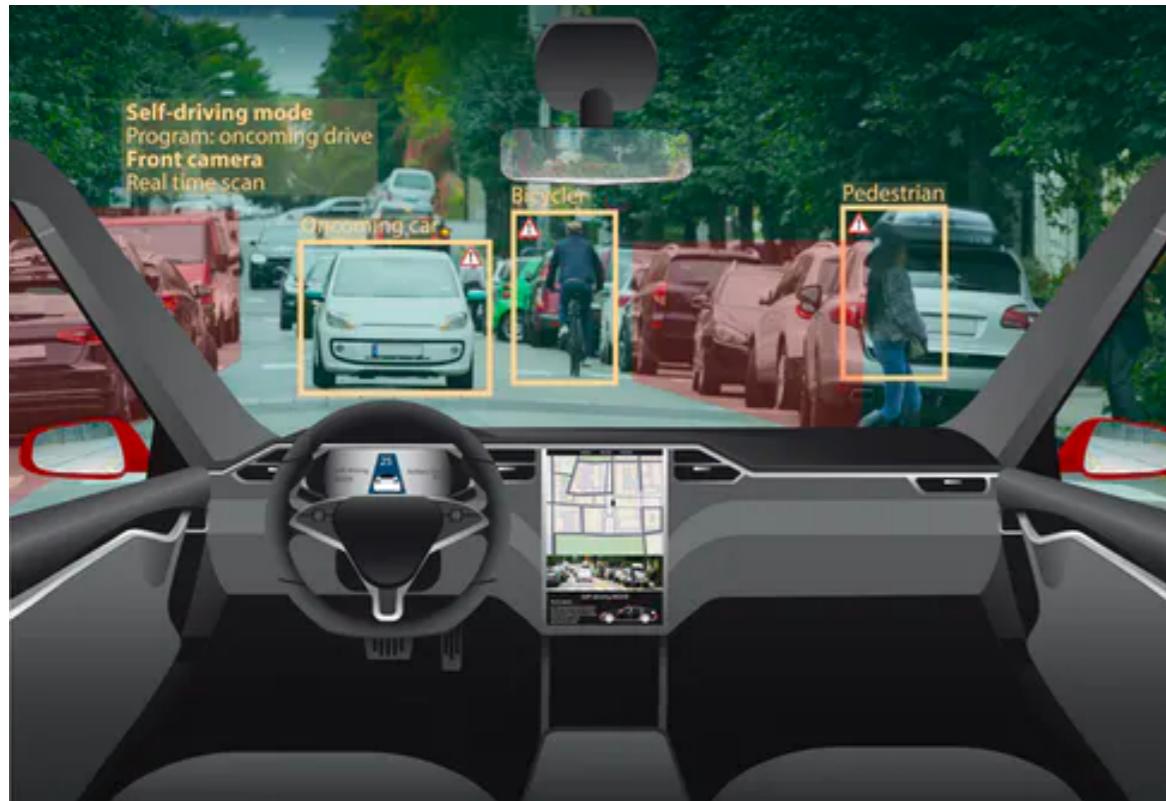




Deep Learning

Lecture 12



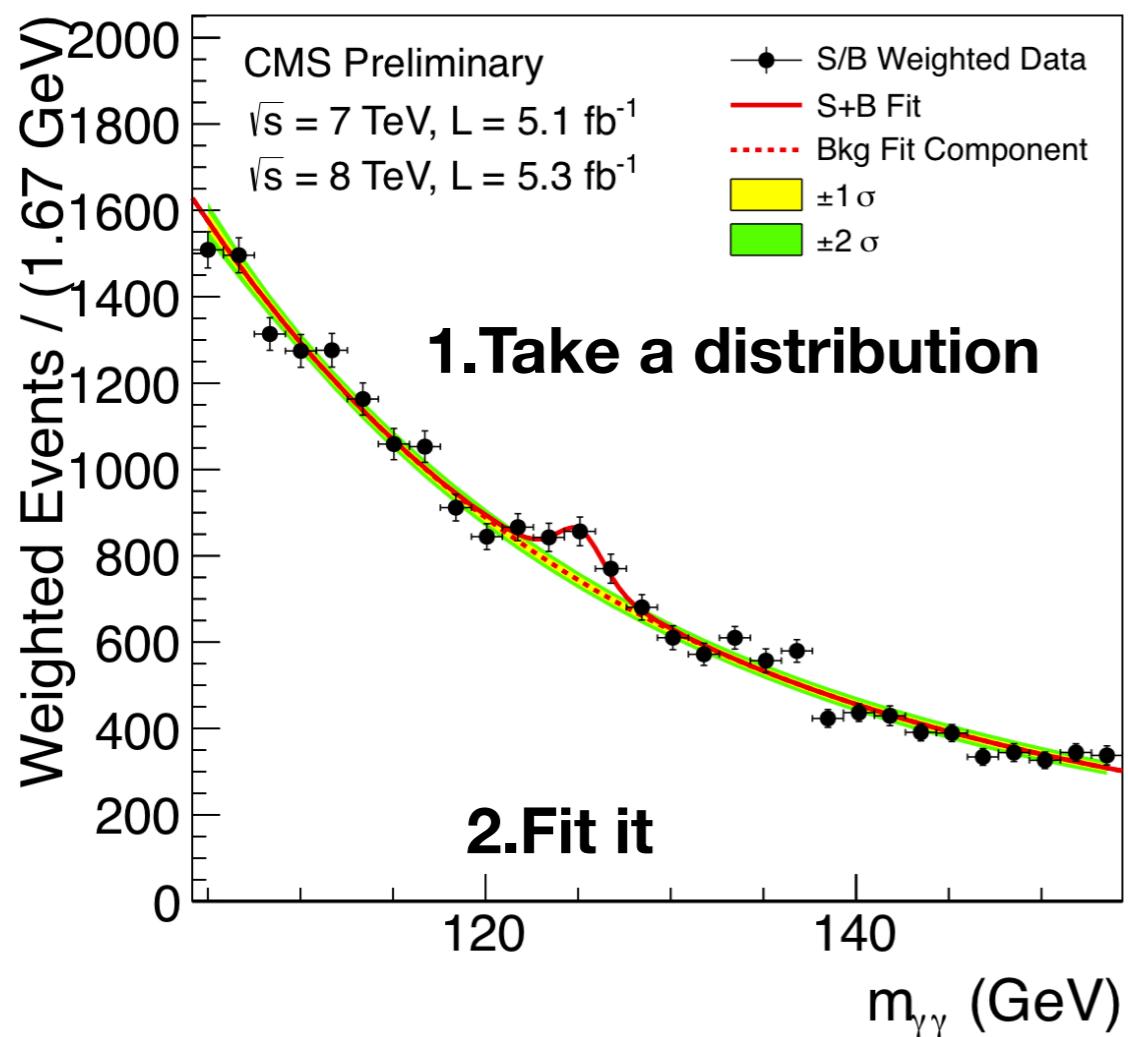
Phil Harris
Stolen from Dylan Rankin

Key Terms

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

Origins of Machine Learning

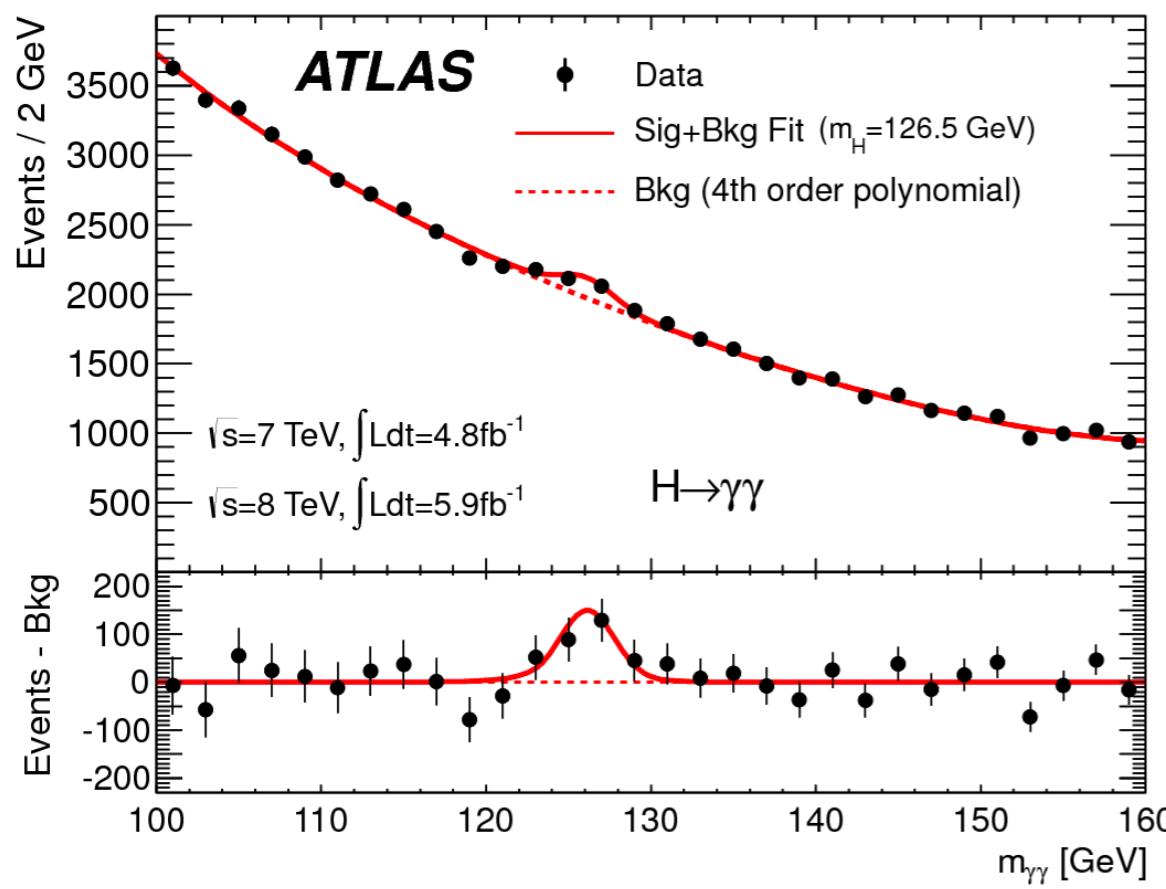
- I assume you are all familiar with fitting



3. Change physics forever

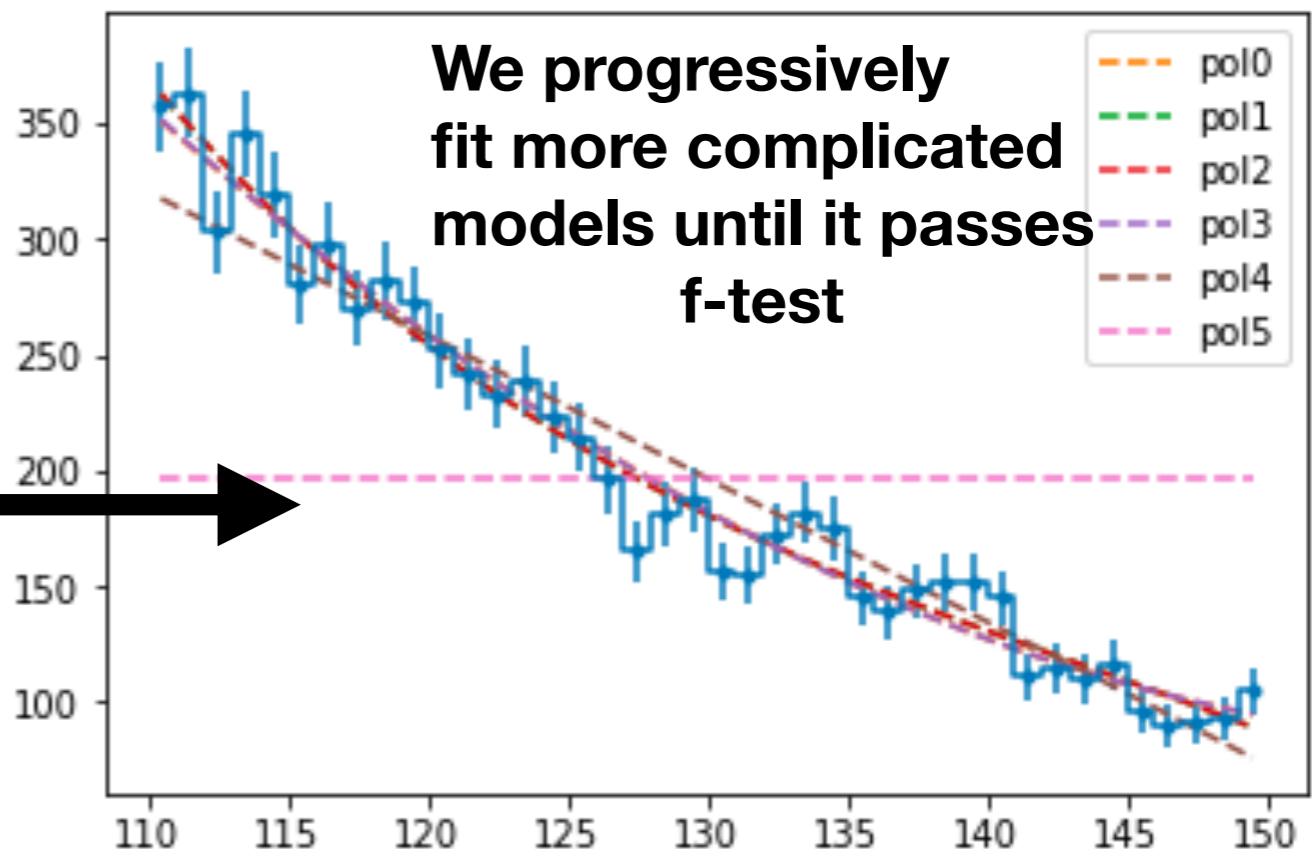
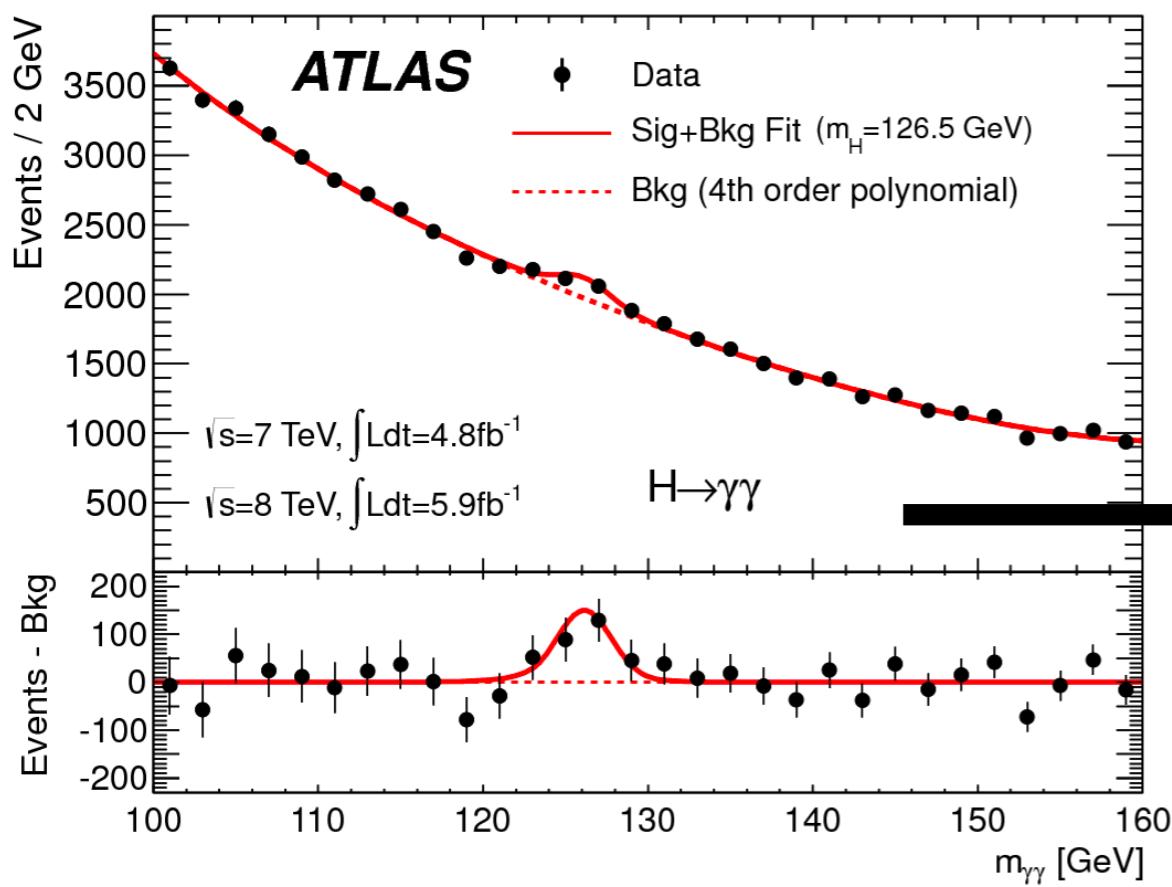
What is the right fit function?

