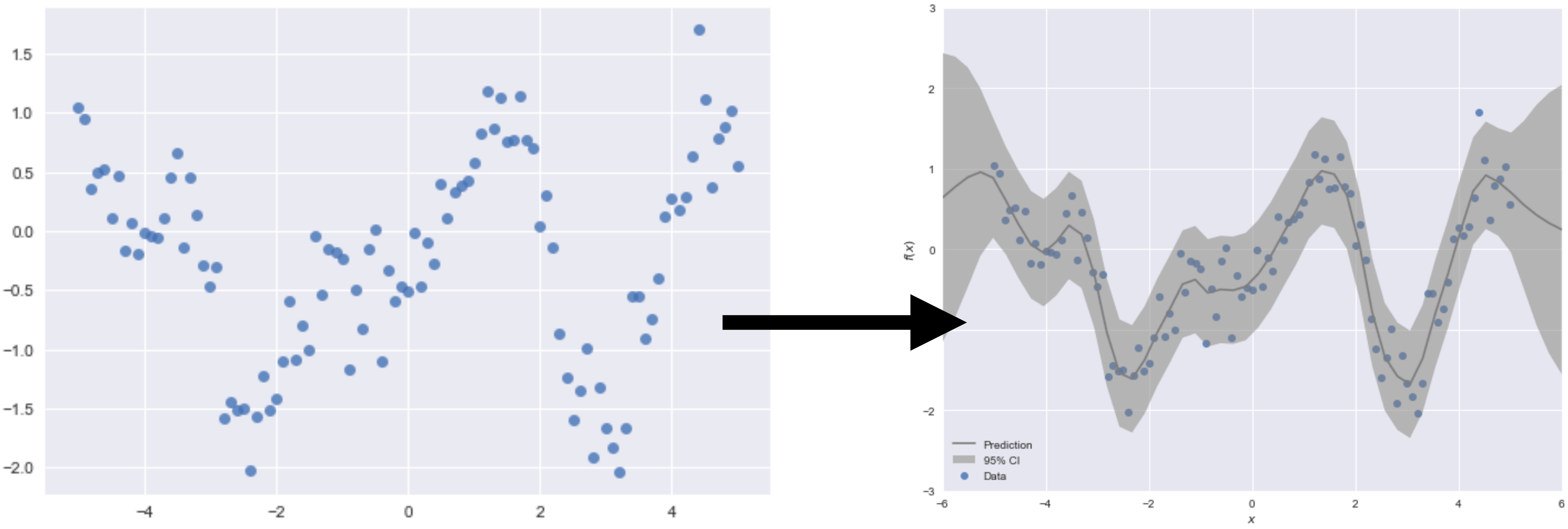
- Gaussian processes allow us to build function choice from the data

Splines+GaussianProcesses



- Gaussian processes allow us to build function choice from the data

Splines+GaussianProcesses

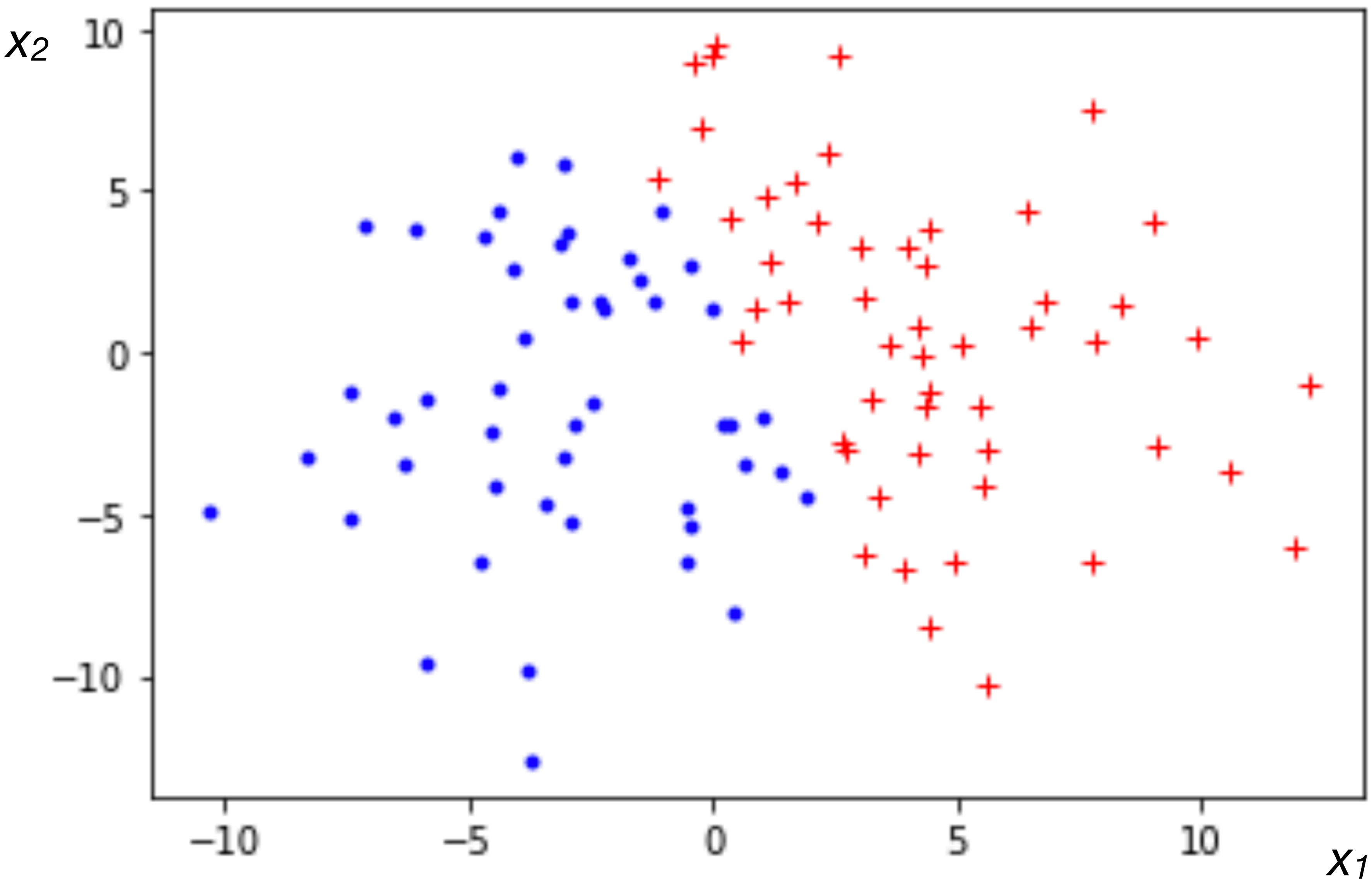


- How do we fully automate this whole procedure of approximation?

Google Collab

https://colab.research.google.com/github/MIT-8s50/course/blob/main/Lecture10/deeplearning.ipynb

Simple Machine Learning

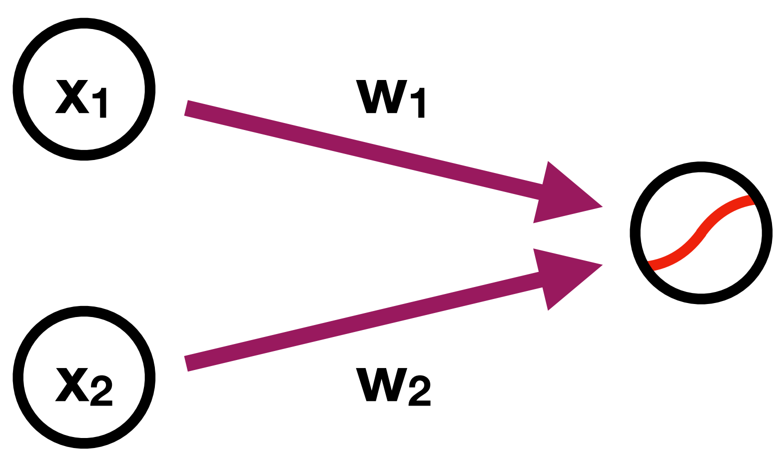


How can I predict if a point is red or blue given x_1 and x_2?

(Lets use the notebook)

Logistic Regression

- Simplest neural network
  - No hidden layers
  - Two inputs
  - One output neuron with a sigmoid activation.

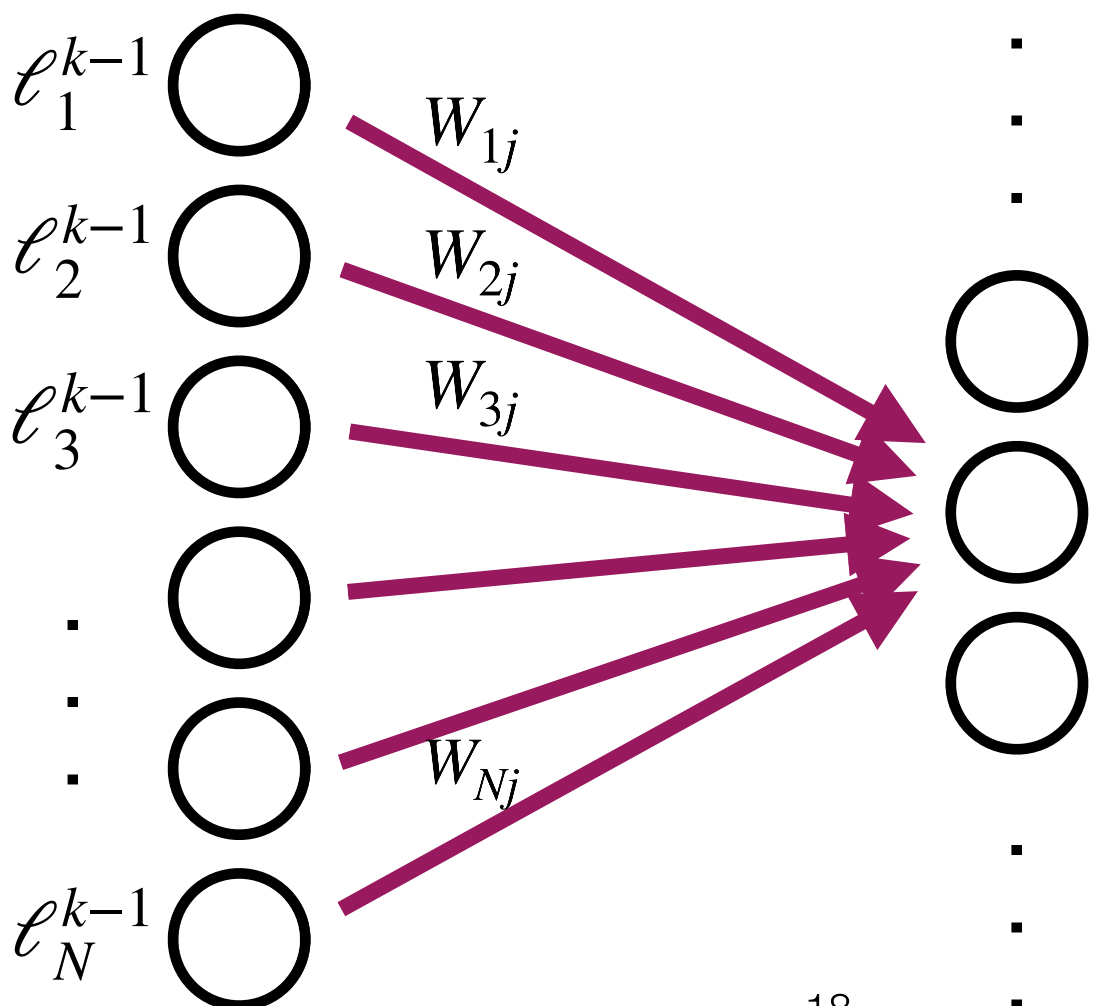


$$\ell = \frac{1}{1 + e^{\mathbf{w}^T\mathbf{x}+b}}$$

$$\mathbf{w}^T = (w_1 \;\; w_2)$$

Neural Network

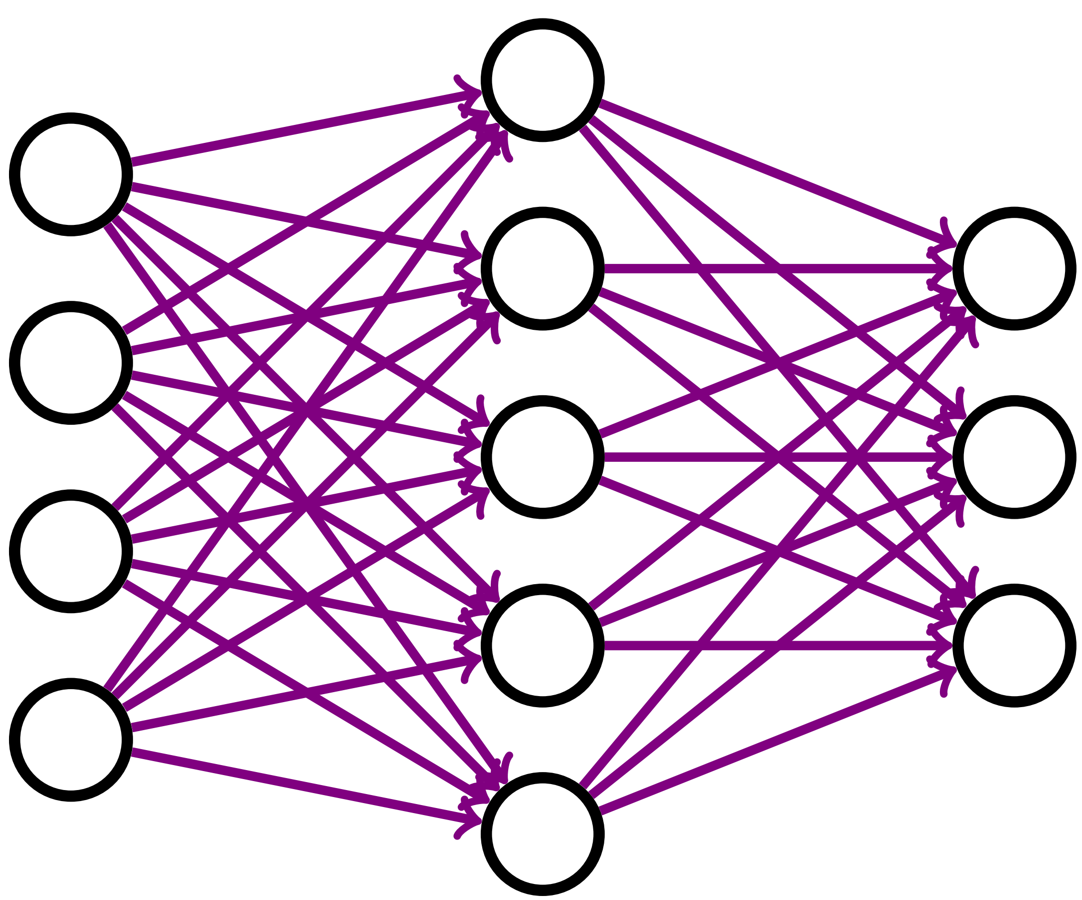
- Generically composed of neurons which receive inputs with *weights (W)* and a *bias (b)*, and pass outputs based on an *activation function (ϕ)*



$$\ell_j^k = \phi(W_{ij}\ell_i^{k-1} + b_j)$$

Dense Neural Network

- **Multiple layers:** output of previous layer is **fed forward** to next layer after applying **non-linear** activation function
- **Fully connected:** many independent weights
- **Learning:** Use analytic derivatives and stochastic gradient descent to find optimal weights

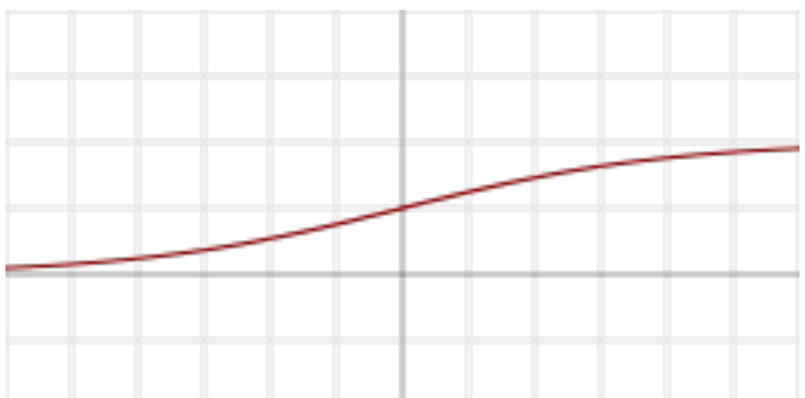


Architecture:

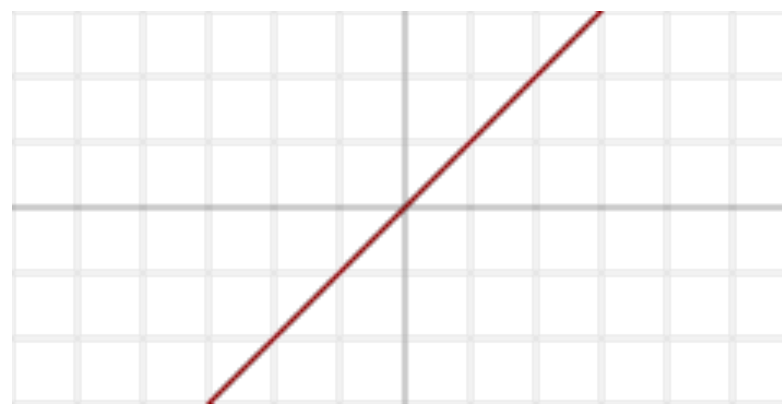
Dense Network
Fully-connected Network (FC)
Multi-Layer Perceptron (MLP)

Activation Functions

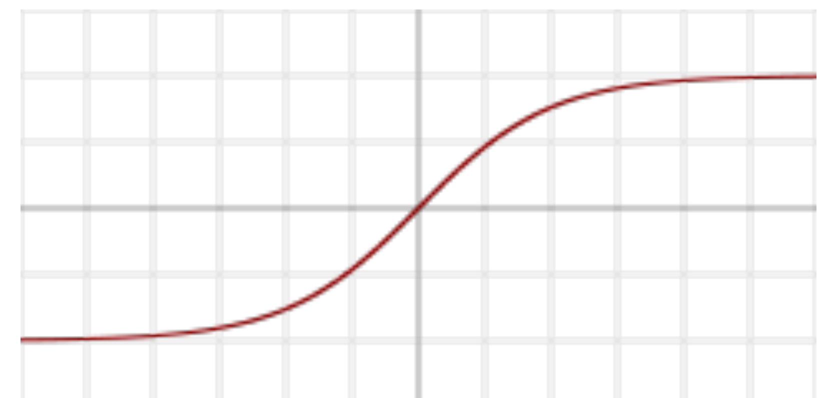
Sigmoid



Linear



Tanh

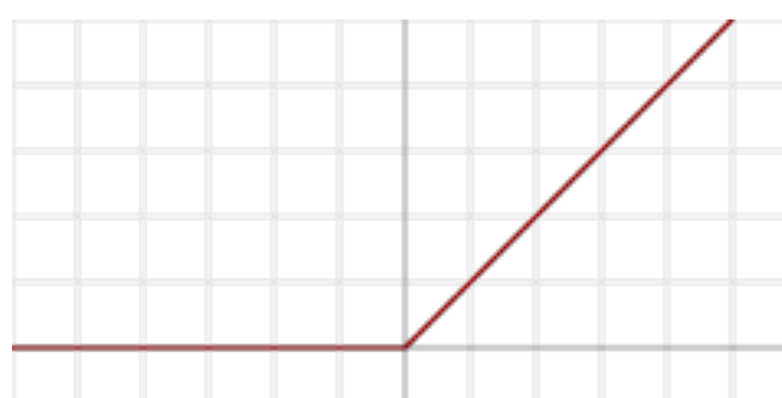


Softmax

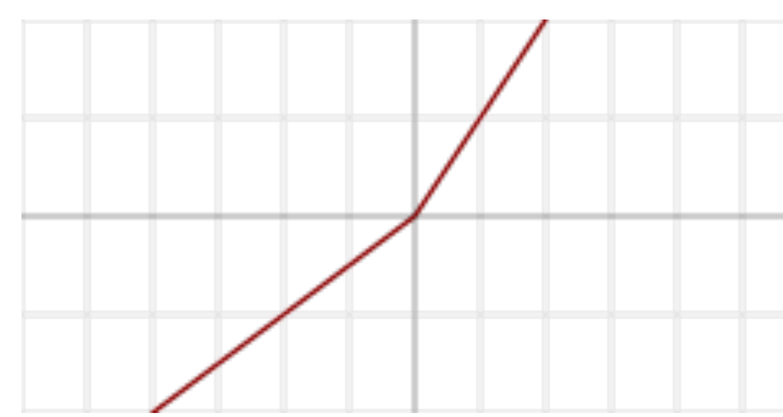
(multiclass)

$$\frac{e^{x_i}}{\sum_{j=1}^{J} e^{x_j}}$$

ReLU

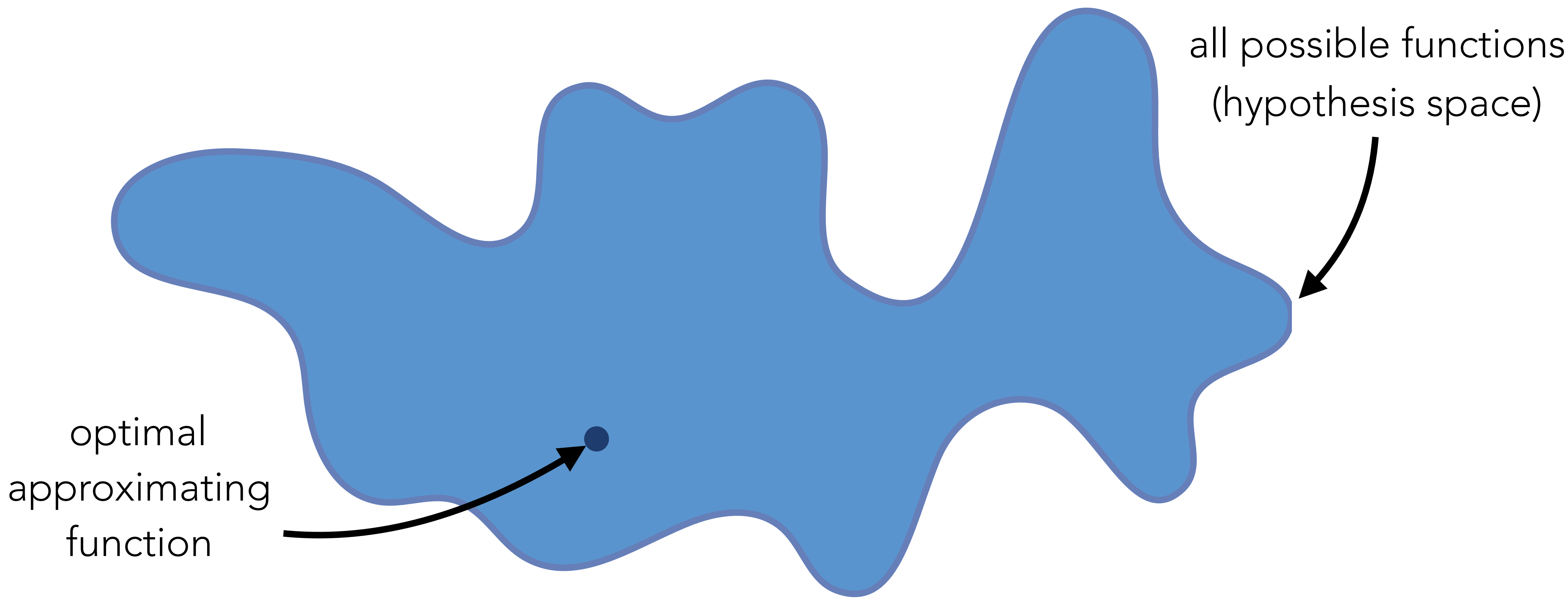


LeakyReLU



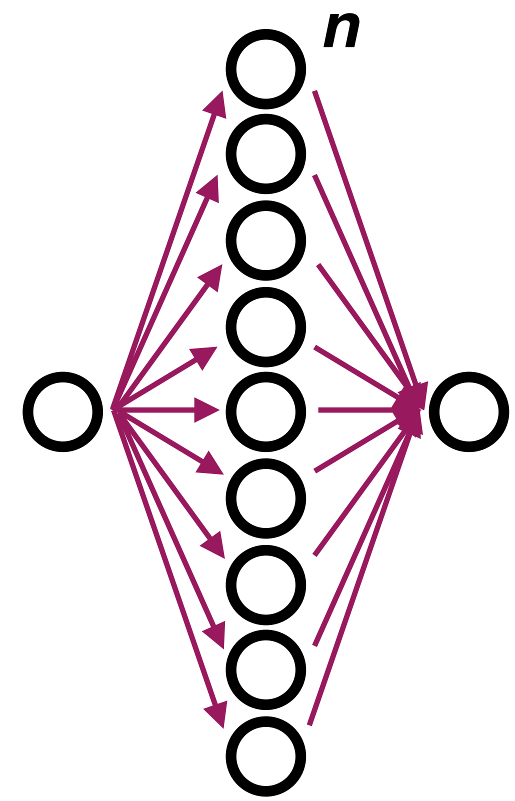
Neural Network

- Neural networks are universal function approximators, but we still must find an optimal approximating function