- 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 background model?



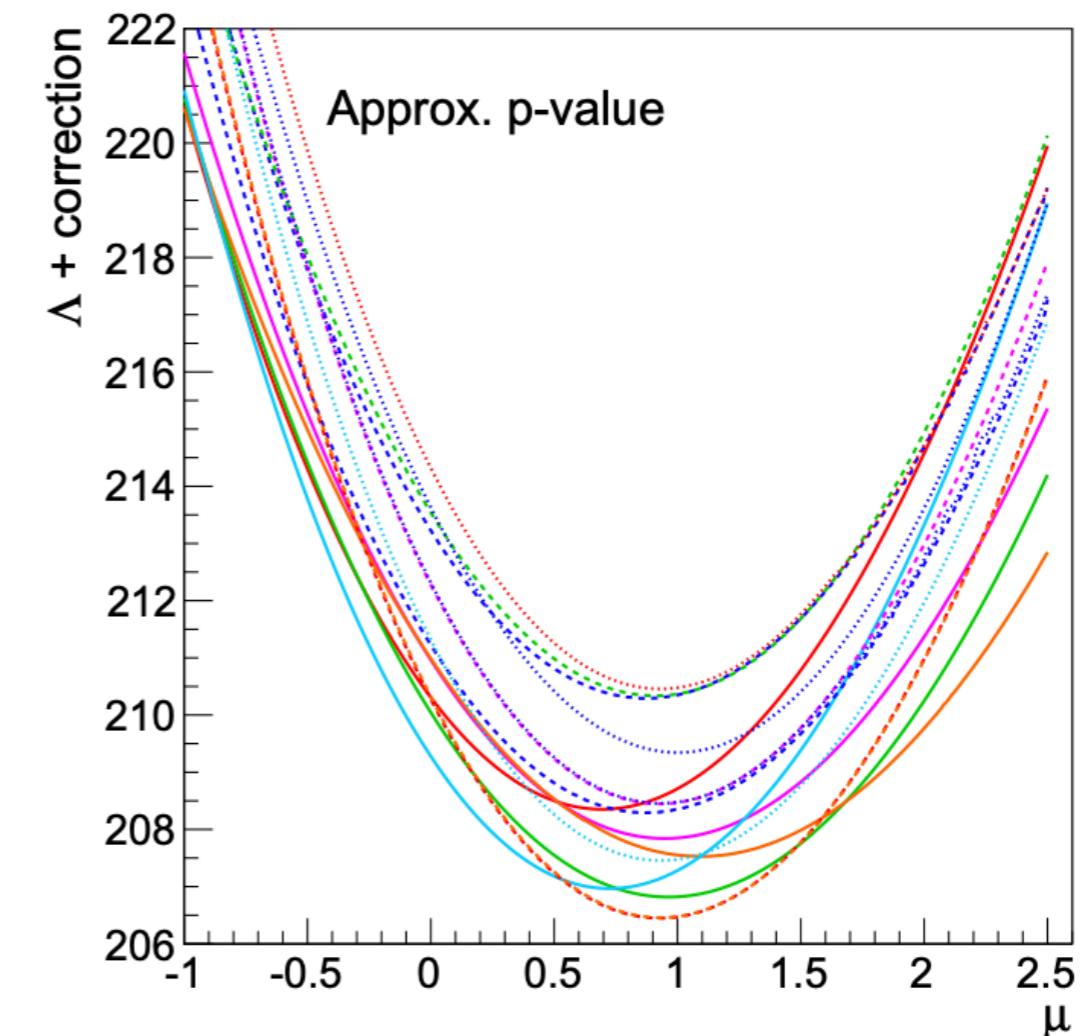
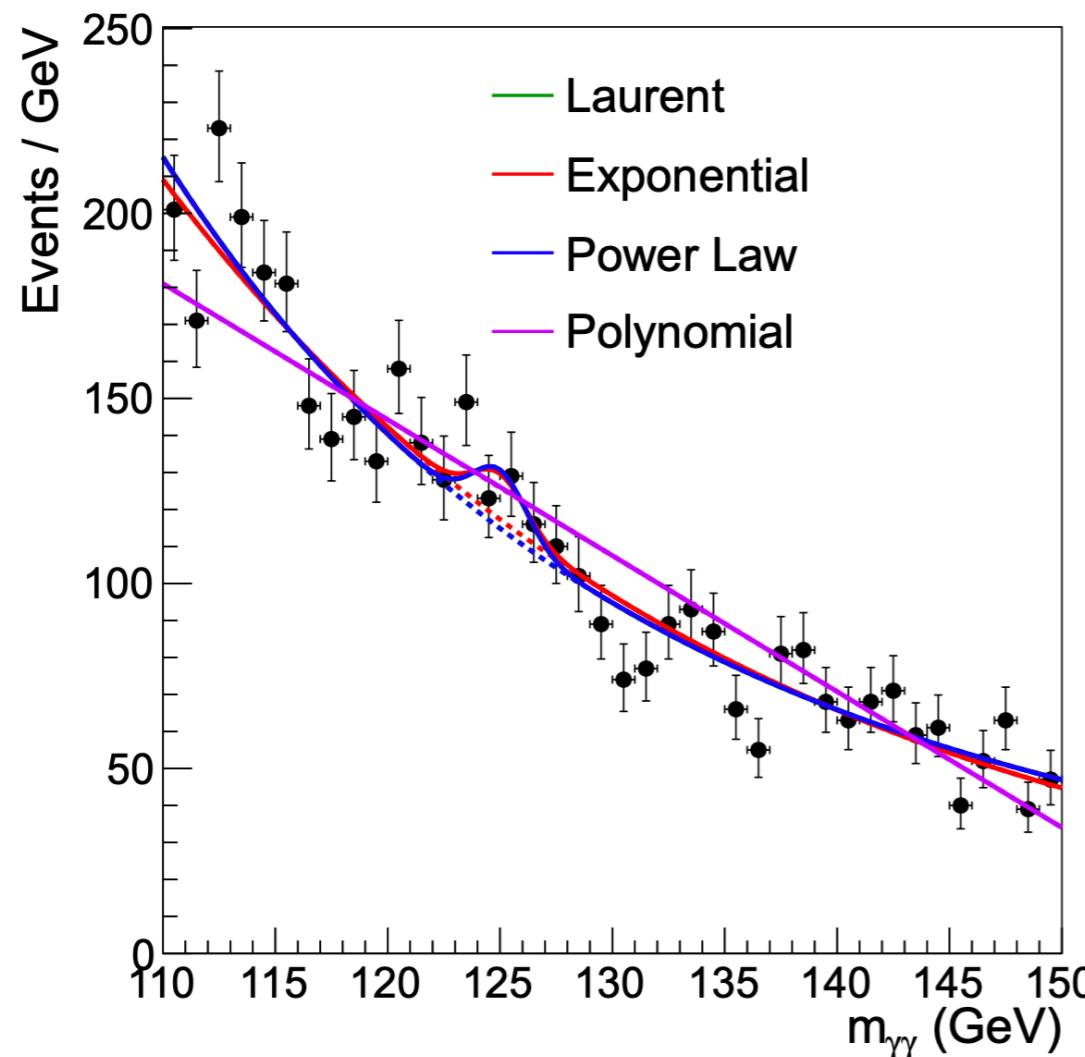
What is the right fit function?

- 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 background model?