- We do so by adjusting the weights, architecture

Approximating



- Layer: $\ell(\mathbf{x}) = \phi(\mathbf{W}^T\mathbf{x} + \mathbf{b})$

- Linear+Linear:

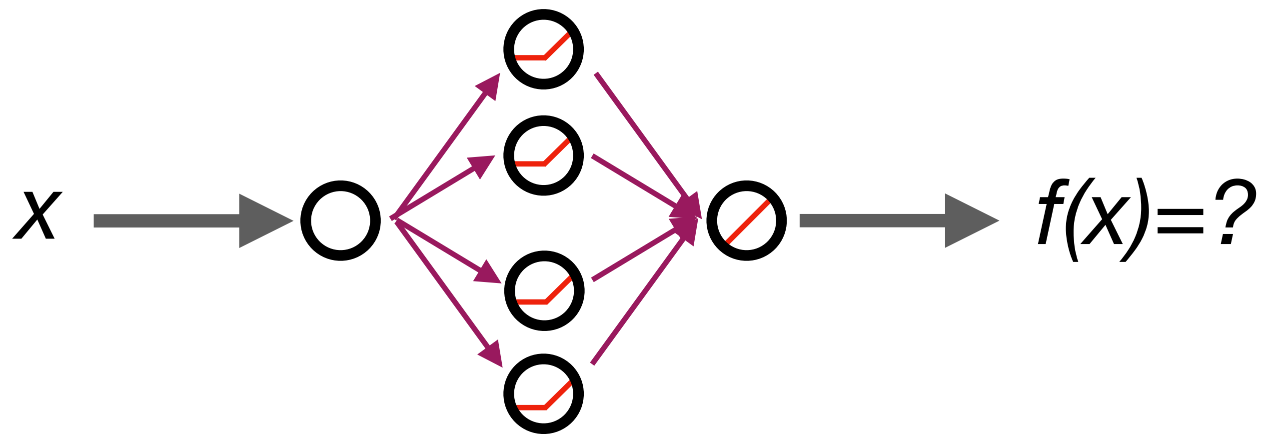
$$\begin{aligned}\ell_{\text{linear}}(\ell_{\text{linear}}(\mathbf{x})) &= \ell_{\text{linear}}(\mathbf{W}_1^T\mathbf{x} + \mathbf{b}_1) \\ &= \mathbf{W}_2^T(\mathbf{W}_1^T\mathbf{x} + \mathbf{b}_1) + \mathbf{b}_2 \\ &= \mathbf{W}_2^T\mathbf{W}_1^T\mathbf{x} + \mathbf{W}_2^T\mathbf{b}_1 + \mathbf{b}_2 \\ \tilde{\ell}_{\text{linear}}(\mathbf{x}) &= \tilde{\mathbf{W}}^T\mathbf{x} + \tilde{\mathbf{b}}\end{aligned}$$

- ReLU+Linear:

$$\ell_{\text{linear}}(\ell_{\text{ReLU}}(\mathbf{x})) = \begin{cases}\tilde{\mathbf{W}}^T\mathbf{x} + \tilde{\mathbf{b}}\ , & \mathbf{W}_1^T\mathbf{x} + \mathbf{b}_1 > 0 \\ \mathbf{b}_2\ , & \mathbf{W}_1^T\mathbf{x} + \mathbf{b}_1 < 0\end{cases}$$

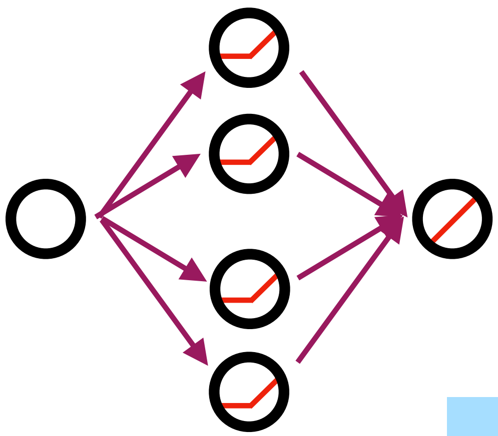
n boundaries

Approximating (Example)



What kind of function can this network architecture approximate?

Approximating (Example)



$$= \tilde{\mathbf{W}}^T\mathbf{x} + \tilde{\mathbf{b}}\ ,\quad \mathbf{W}_1^T\mathbf{x} + \mathbf{b}_1 > 0$$

$$\mathbf{b}_2\ ,\quad \mathbf{W}_1^T\mathbf{x} + \mathbf{b}_1 < 0$$

$\mathbf{W}_2^T\mathbf{W}_1^T\mathbf{x}$ $\quad$ $\mathbf{W}_2^T\mathbf{b}_1$

$$W_{11}^2W_{11}^1x + W_{21}^2W_{12}^1x + W_{31}^2W_{13}^1x + W_{41}^2W_{14}^1x + W_{11}^2b_1^1 + W_{21}^2b_2^1 + W_{31}^2b_3^1 + W_{41}^2b_4^1 + b_1^2$$

$x > -\frac{b_4^1}{W_{14}^1}$ **All** $(W_{11}^2W_{11}^1 + W_{21}^2W_{12}^1 + W_{31}^2W_{13}^1 + W_{41}^2W_{14}^1)x + W_{11}^2b_1^1 + W_{21}^2b_2^1 + W_{31}^2b_3^1 + W_{41}^2b_4^1 + b_1^2$

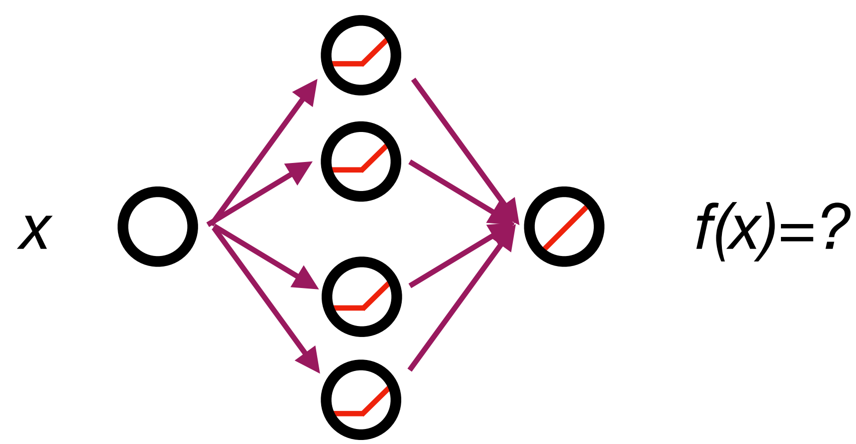
$-\frac{b_4^1}{W_{14}^1} > x > -\frac{b_3^1}{W_{13}^1}$ **Drop One** $(W_{11}^2W_{11}^1 + W_{21}^2W_{12}^1 + W_{31}^2W_{13}^1)x + W_{11}^2b_1^1 + W_{21}^2b_2^1 + W_{31}^2b_3^1 + b_1^2$

$-\frac{b_3^1}{W_{13}^1} > x > -\frac{b_2^1}{W_{12}^1}$ **Drop Two** $(W_{11}^2W_{11}^1 + W_{21}^2W_{12}^1)x + W_{11}^2b_1^1 + W_{21}^2b_2^1 + b_1^2$

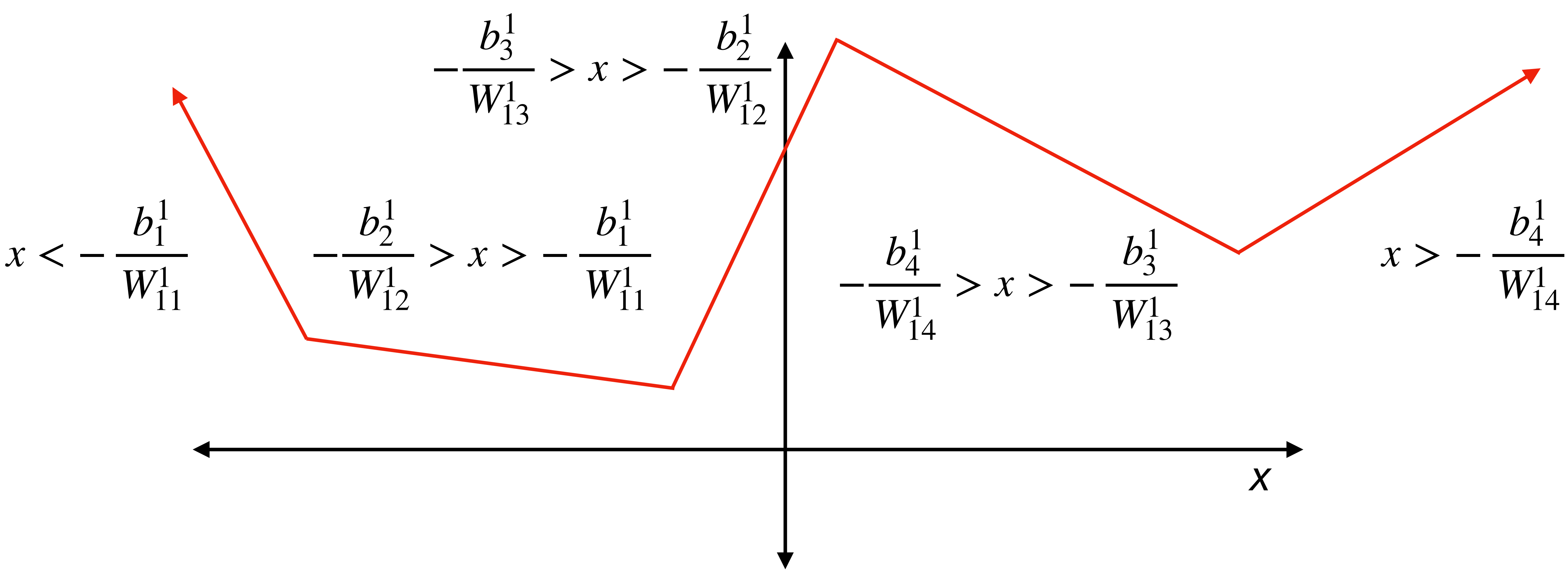
$-\frac{b_2^1}{W_{12}^1} > x > -\frac{b_1^1}{W_{11}^1}$ **Drop Three** $(W_{11}^2W_{11}^1)x + W_{11}^2b_1^1 + b_1^2$