Building a Model

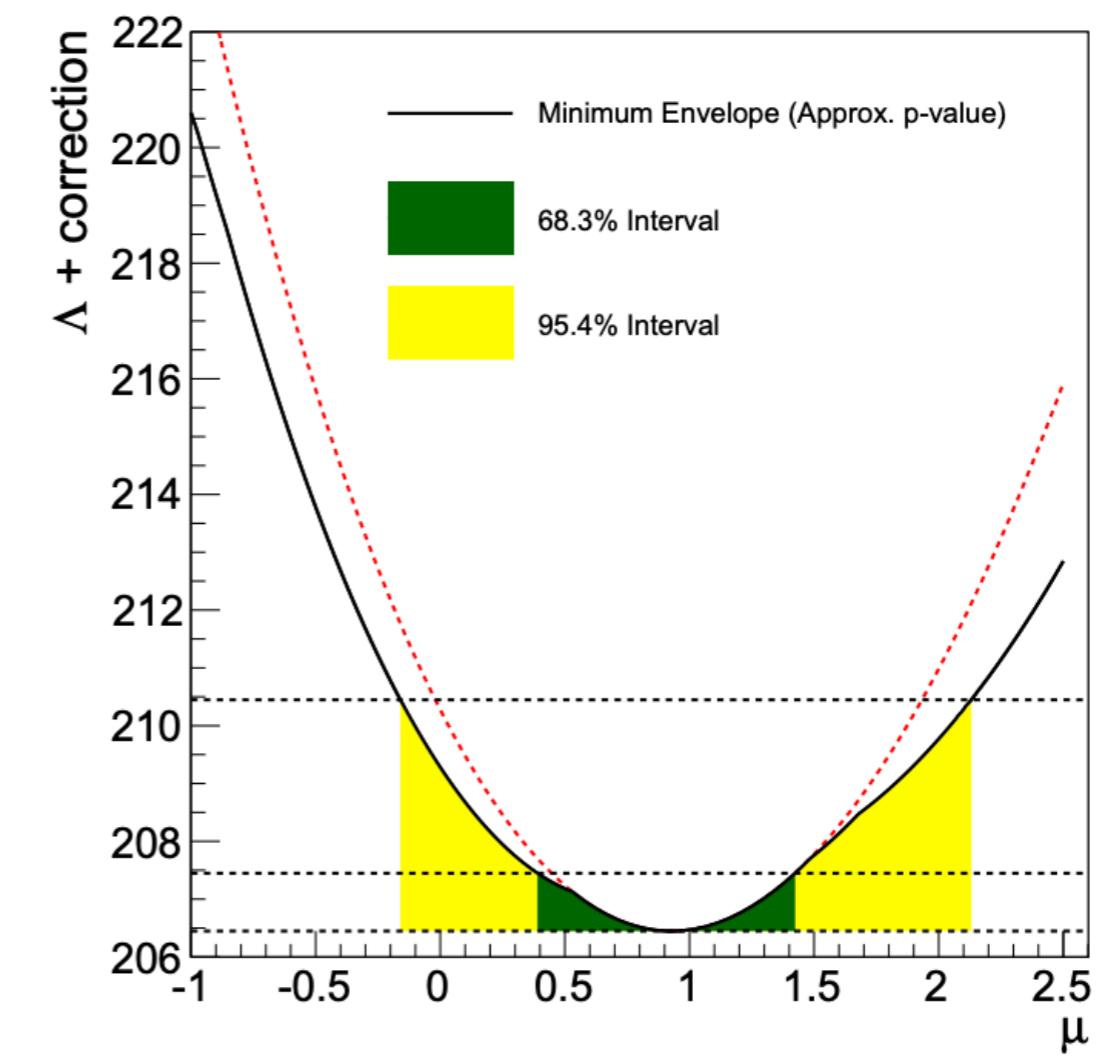
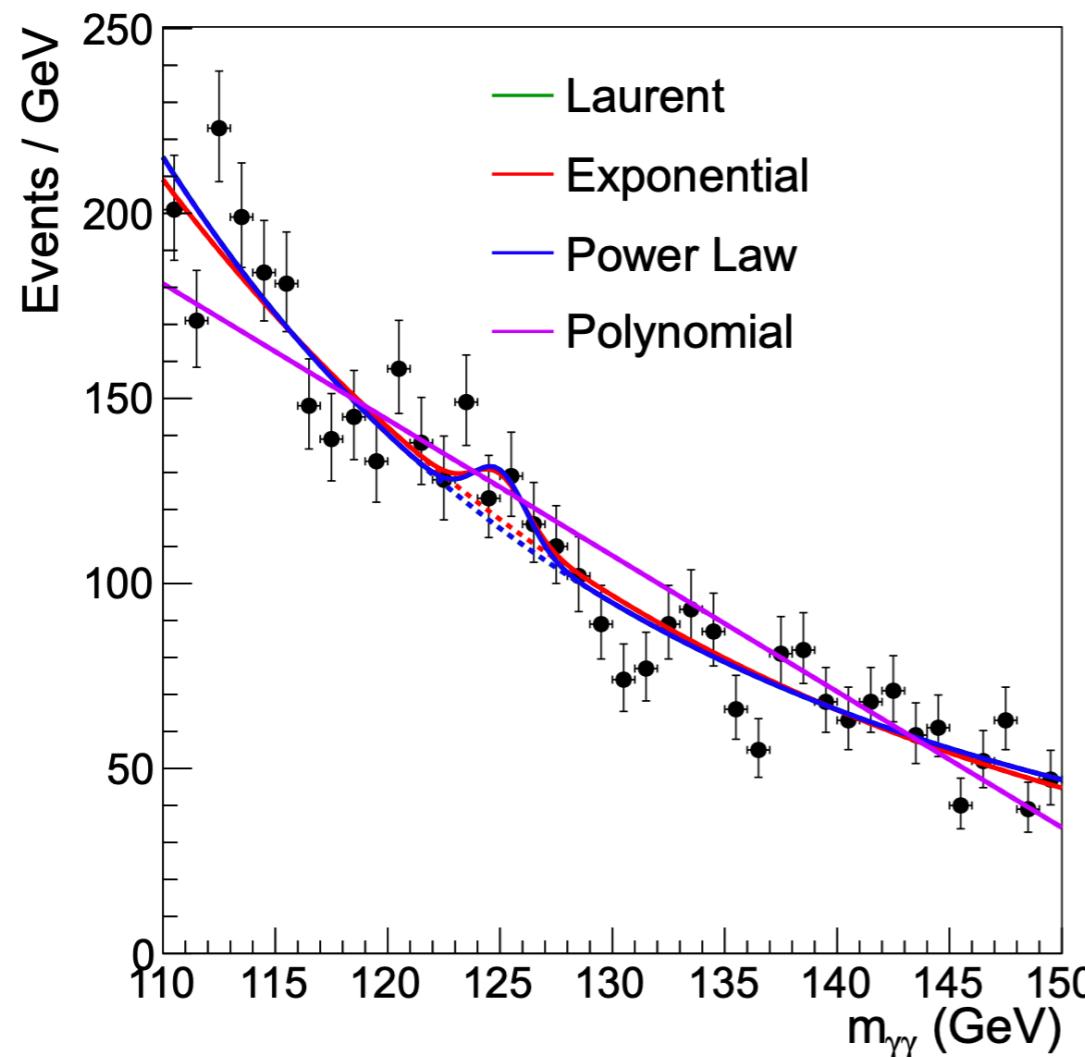
- Throwing a barrage of functions at the problem



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

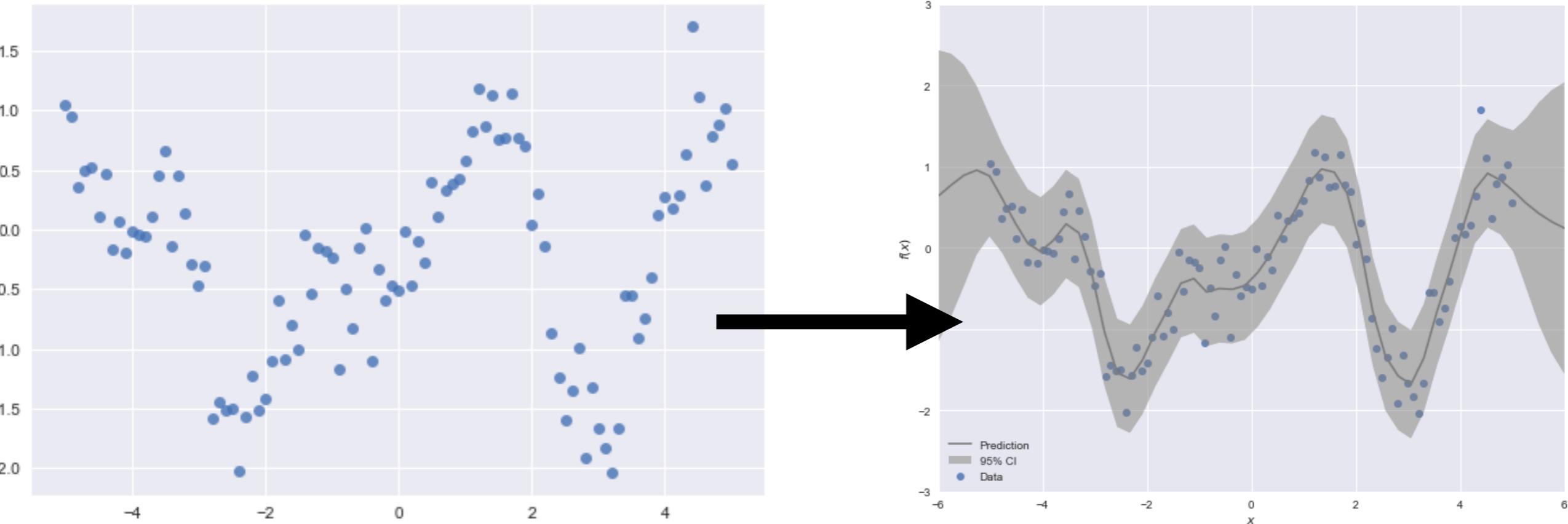
Building a Model

- Throwing a barrage of functions at the problem



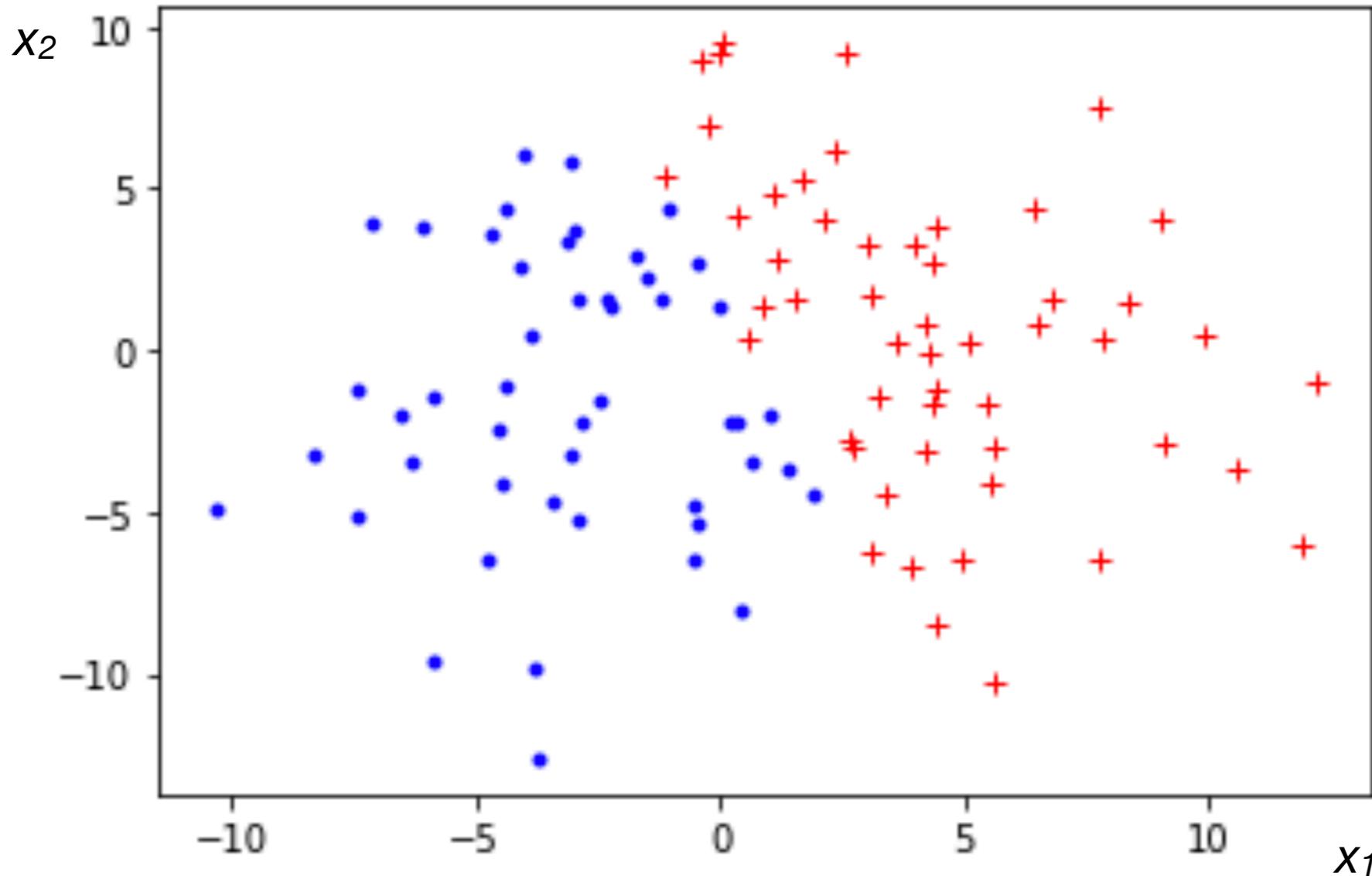
We can try a whole library of functions
The likelihood we get translates to our fit

Splines+GaussianProcesses



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

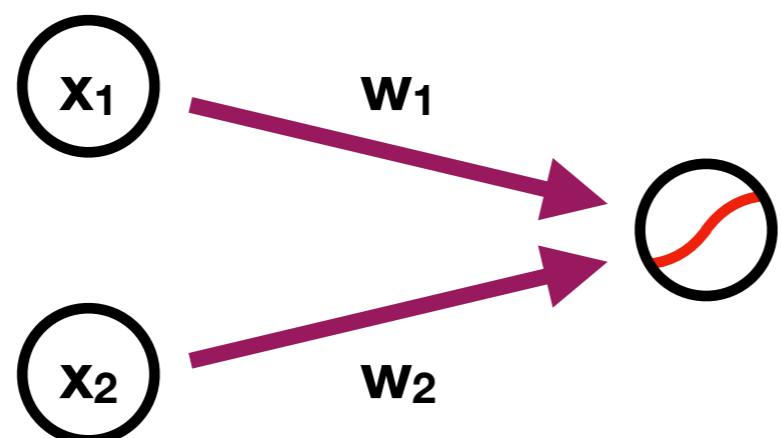
Simple Machine Learning



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.

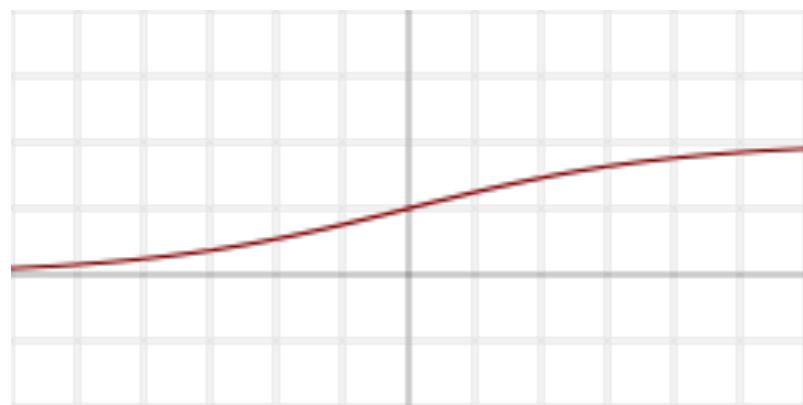


$$\ell = \frac{1}{1 + e^{\mathbf{w}^T \mathbf{x} + b}}$$

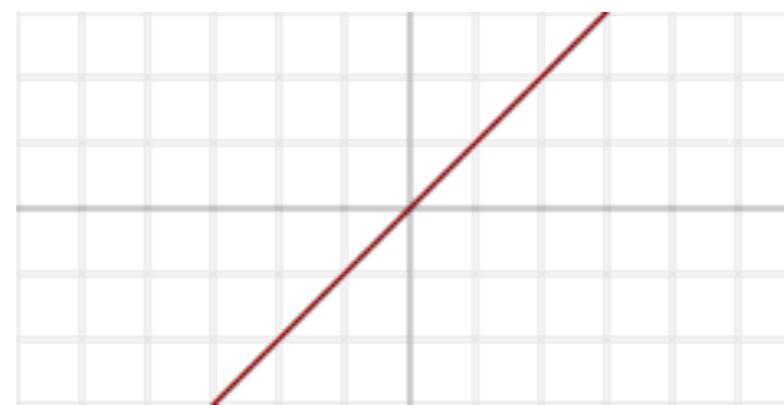
$$\mathbf{w}^T = (w_1 \ w_2)$$

Activation Functions

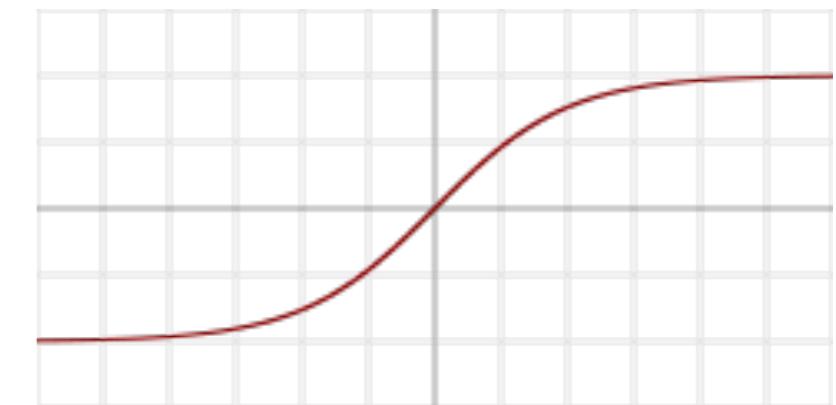
Sigmoid



Linear



Tanh

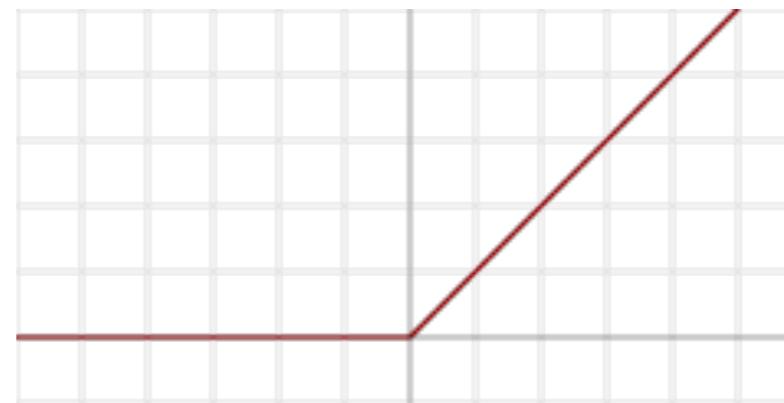


Softmax

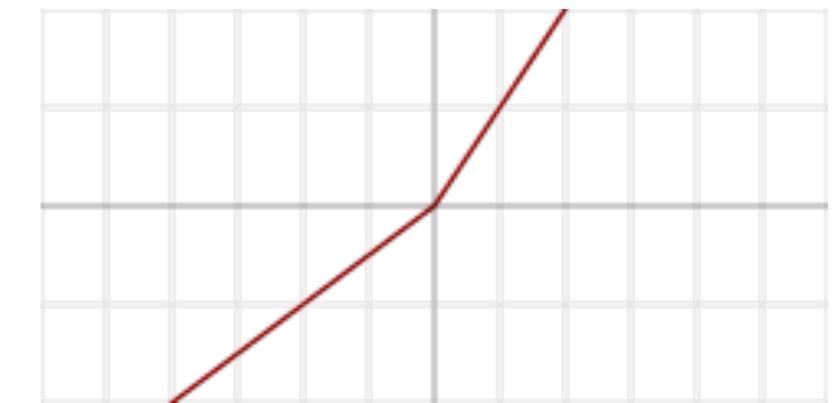
(multiclass)

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

ReLU

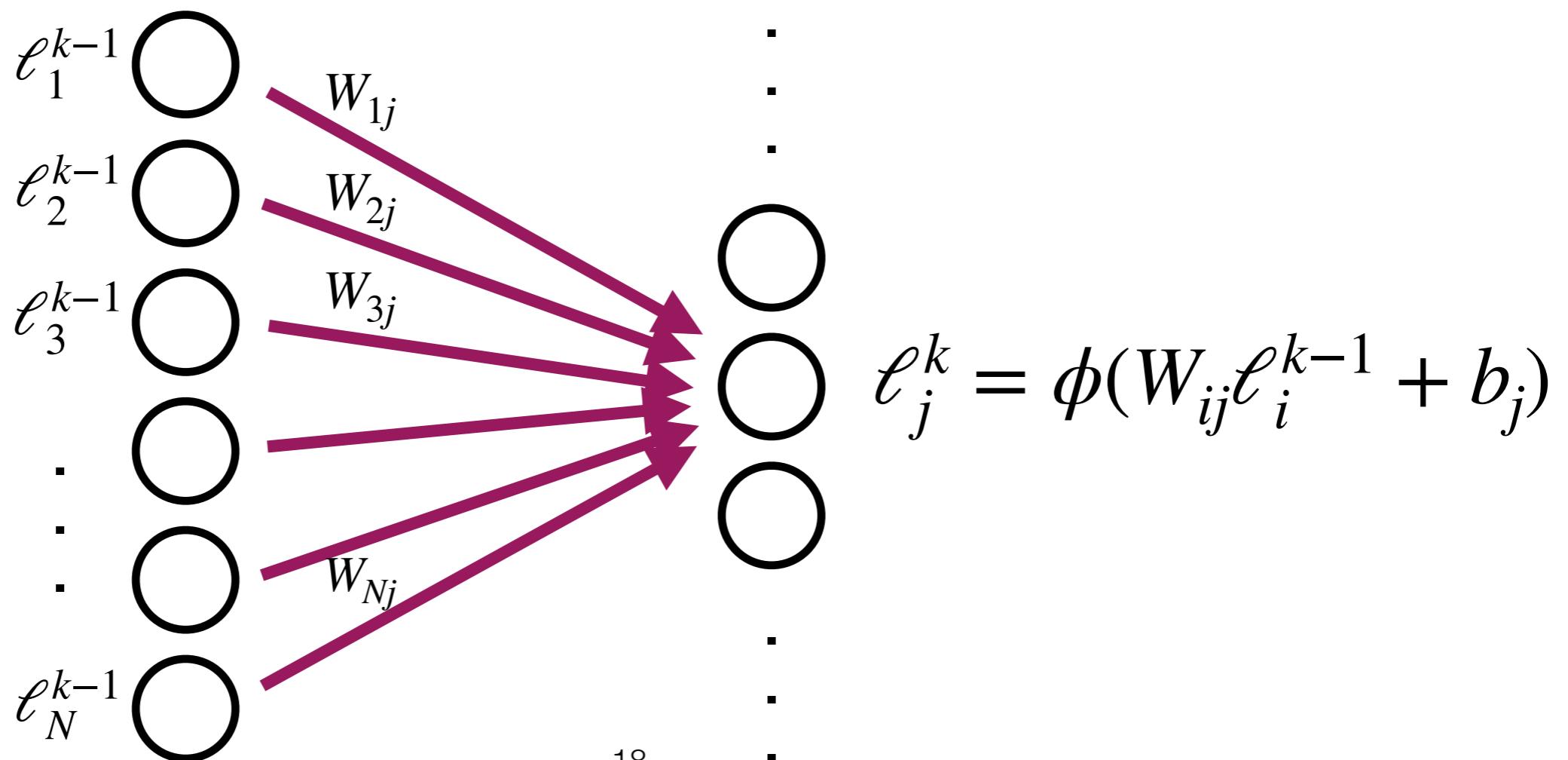


LeakyReLU



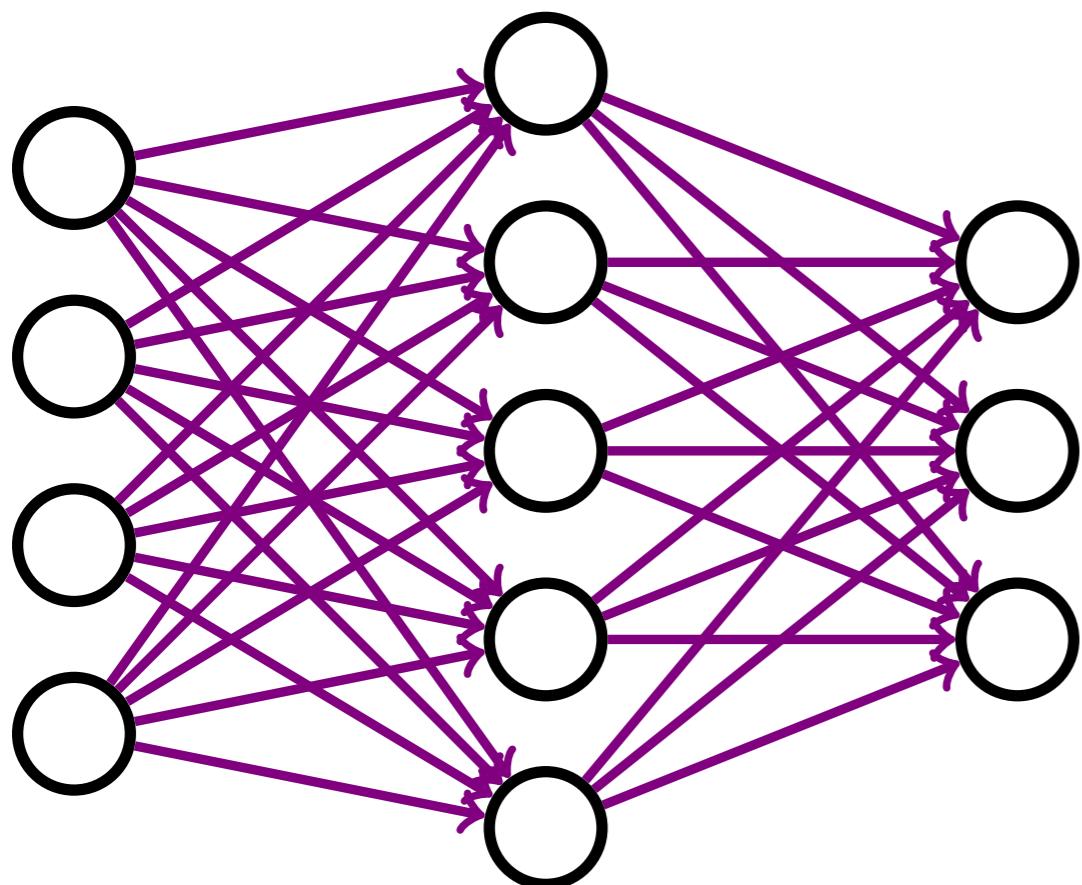
Neural Network

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



Dense Neural Network

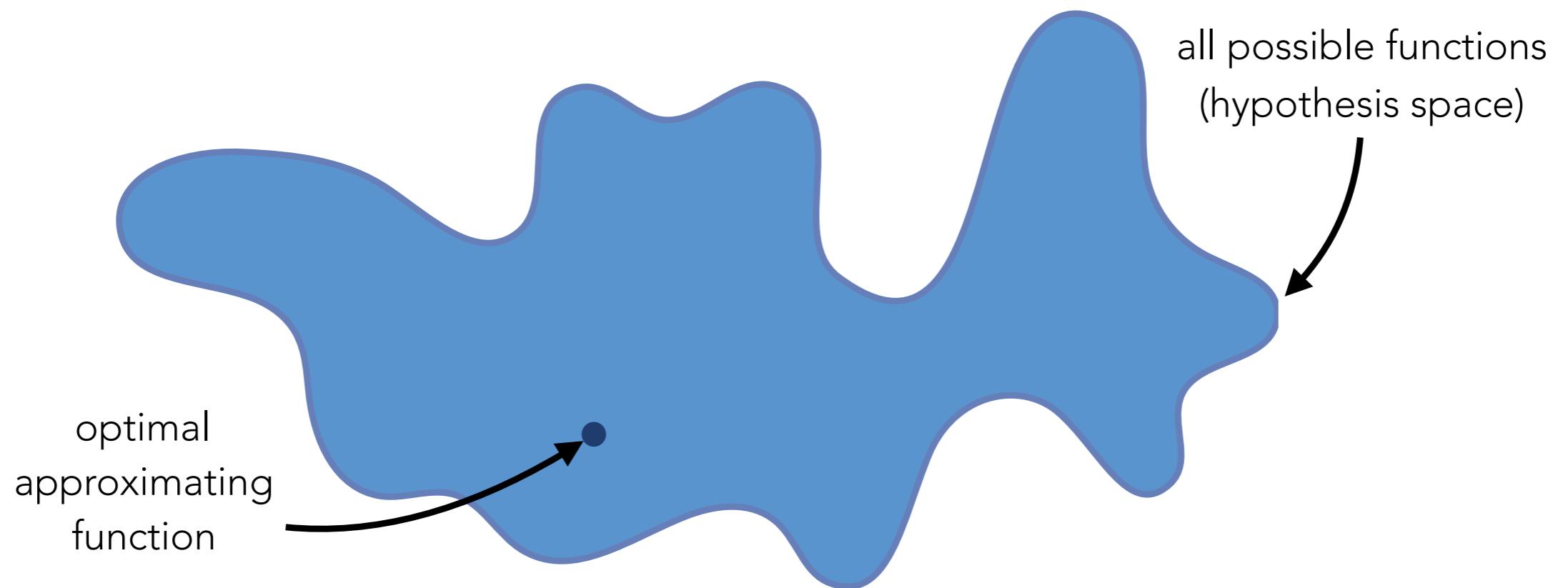
- **Multiple layers:** output of previous layer is **fed forward** to next layer after applying **non-linear** activation function
- **Fully connected:** many independent weights
- **Learning:** Use analytic derivatives and stochastic gradient descent to find optimal weights