$x < -\frac{b_1^1}{W_{11}^1}$ **Drop It** b_1^2 **Like it's Hot**

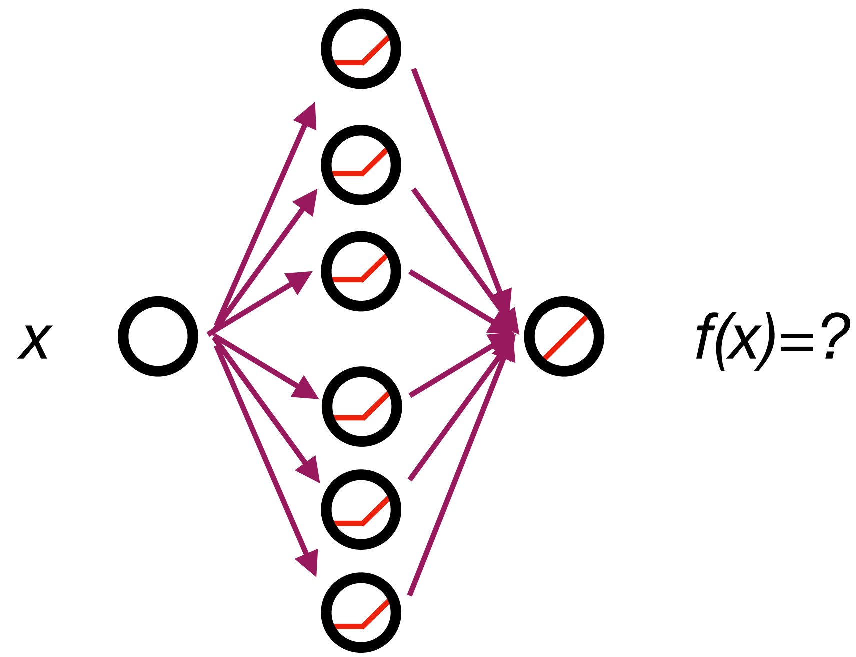
Approximating (Example)



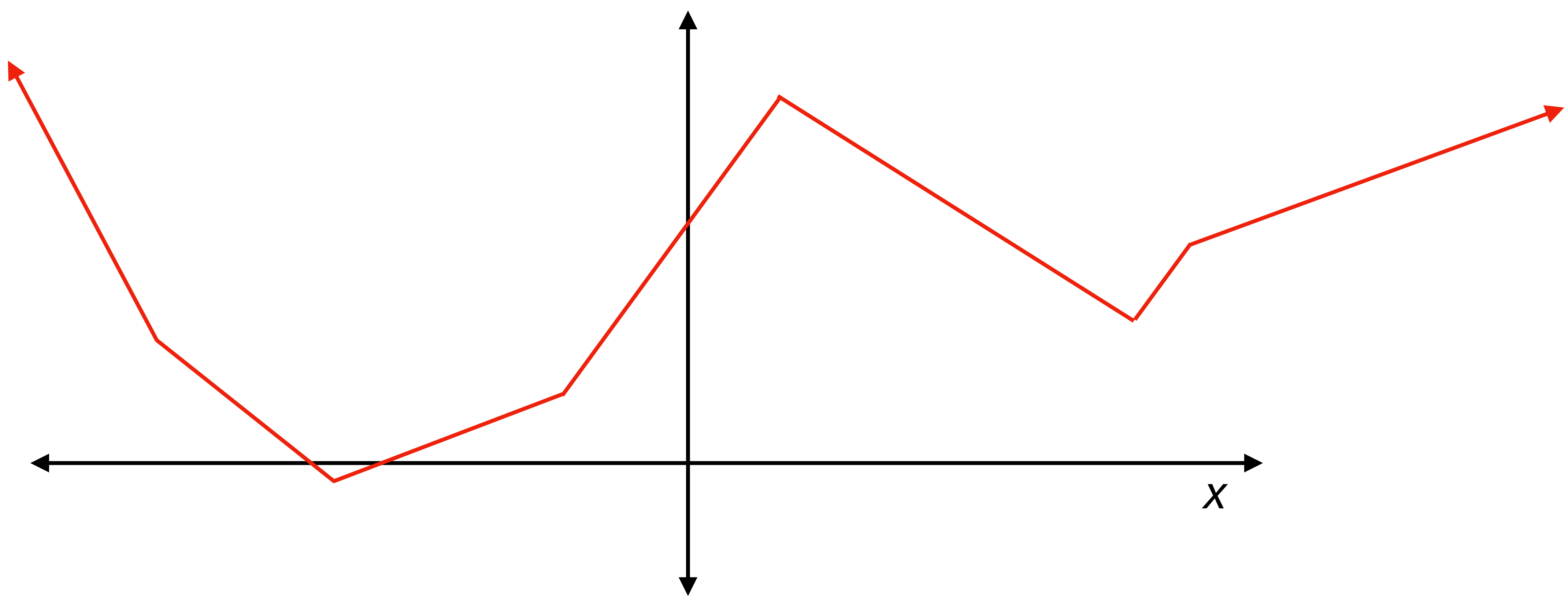
What kind of function can this network architecture approximate?



Approximating (Example)

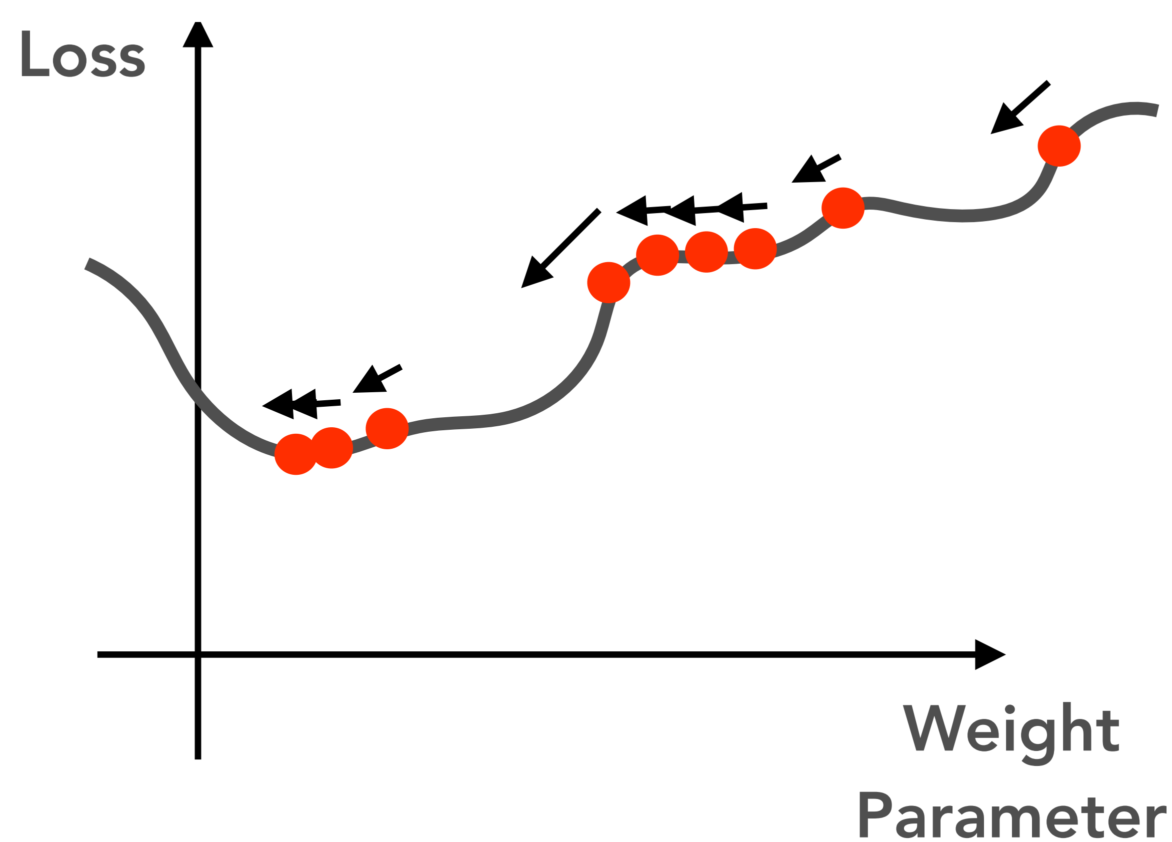


What kind of function can this network architecture approximate?

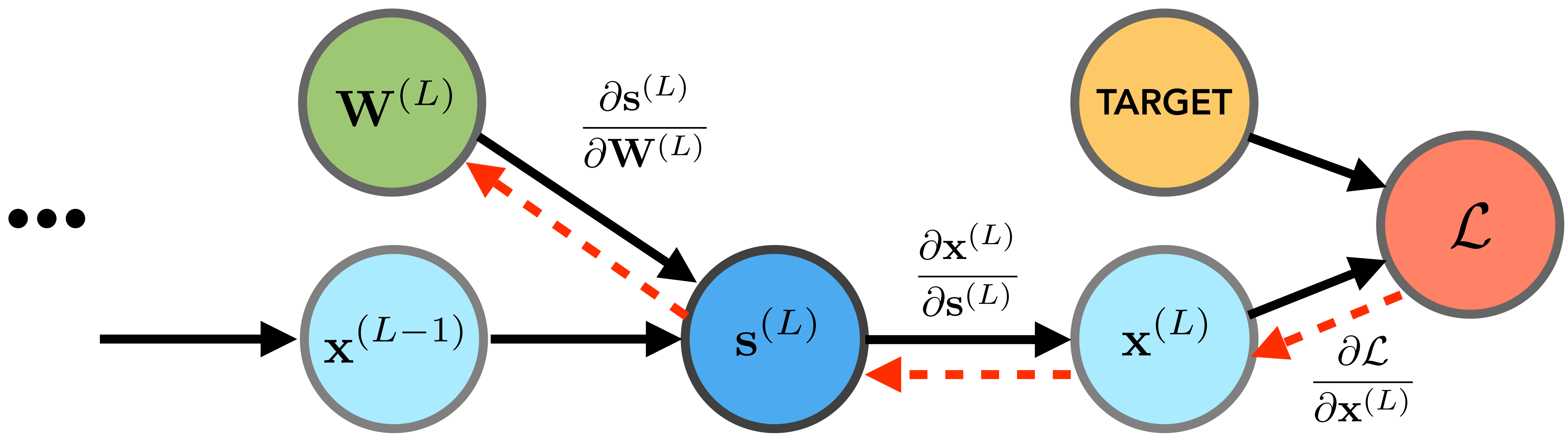


Learning - Optimization

- To learning the optimal weights we need the **derivative** of the loss w.r.t. weight
  - $w \rightarrow w - \alpha \dfrac{\partial \mathscr{L}}{\partial w}$
- With multiple weights we need the gradient of the loss w.r.t. weights
  - $\mathbf{w} \rightarrow \mathbf{w} - \alpha \nabla_{\mathbf{w}} \mathscr{L}$

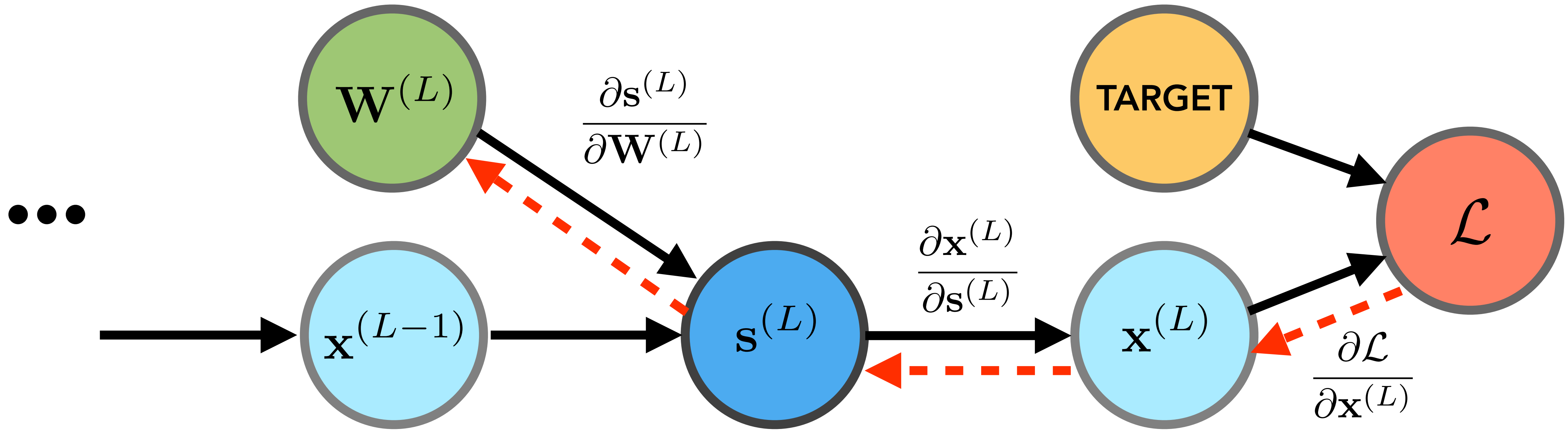


Backpropagation



- Can write a neural network as a function of composed operations
    - $\phi_L(\mathbf{w}_L, \phi_{L-1}(\mathbf{w}_{L-1}, \ldots \phi_1(\mathbf{w}_1, \mathbf{x}) \ldots))$
- The loss is a function of the network output → use chain rule to calculate gradients

Backpropagation



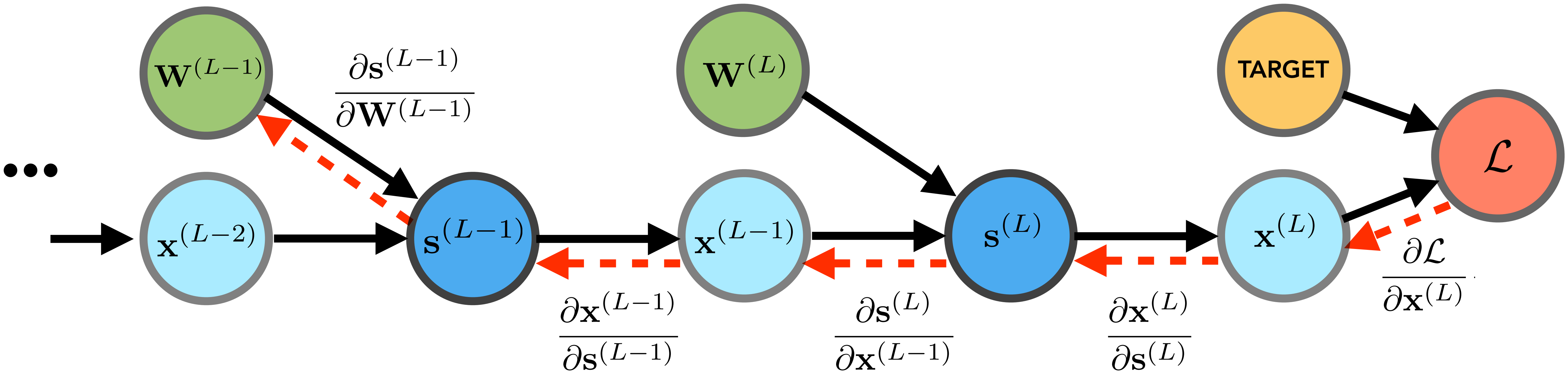
$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}^{(L)}} = \frac{\partial \mathcal{L}}{\partial \mathbf{x}^{(L)}} \frac{\partial \mathbf{x}^{(L)}}{\partial \mathbf{s}^{(L)}} \frac{\partial \mathbf{s}^{(L)}}{\partial \mathbf{W}^{(L)}}$$

depends on the form of the loss

derivative of the non-linearity

$$\frac{\partial}{\partial \mathbf{W}^{(L)}} (\mathbf{W}^{(L)\intercal} \mathbf{x}^{(L-1)})$$
$$= \mathbf{x}^{(L-1)\intercal}$$

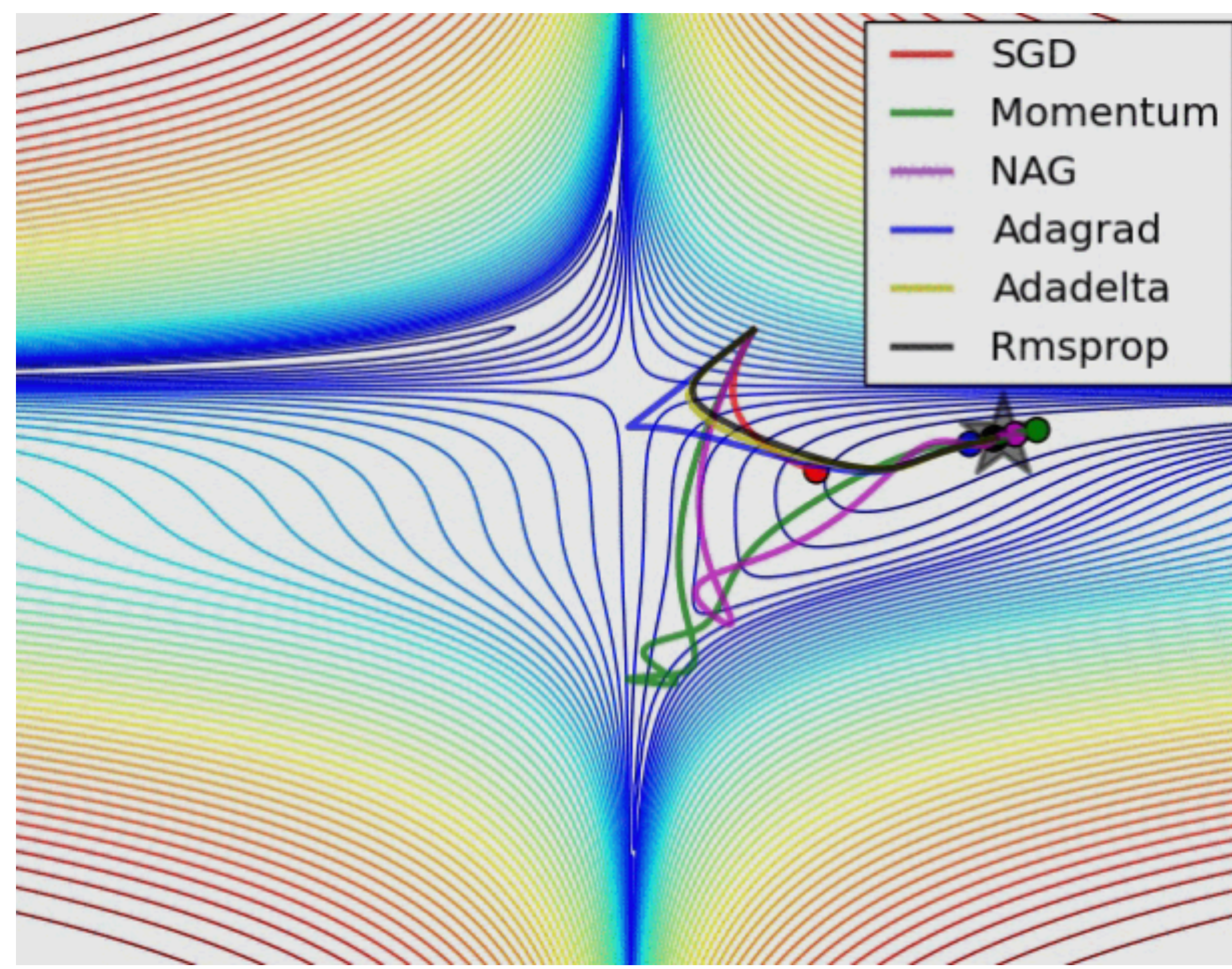
Backpropagation



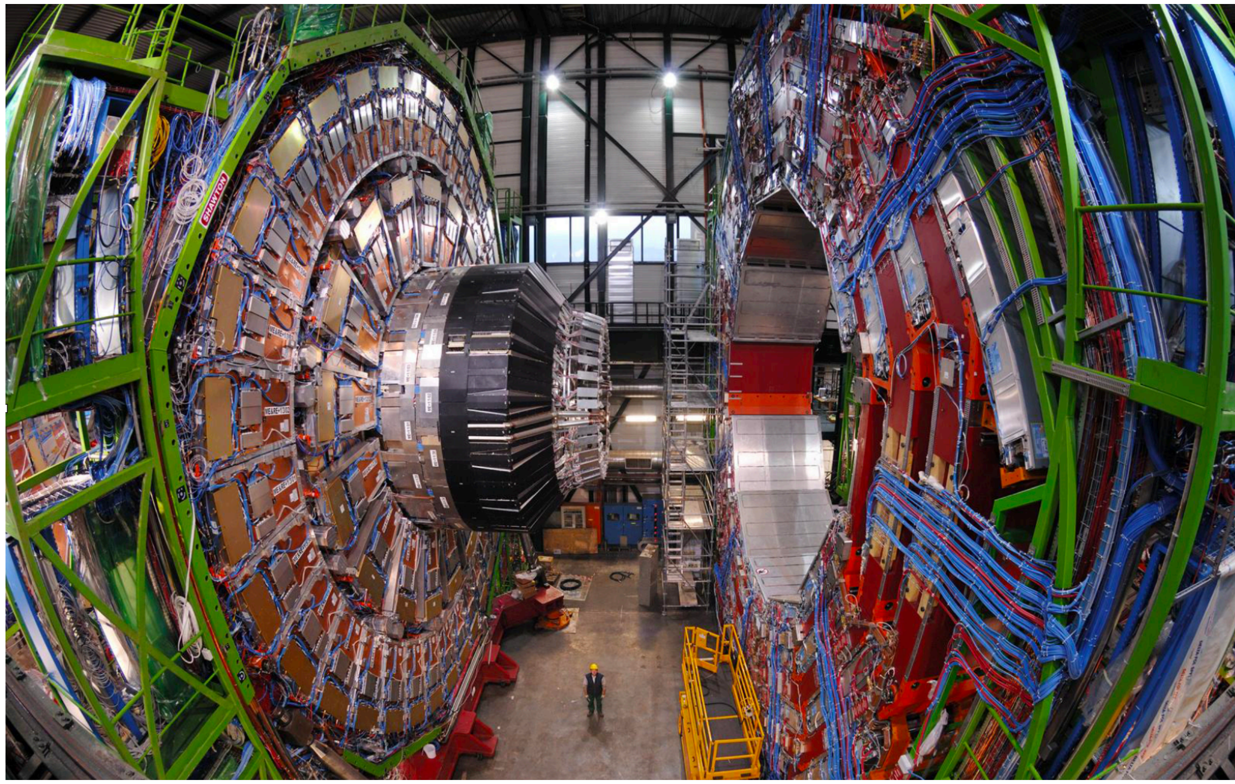
$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}^{(L)}} = \frac{\partial \mathcal{L}}{\partial \mathbf{x}^{(L)}} \frac{\partial \mathbf{x}^{(L)}}{\partial \mathbf{s}^{(L)}} \frac{\partial \mathbf{s}^{(L)}}{\partial \mathbf{x}^{(L-1)}} \frac{\partial \mathbf{x}^{(L-1)}}{\partial \mathbf{s}^{(L-1)}} \frac{\partial \mathbf{s}^{(L-1)}}{\partial \mathbf{W}^{(L-1)}}$$

Stochastic Gradient Descent

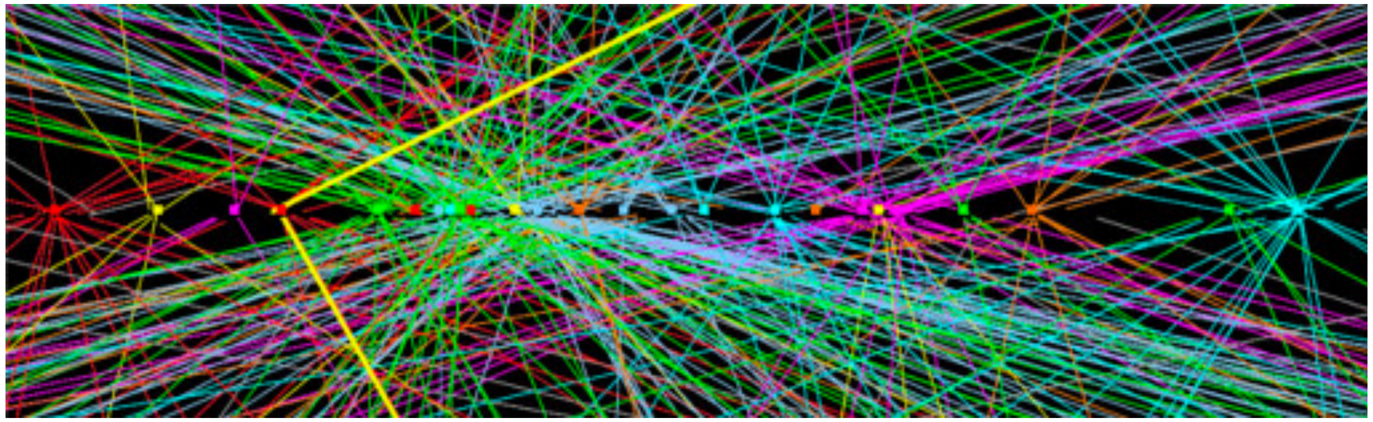
- SGD: $w = w - \alpha \tilde{\nabla}_w \mathscr{L}$
  - Use estimate to traverse the loss function
- Advanced estimates use "memory", other optimizations
  - Able to handle large dimensionality, complex surfaces (saddle points, local minima)