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

Neural Network

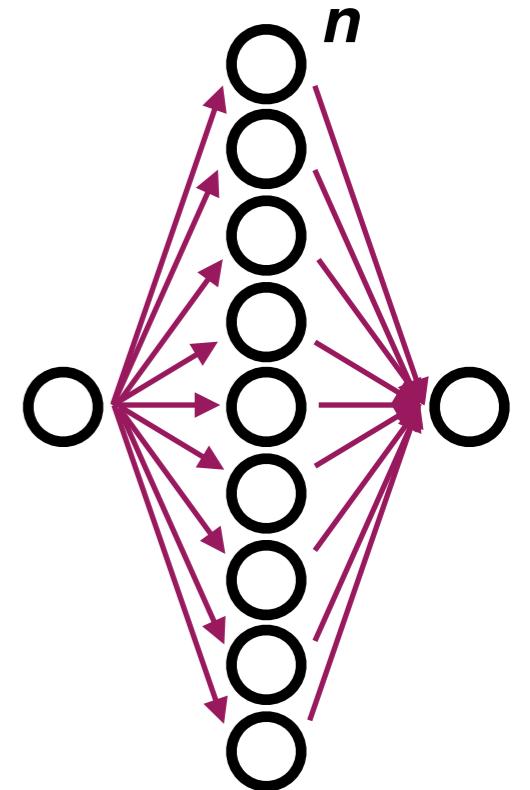
- Neural networks are universal function approximators, but we still must find an optimal approximating function



- We do so by adjusting the weights, architecture

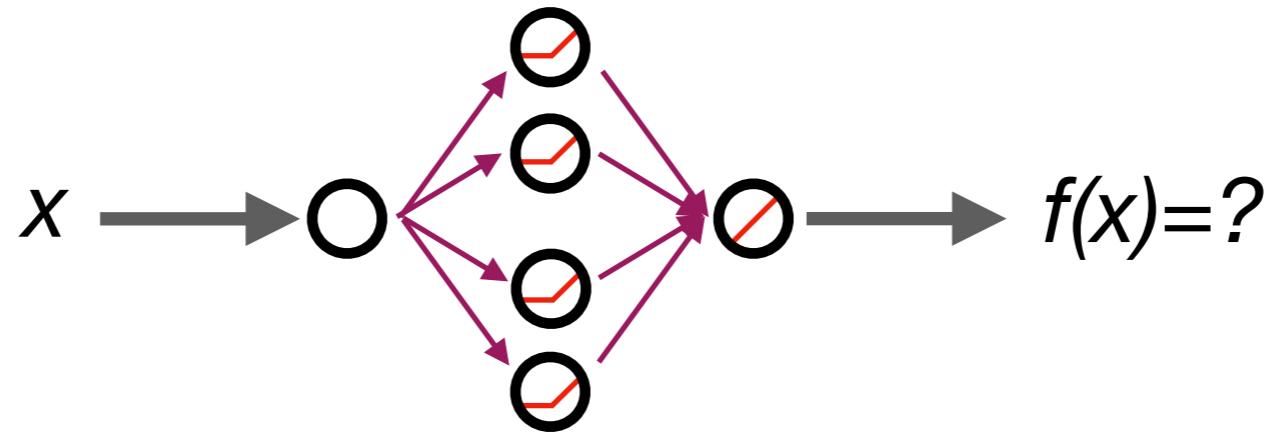
Approximating

- Layer: $\ell(\mathbf{x}) = \phi(\mathbf{W}^T \mathbf{x} + \mathbf{b})$
- Linear+Linear:
$$\begin{aligned}\ell_{\text{linear}}(\ell_{\text{linear}}(\mathbf{x})) &= \ell_{\text{linear}}(\mathbf{W}_1^T \mathbf{x} + \mathbf{b}_1) \\ &= \mathbf{W}_2^T (\mathbf{W}_1^T \mathbf{x} + \mathbf{b}_1) + \mathbf{b}_2 \\ &= \mathbf{W}_2^T \mathbf{W}_1^T \mathbf{x} + \mathbf{W}_2^T \mathbf{b}_1 + \mathbf{b}_2\end{aligned}$$
$$\tilde{\ell}_{\text{linear}}(\mathbf{x}) = \tilde{\mathbf{W}}^T \mathbf{x} + \tilde{\mathbf{b}}$$
- ReLU+Linear:
$$\begin{aligned}\ell_{\text{linear}}(\ell_{\text{ReLU}}(\mathbf{x})) &= \tilde{\mathbf{W}}^T \mathbf{x} + \tilde{\mathbf{b}}, \quad \mathbf{W}_1^T \mathbf{x} + \mathbf{b}_1 > 0 \\ &\quad \mathbf{b}_2, \quad \mathbf{W}_1^T \mathbf{x} + \mathbf{b}_1 < 0\end{aligned}$$



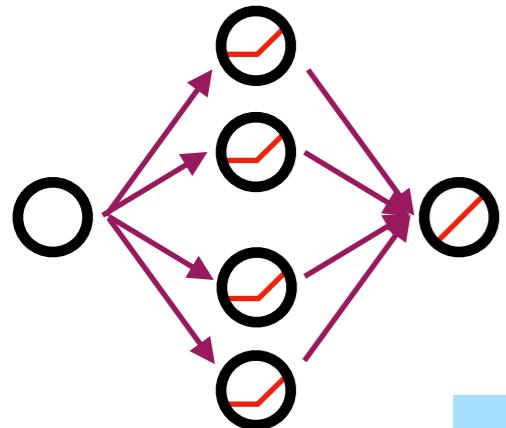
n boundaries

Approximating (Example)



What kind of function can this network architecture approximate?

Approximating (Example)



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

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

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

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

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

$$x > -\frac{b_4^1}{W_{14}^1}$$

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

$$-\frac{b_4^1}{W_{14}^1} > x > -\frac{b_3^1}{W_{13}^1}$$

Drop One

$$(W_{11}^2 W_{11}^1 + W_{21}^2 W_{12}^1 + W_{31}^2 W_{13}^1)x + W_{11}^2 b_1^1 + W_{21}^2 b_2^1 + W_{31}^2 b_3^1 + b_1^2$$

$$-\frac{b_3^1}{W_{13}^1} > x > -\frac{b_2^1}{W_{12}^1}$$

Drop Two

$$(W_{11}^2 W_{11}^1 + W_{21}^2 W_{12}^1)x + W_{11}^2 b_1^1 + W_{21}^2 b_2^1 + b_1^2$$

$$-\frac{b_2^1}{W_{12}^1} > x > -\frac{b_1^1}{W_{11}^1}$$

Drop Three

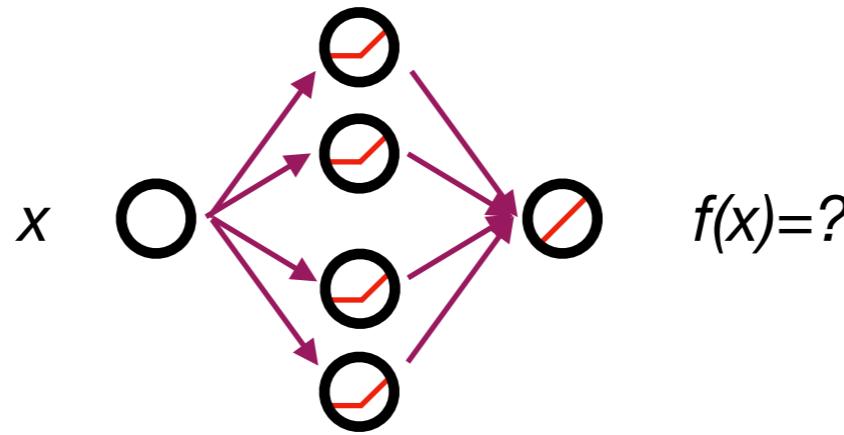
$$(W_{11}^2 W_{11}^1)x + W_{11}^2 b_1^1 + b_1^2$$

$$x < -\frac{b_1^1}{W_{11}^1}$$

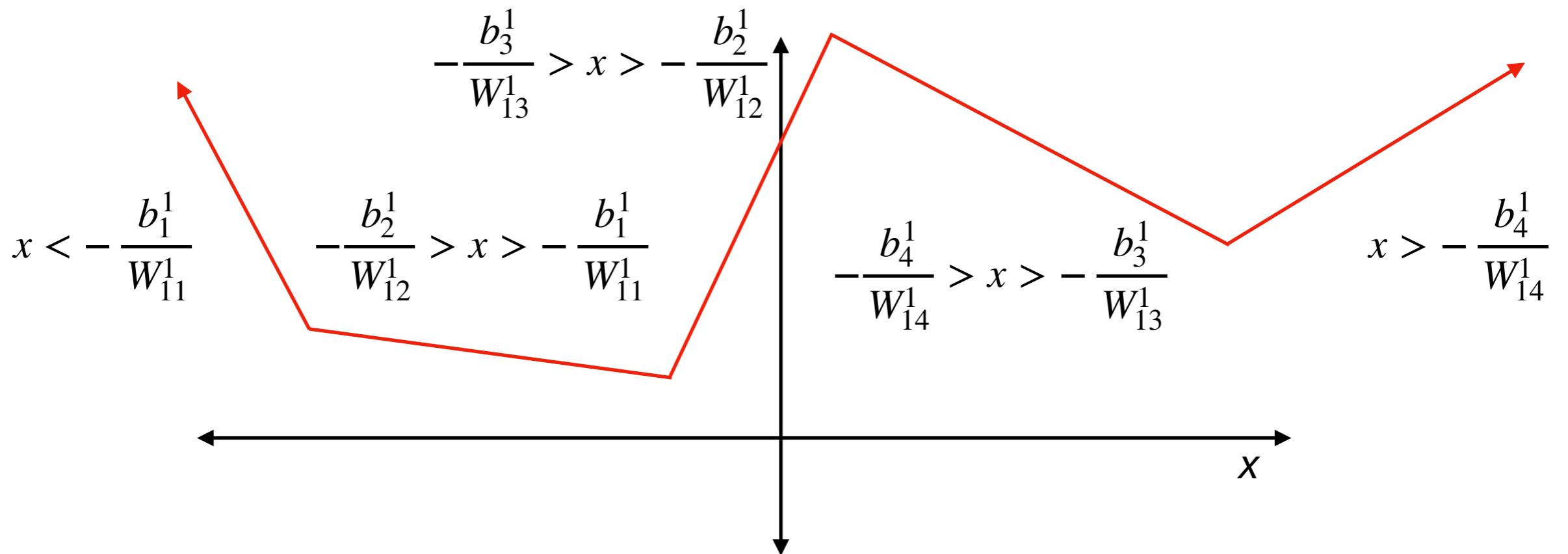
Drop It

$$b_1^2 \quad \text{Like it's Hot}$$

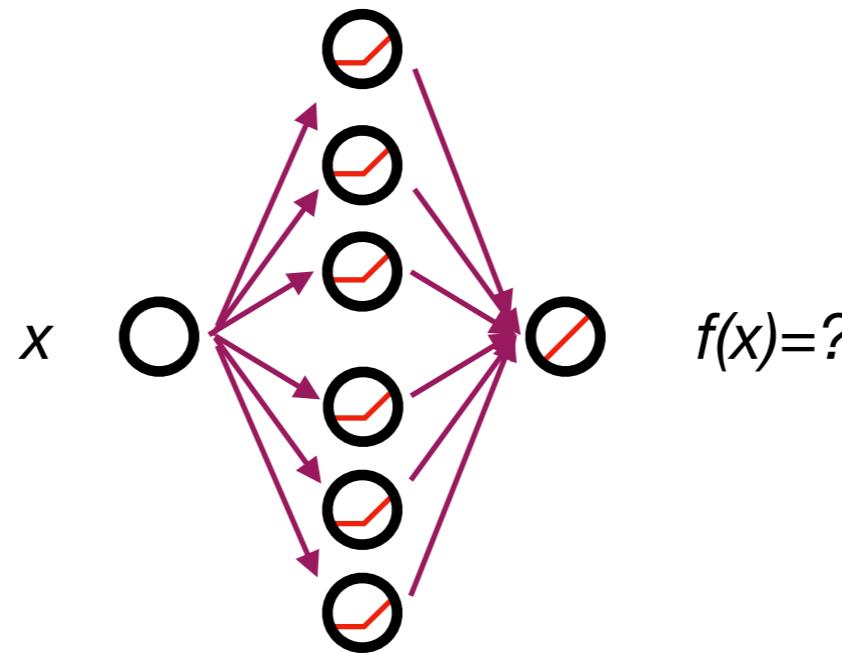
Approximating (Example)



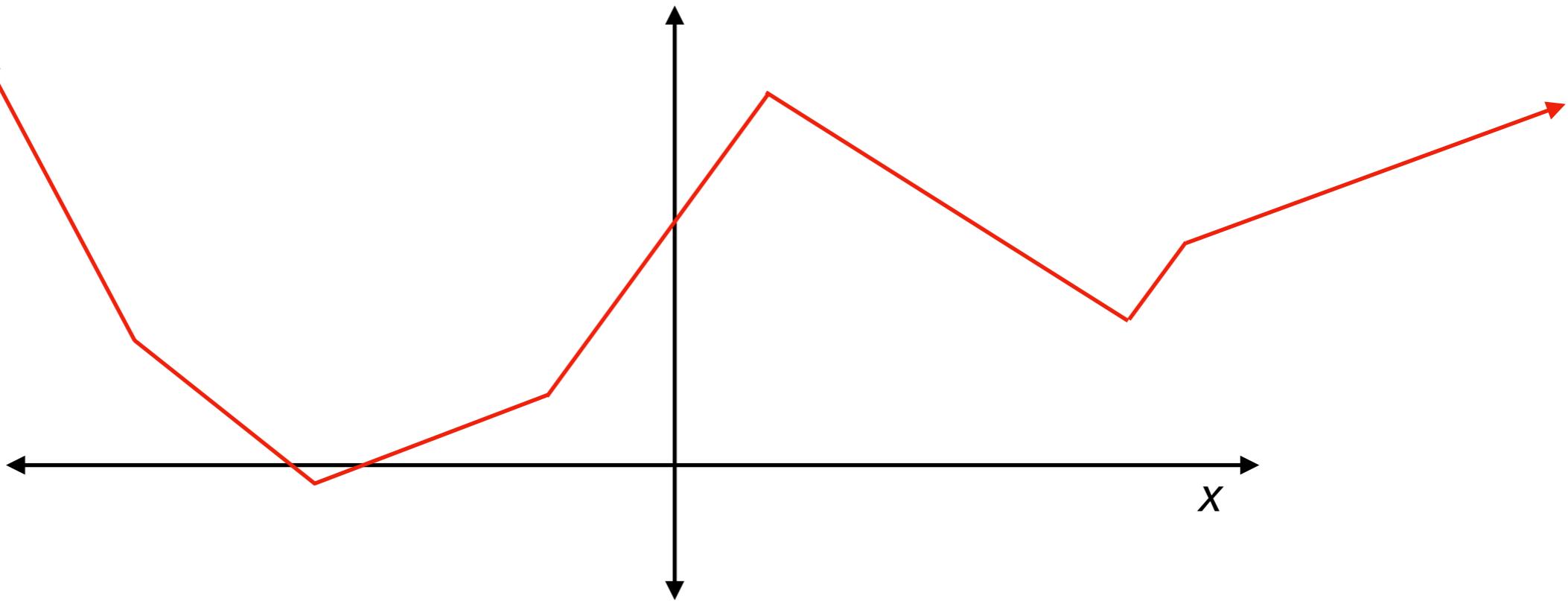
What kind of function can this network architecture approximate?