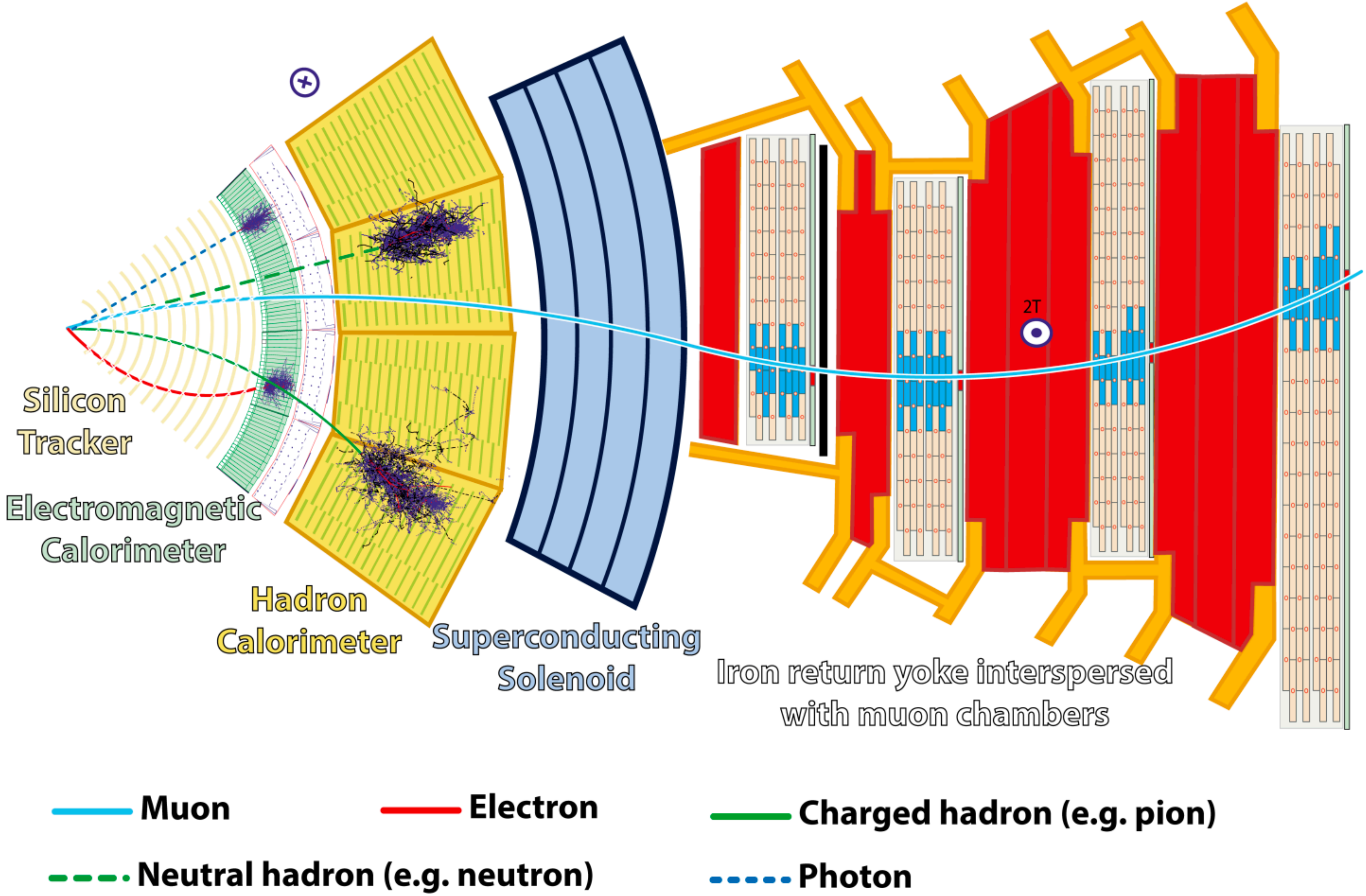
Large Hadron Collider



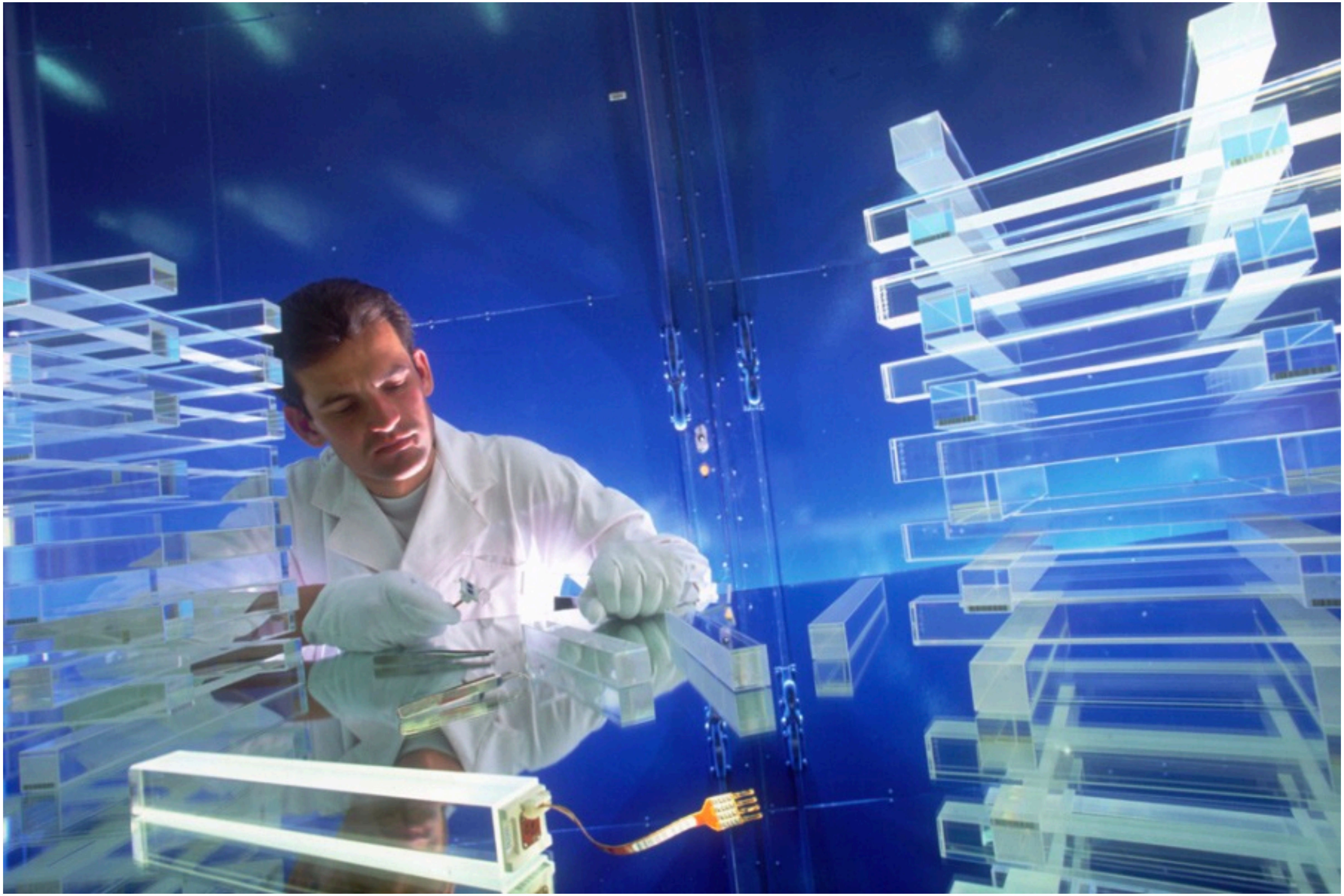
- Large Hadron Collider (proton - proton)
- Dedicated detectors to record data from collisions
  - Electromagnetic calorimeter (ECAL) is designed to measure energy of EM particles, **e + γ (EG)**
- Many overlapping collisions in addition to the primary one, called **pileup (PU)**
- Want to design algorithm to distinguish primary **EG** energy deposits from **PU**