Approximating (Example)



What kind of function can this network architecture approximate?



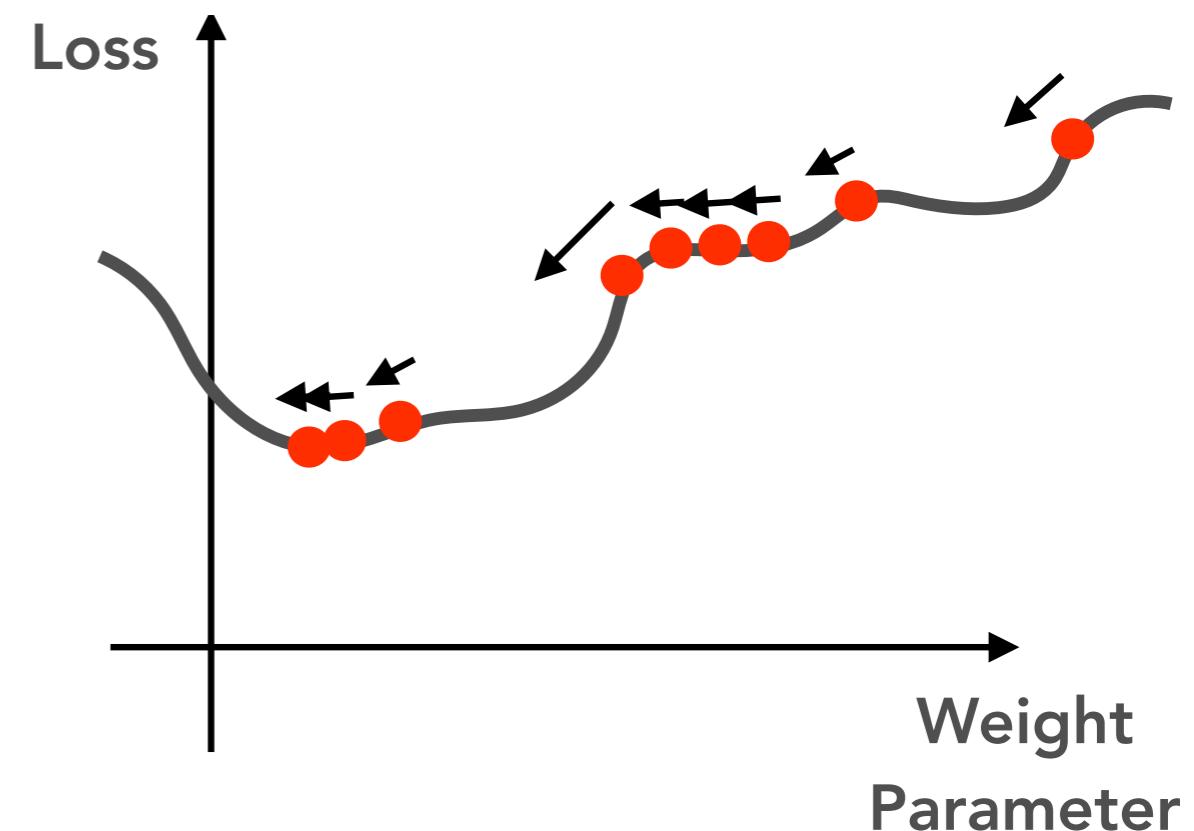
Learning - Optimization

- To learning the optimal weights we need the **derivative** of the loss w.r.t. weight

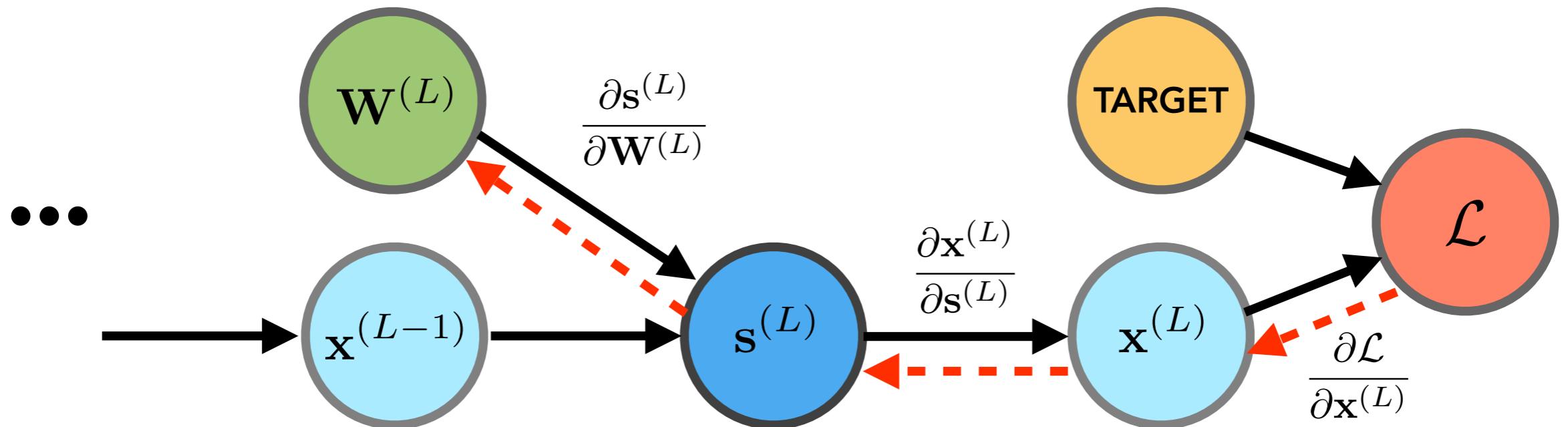
- $w \rightarrow w - \alpha \frac{\partial \mathcal{L}}{\partial w}$

- With multiple weights we need the gradient of the loss w.r.t. weights

- $\mathbf{w} \rightarrow \mathbf{w} - \alpha \nabla_{\mathbf{w}} \mathcal{L}$

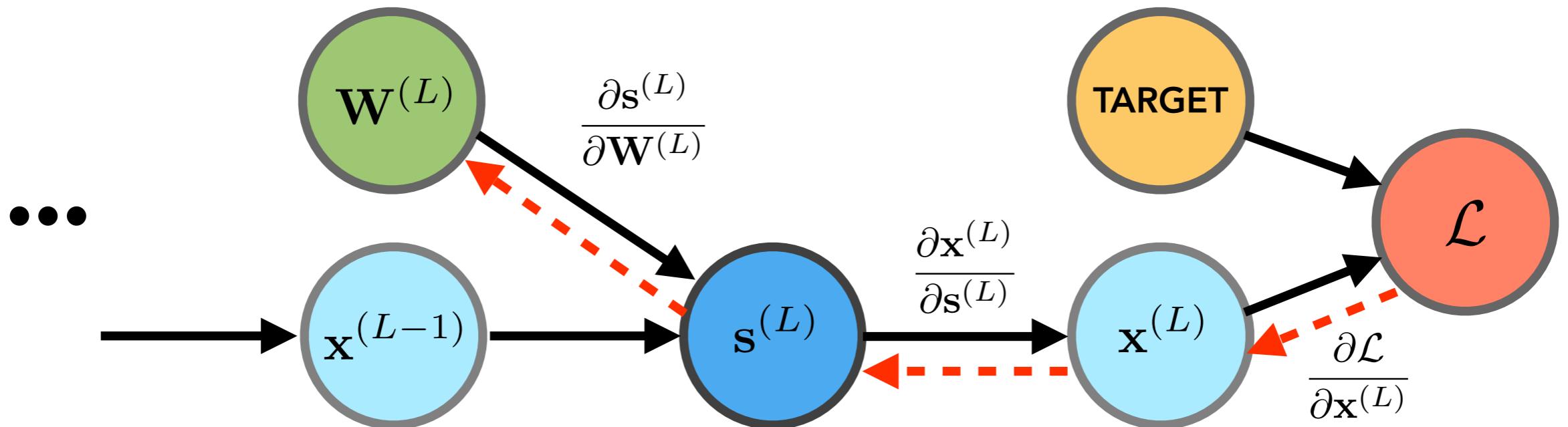


Backpropagation



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

Backpropagation



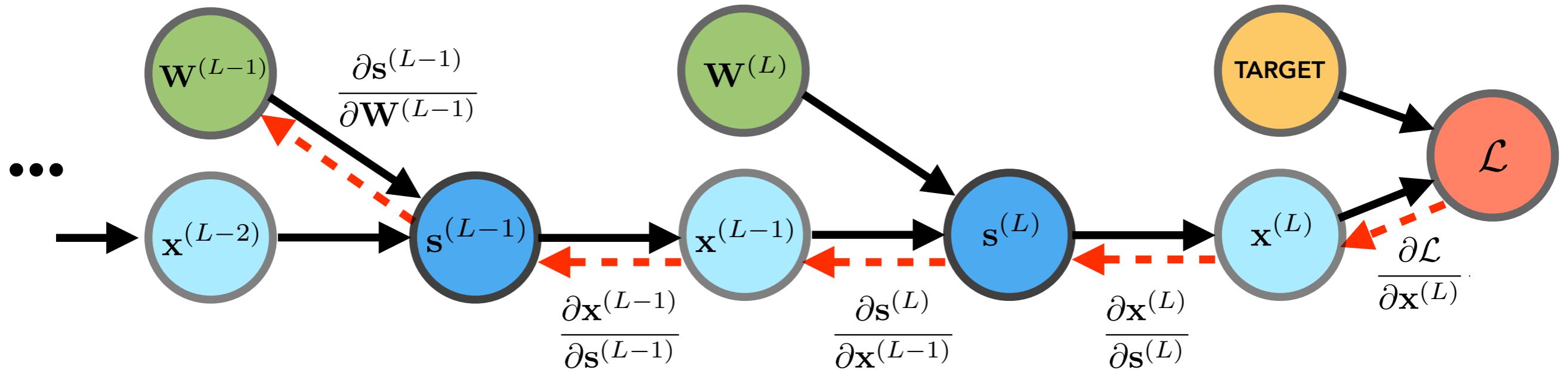
$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}^{(L)}} = \frac{\partial \mathcal{L}}{\partial x^{(L)}} \frac{\partial x^{(L)}}{\partial s^{(L)}} \frac{\partial s^{(L)}}{\partial \mathbf{W}^{(L)}}$$

depends on the
form of the loss

derivative of the
non-linearity

$$\begin{aligned} \frac{\partial}{\partial \mathbf{W}^{(L)}} (\mathbf{W}^{(L)} \tau_{x^{(L-1)}}) \\ = \mathbf{x}^{(L-1)} \tau \end{aligned}$$