p →  ← ***p***

CMS Detector

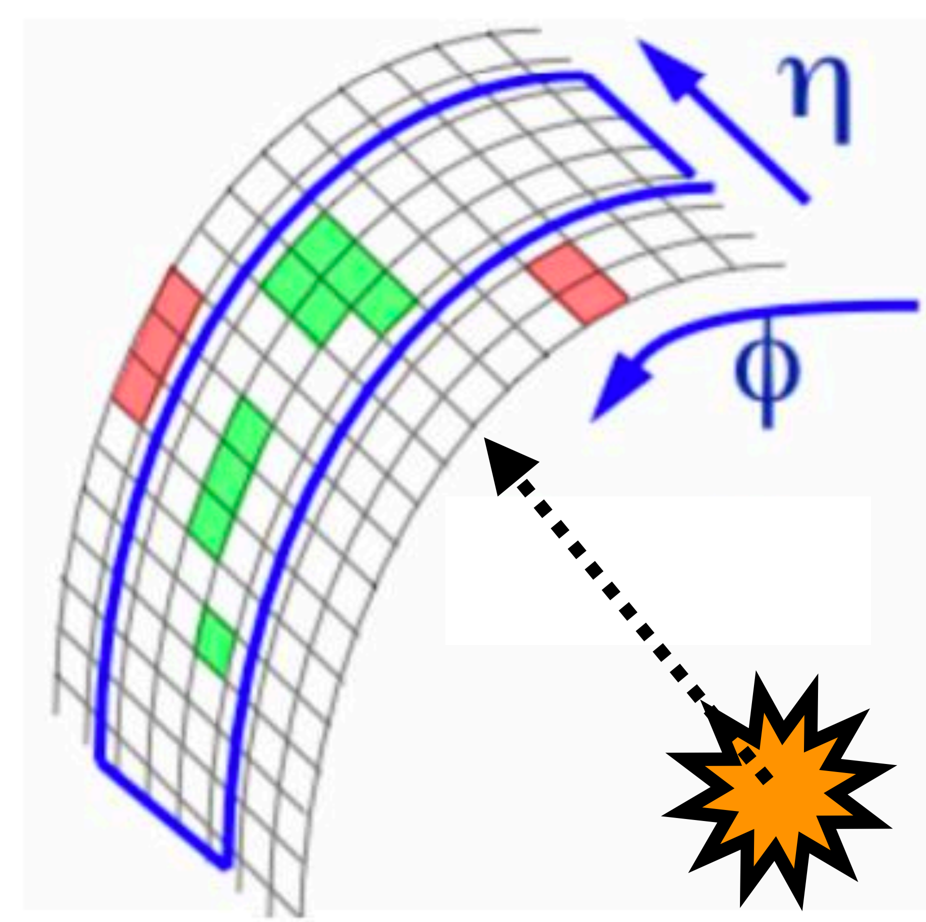


CMS ECAL

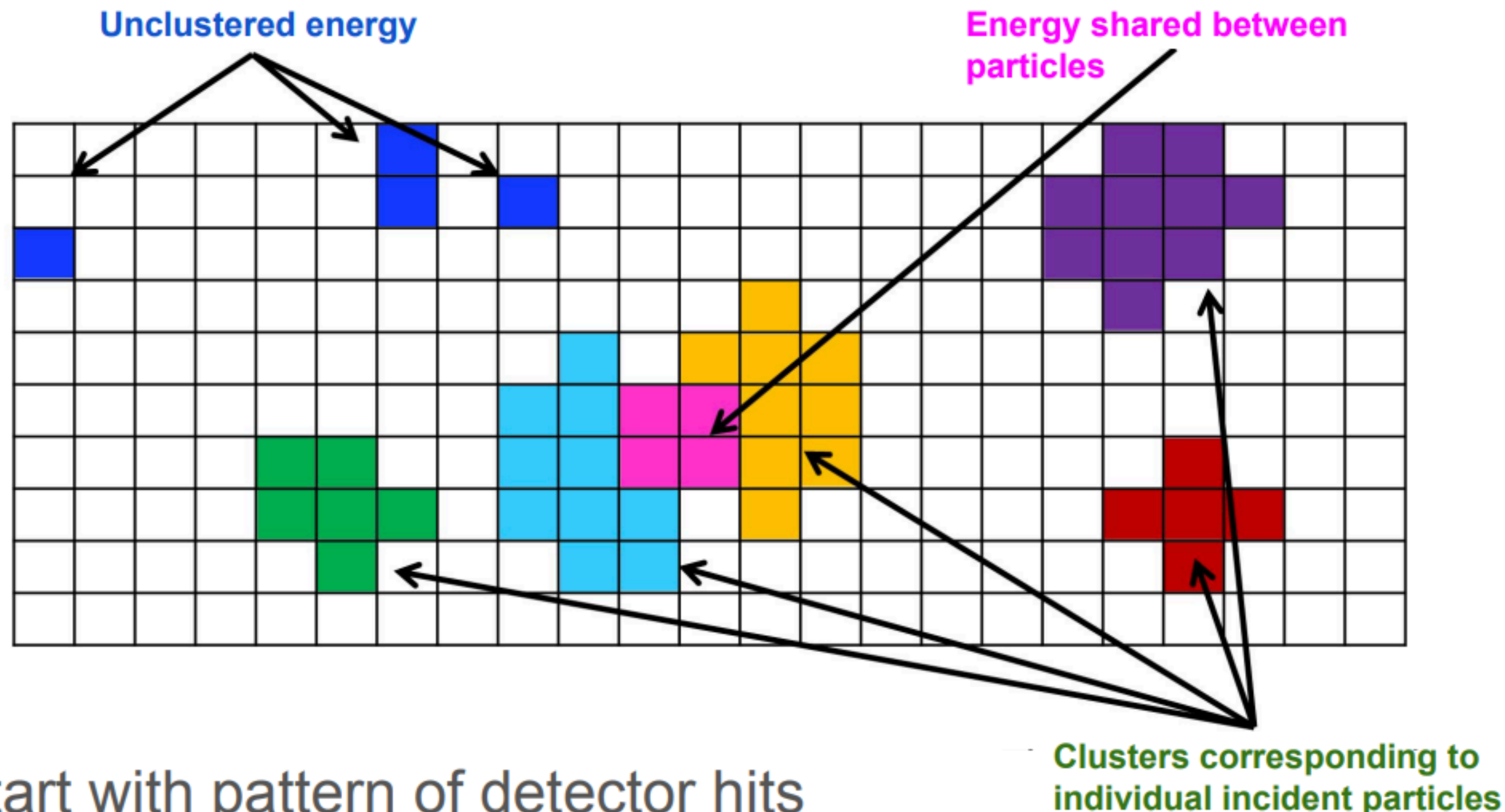


Dataset Explanation

- Real photon and electrons will deposit energy in certain patterns
    - PU is either random or from other sources, should not match
- We describe energy deposits using "shower shape variables"
    - i.e. How much energy is in certain regions around the center? How correlated are deposits in η and ϕ? How many crystals in particular directions?

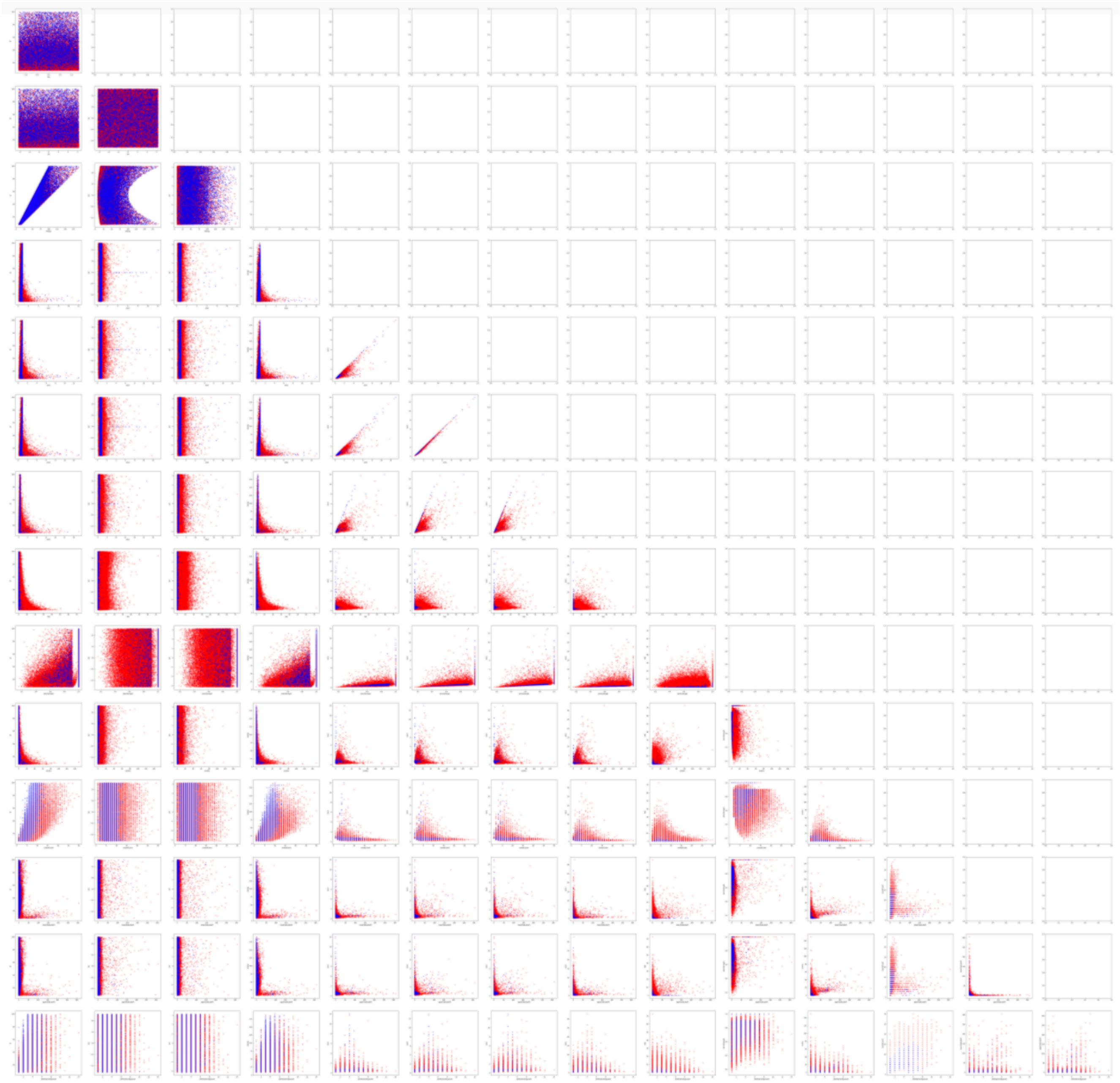


Dataset Visualization



- Start with pattern of detector hits
- Two main tasks:
    - Associate each incident particle with a collection of hits

Designing ML



How can I predict if a point is red or blue given x_1 and x_2 and x_3 and …?

(Lets use the notebook)

Batch Norm & Dropout

- In addition we often employ these two to improve perf

Batch Norm makes output Gaussian-like

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\ldots m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

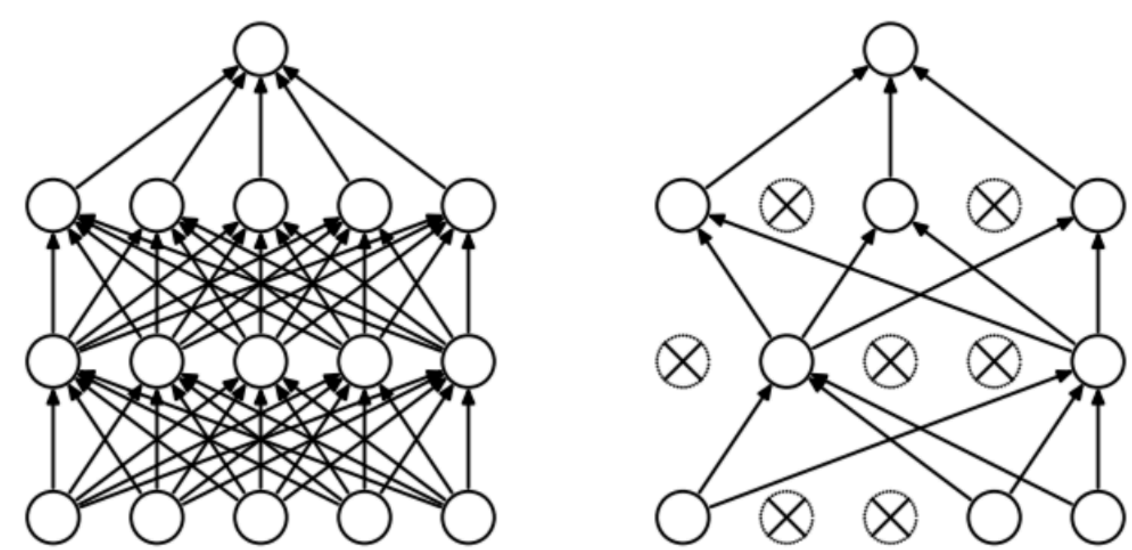
$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Dropout randomly removes weights



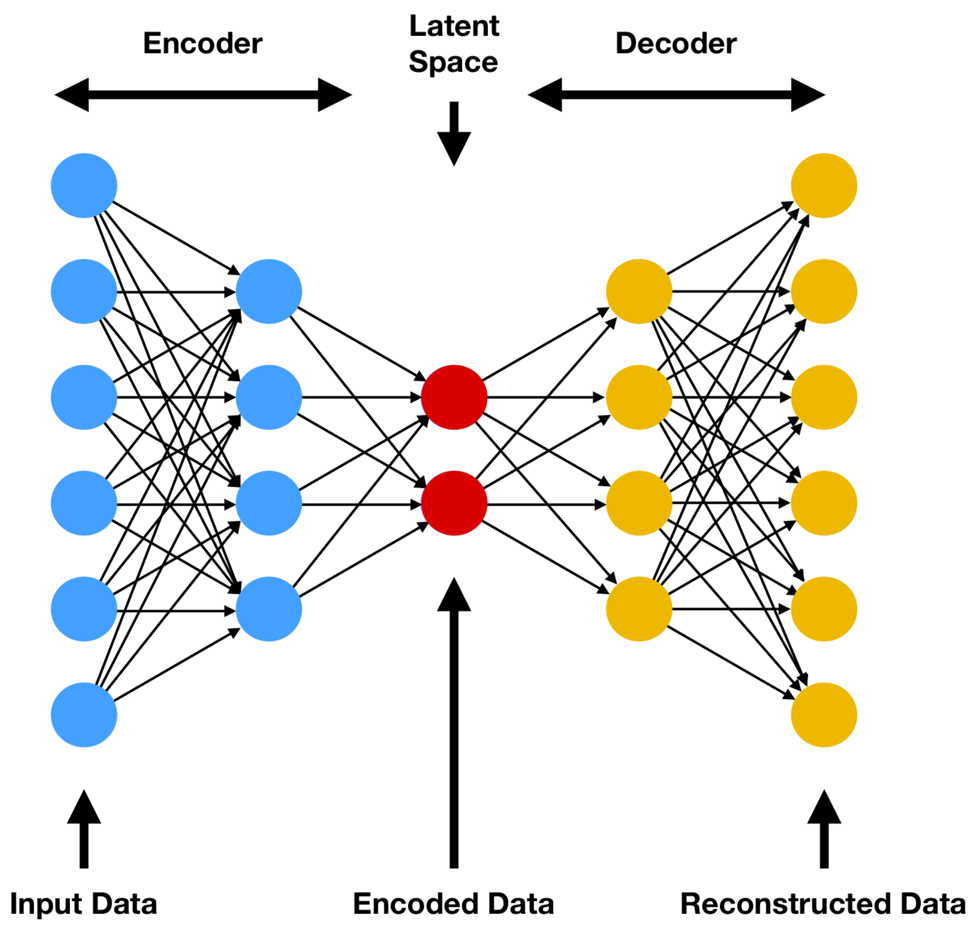
(a) Standard Neural Net (b) After applying dropout.