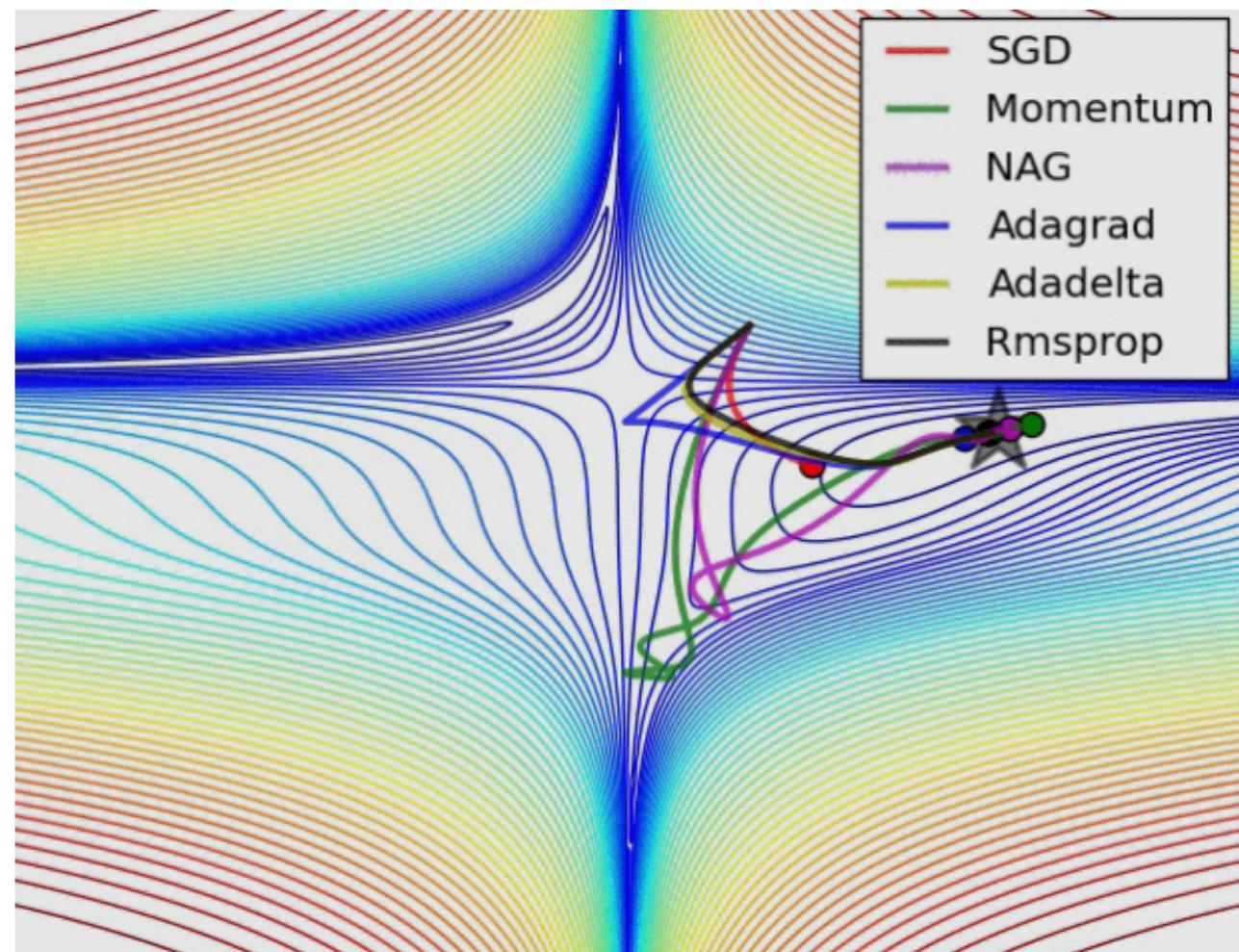
Backpropagation



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

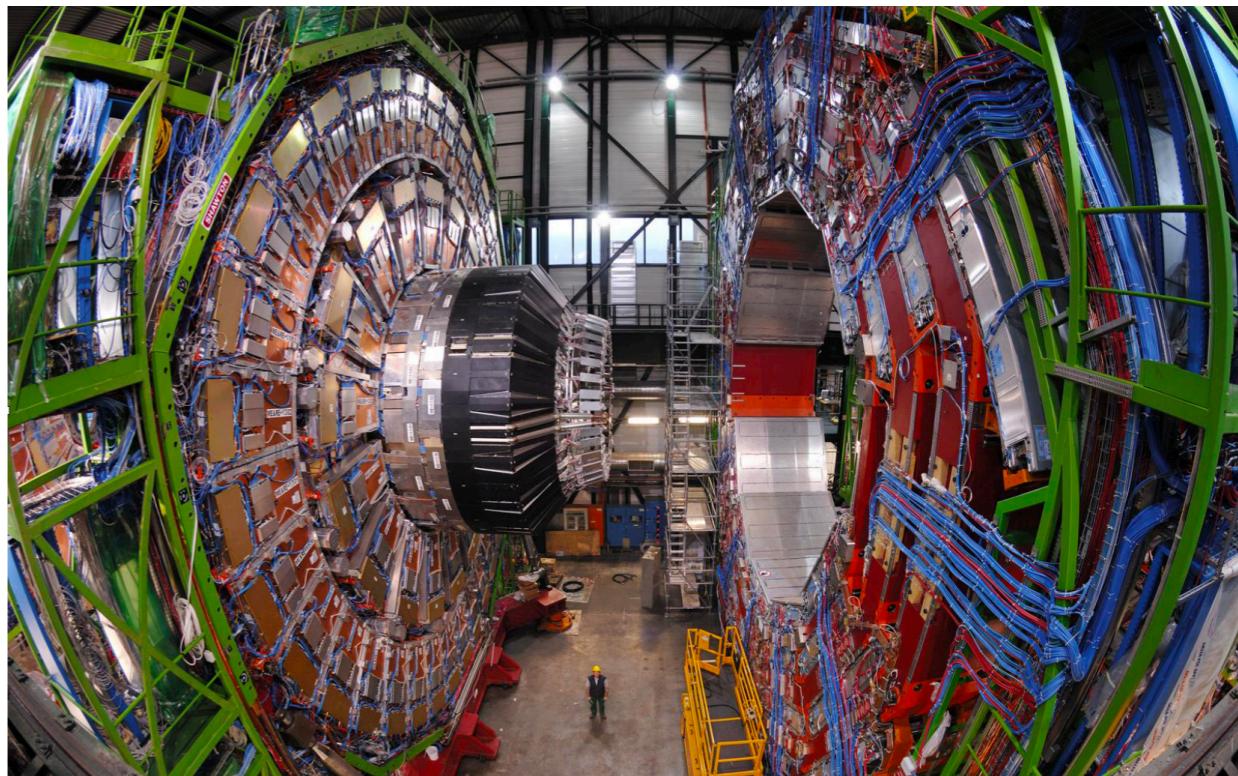
Stochastic Gradient Descent

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

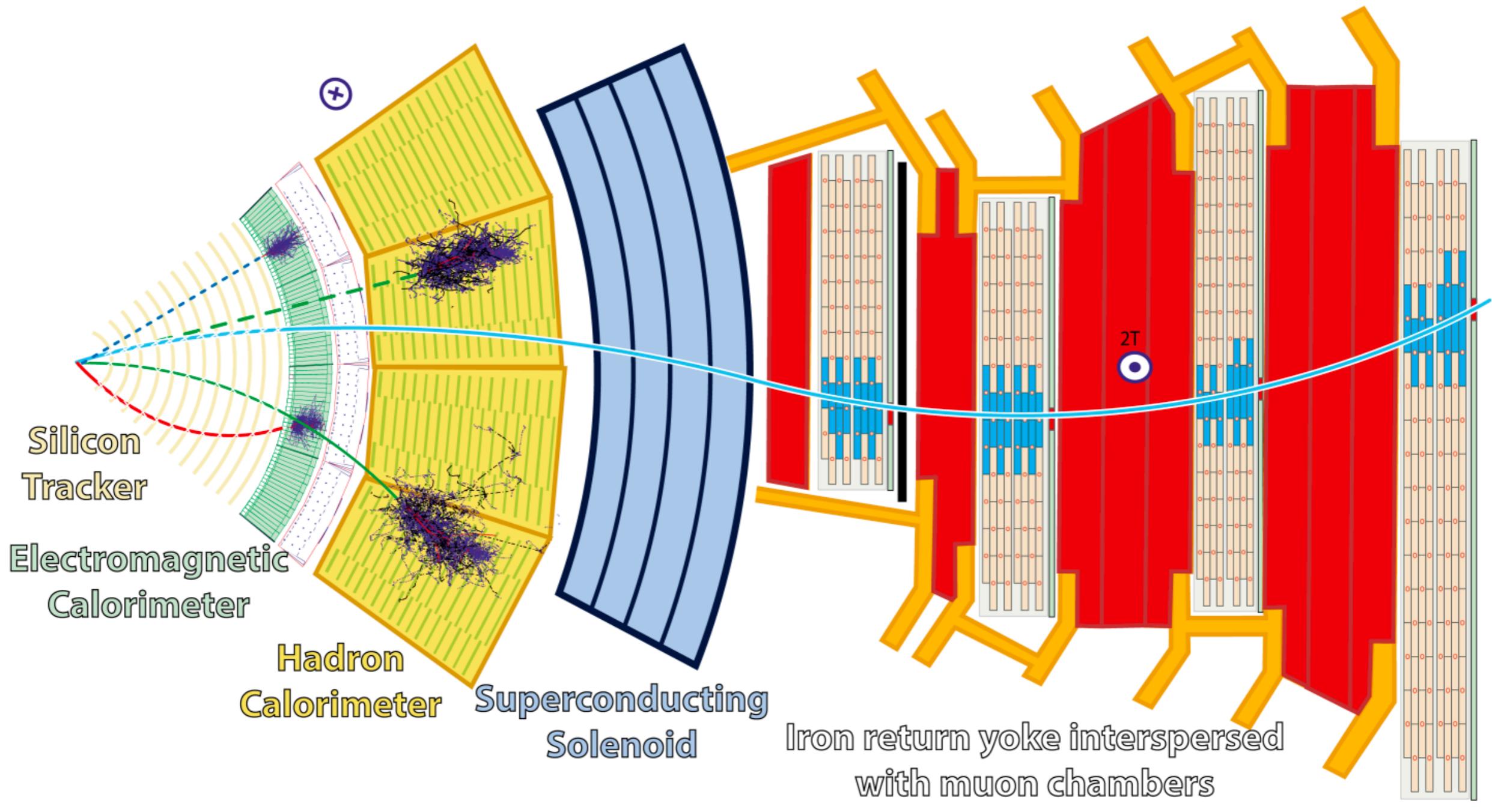


Large Hadron Collider

- Large Hadron Collider (proton - proton)
- Dedicated detectors to record data from collisions
 - Electromagnetic calorimeter (ECAL) is designed to measure energy of EM particles, $e + \gamma$ (**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**



CMS Detector



— Muon

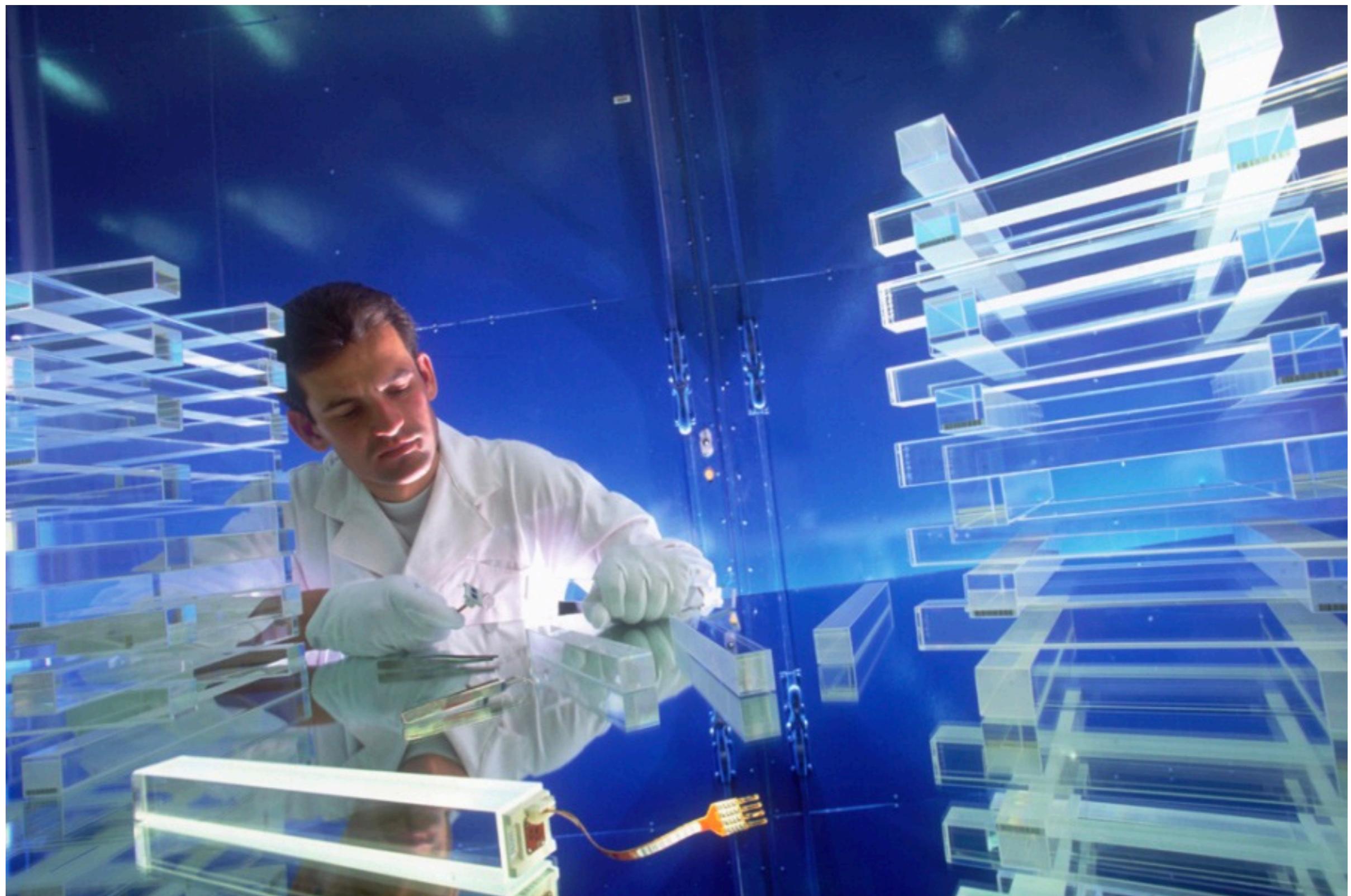
— Electron

— Charged hadron (e.g. pion)

— Neutral hadron (e.g. neutron)