- Both Strategies make NNs more robust to deviations

More Complex Architectures

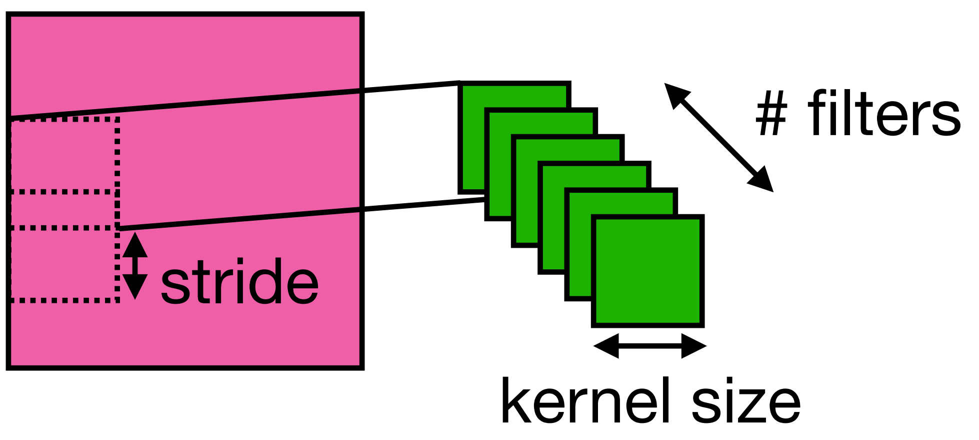
Unsupervised Learning

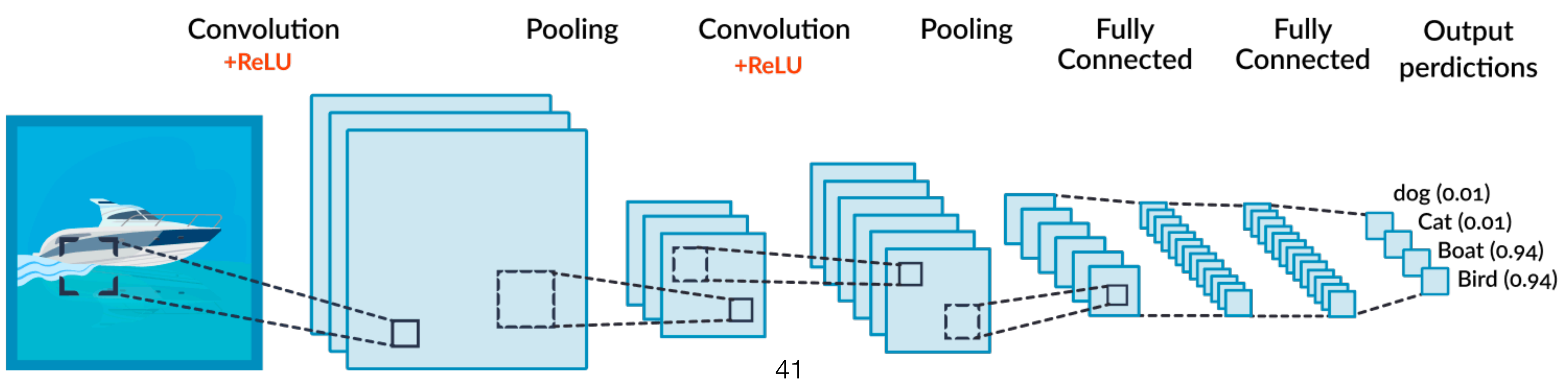
- What if we don't have/use labels?
- Autoencoder (AE):
    - *Loss = output - input*
    - Latent space can be used for clustering
    - If one class of data is used to train, different classes may not reconstruct well (anomaly detection)



Convolutional Neural Network (CNN)

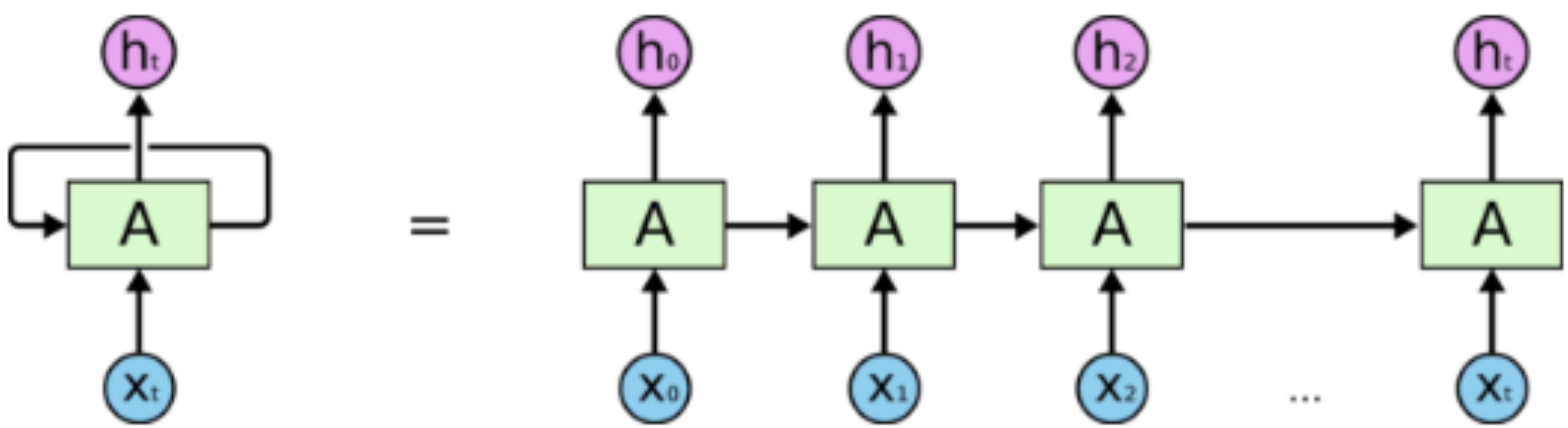
- Used extensively for images (Conv2D), also useful for 1D and 3D cases
- Small dense network takes a local region as input, scans over whole image
    - *# filters, kernel size, stride*
    - Typically followed by Pooling layer to reduce dimensionality





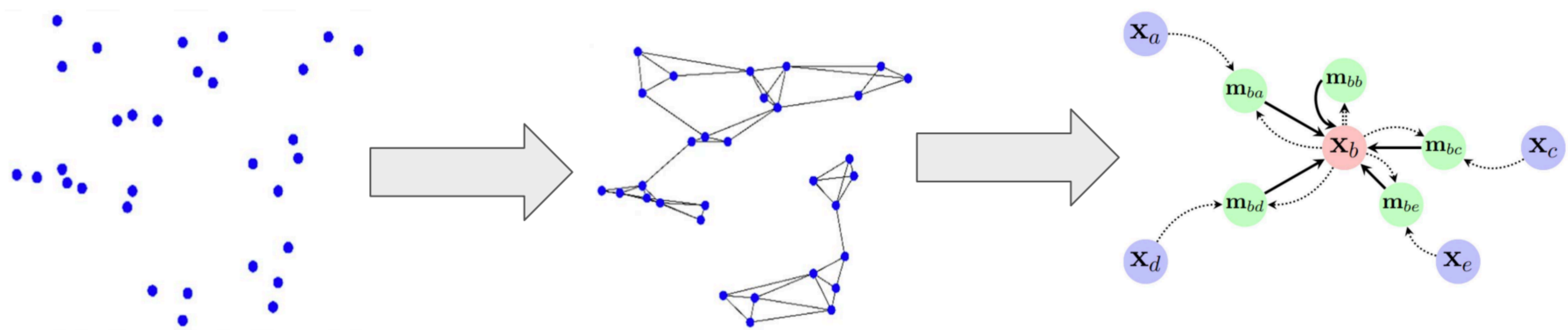
Recurrent Neural Network (RNN)

- Designed for sequential inputs (ex. language)
  - Retain "memory"
- Long short-term memory (LSTM), gated recurrent unit (GRU)



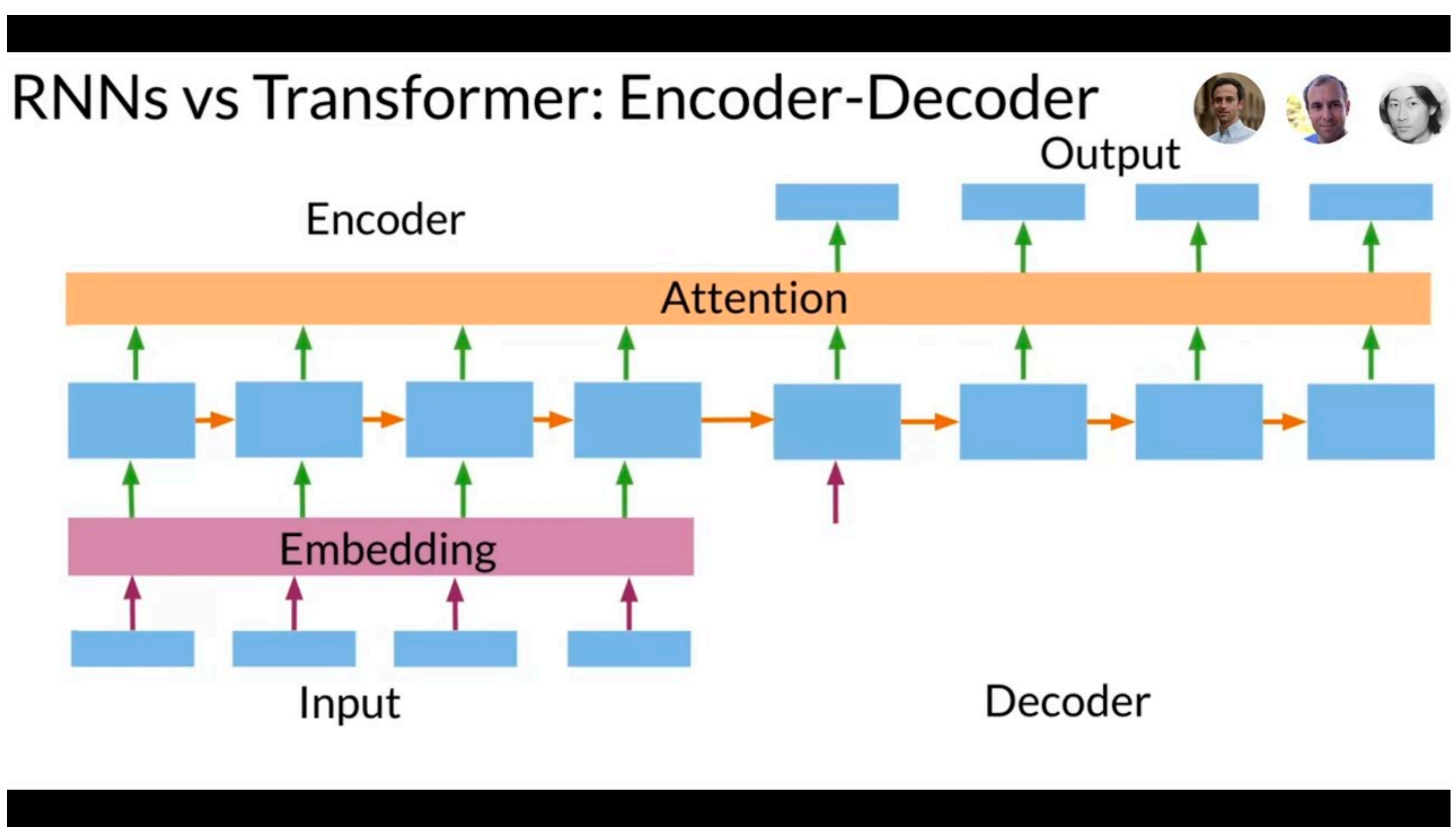
Graph Neural Network (GNN)

- Take in a set of points
  - Construct a graph out of these points
    - Make links between the neighbors
  - Iterate back and forth with the neighbors



Transformers

- Originated from recursive neural networks
  - Reading a whole sentence is better than iteratively
    - Done by applying Attention (effectively a big linear layer)
  - Basically a Graph with a notion of ordering



Next Lecture

- We are going to look at another application of ML
  - Using ML for regressions
  - How can we solve for complex physical scenarios