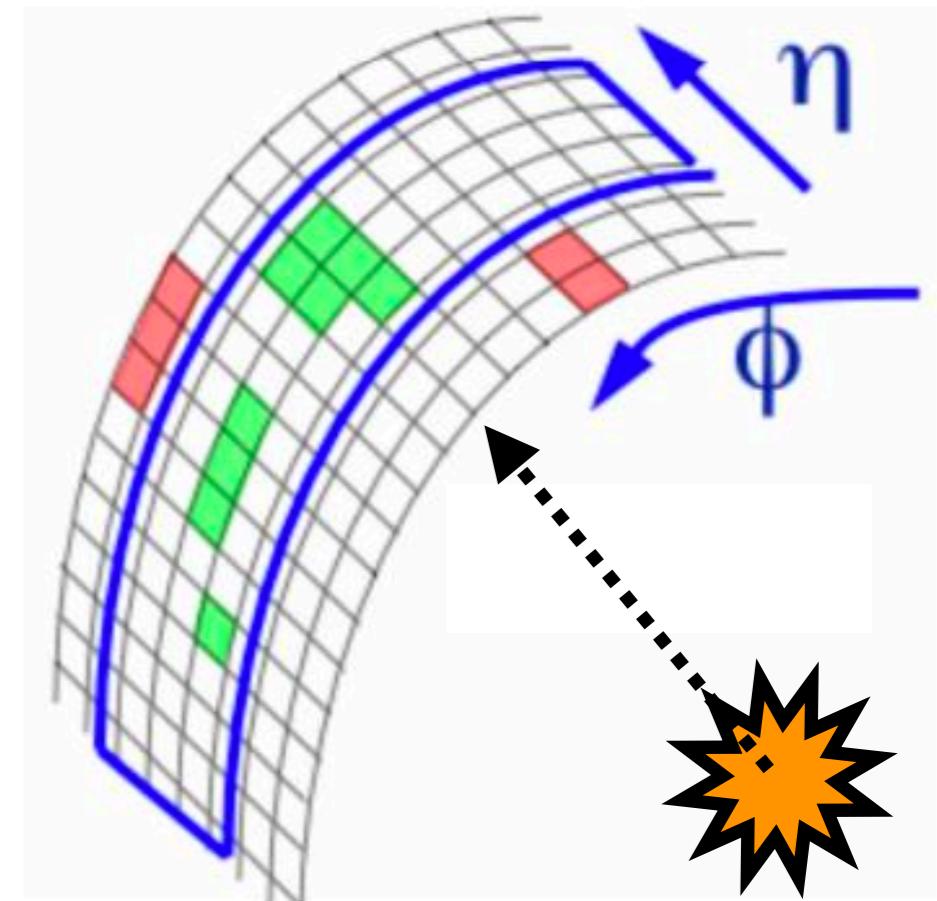
----- Photon

CMS ECAL

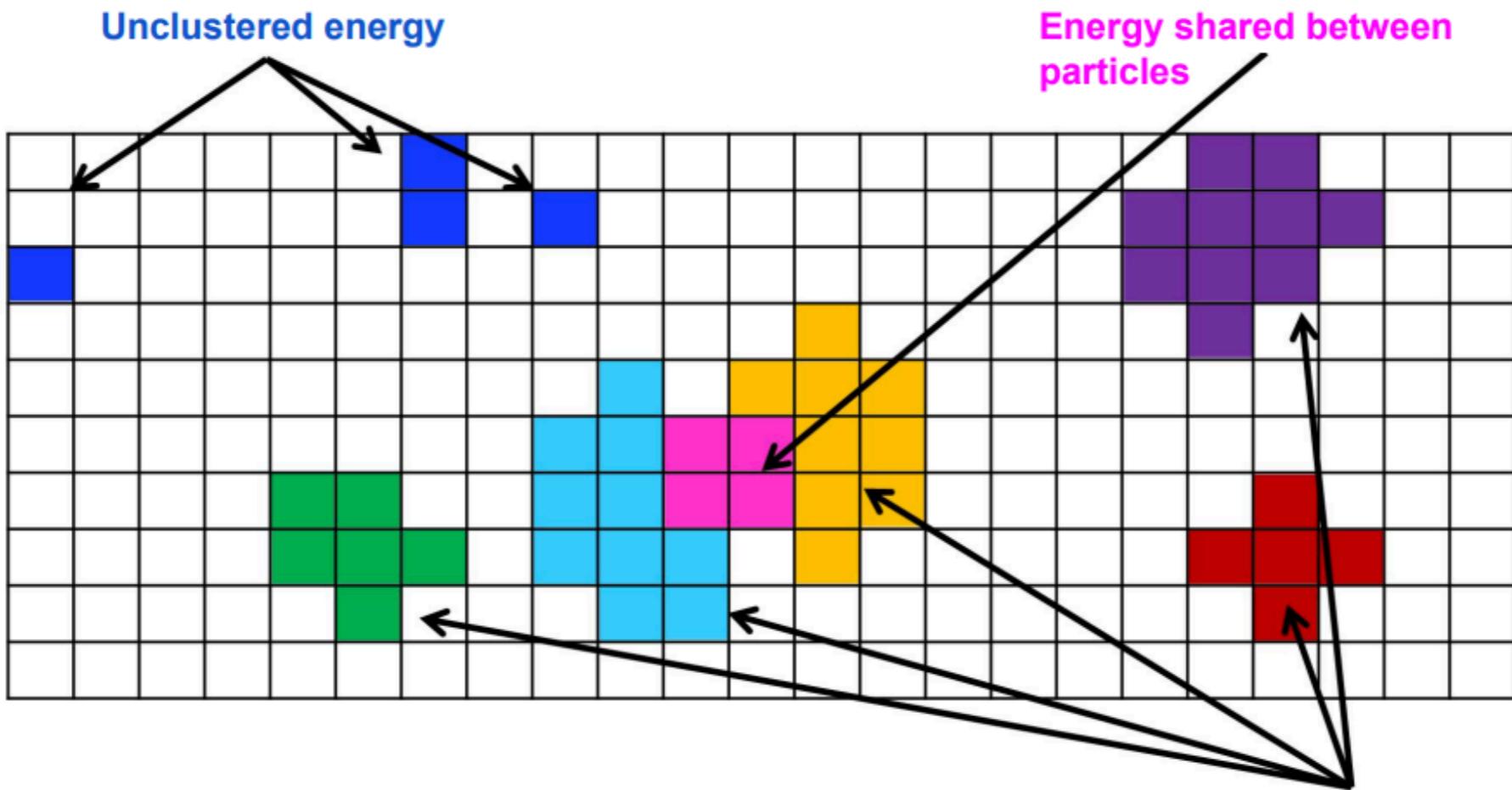


Dataset Explanation

- 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?

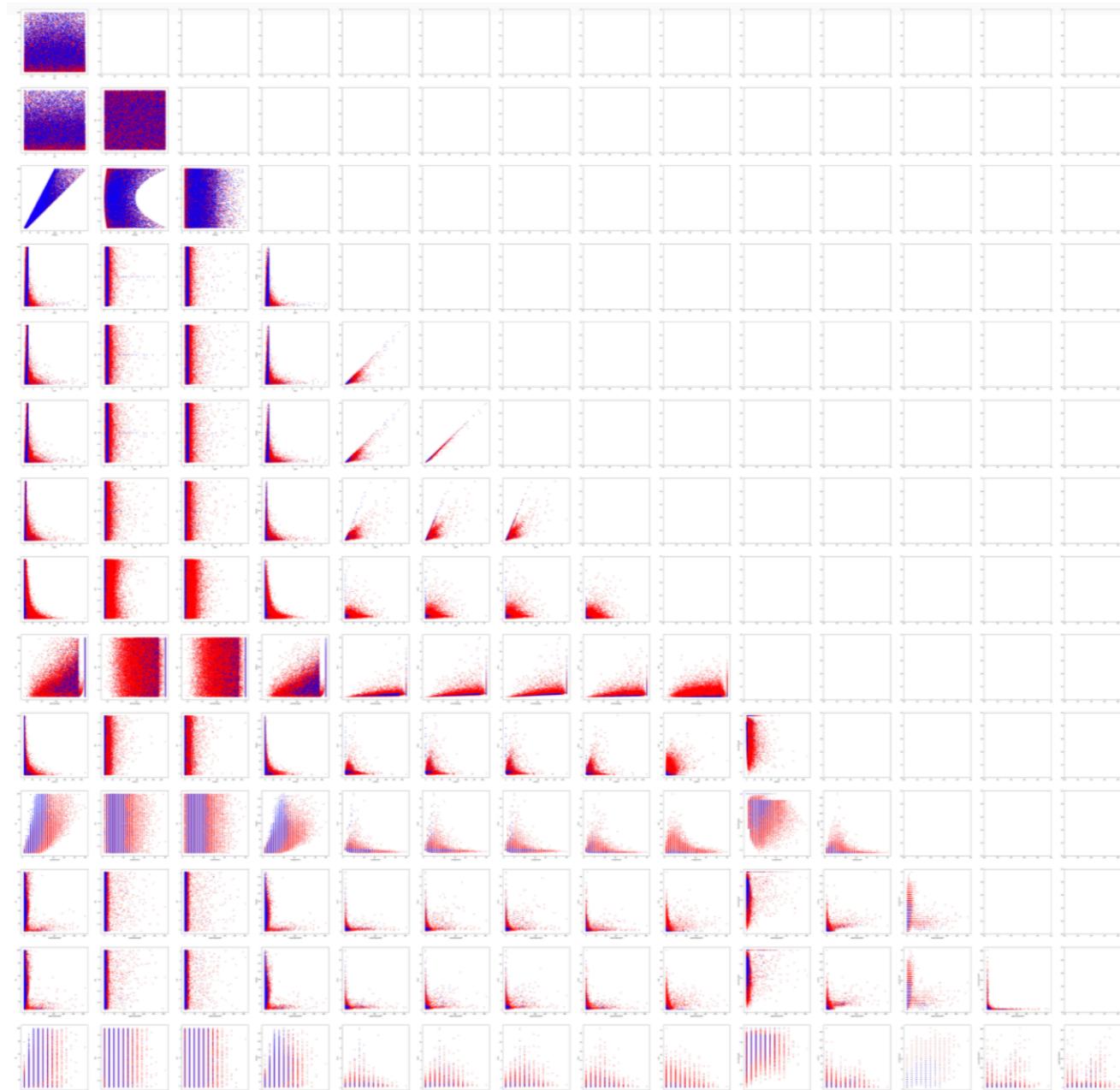


Dataset Visualization



- 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: $\mathcal{B} = \{x_1 \dots m\}$;
Parameters to be learned: γ, β

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

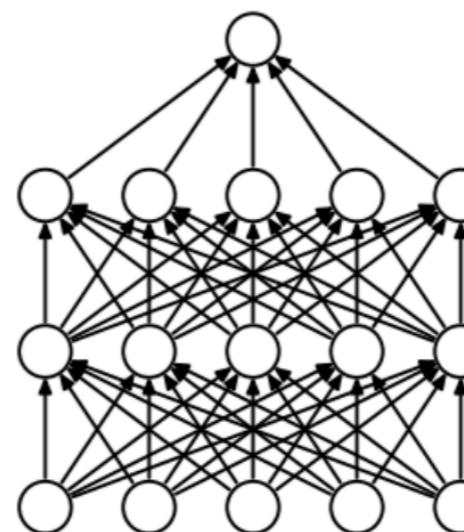
$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m} \sum_{i=1}^m x_i \quad // \text{mini-batch mean}$$

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

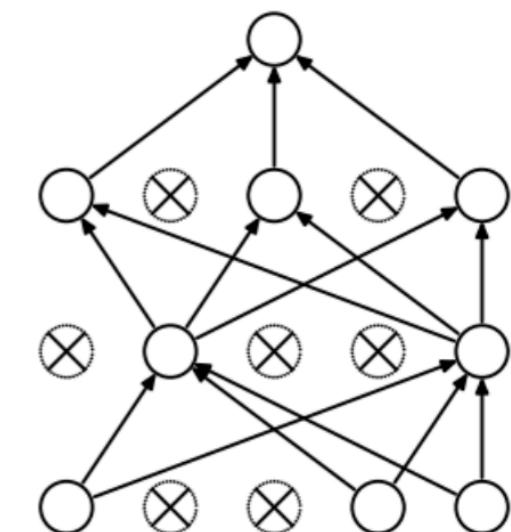
$$\hat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \quad // \text{normalize}$$

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

Dropout randomly removes weights



(a) Standard Neural Net



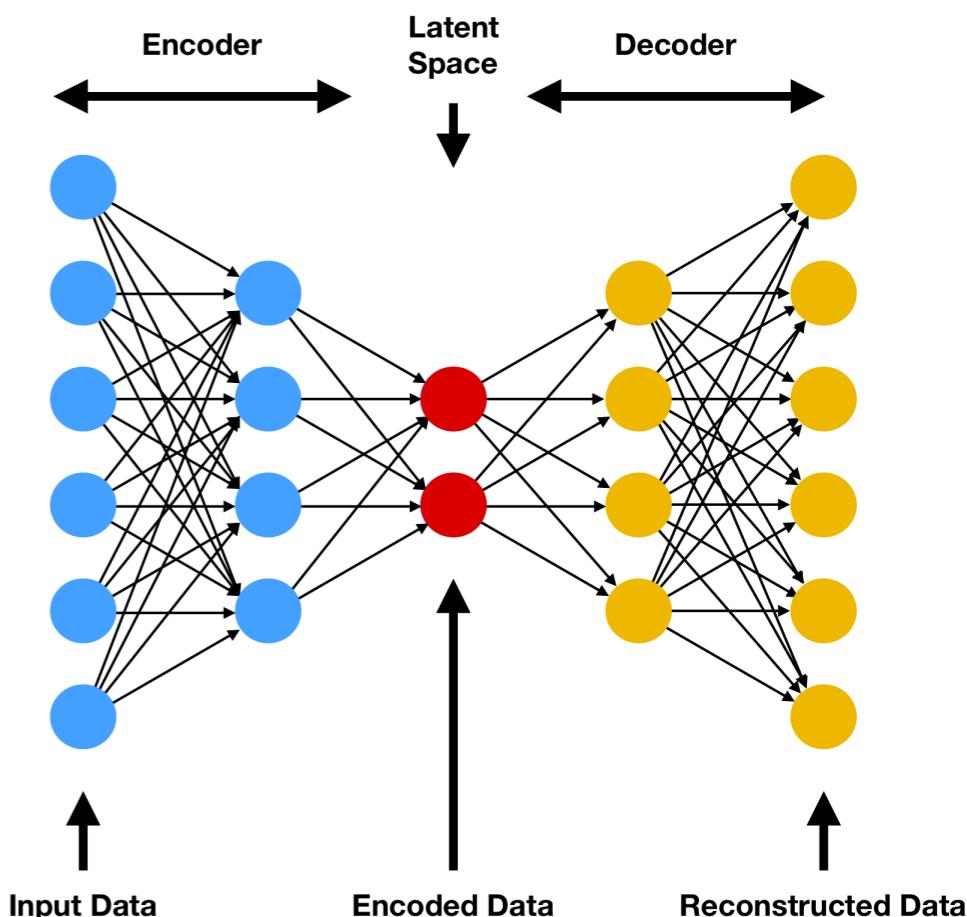
(b) After applying dropout.

- Both Strategies make NNs more robust to deviations

More Complex Architectures

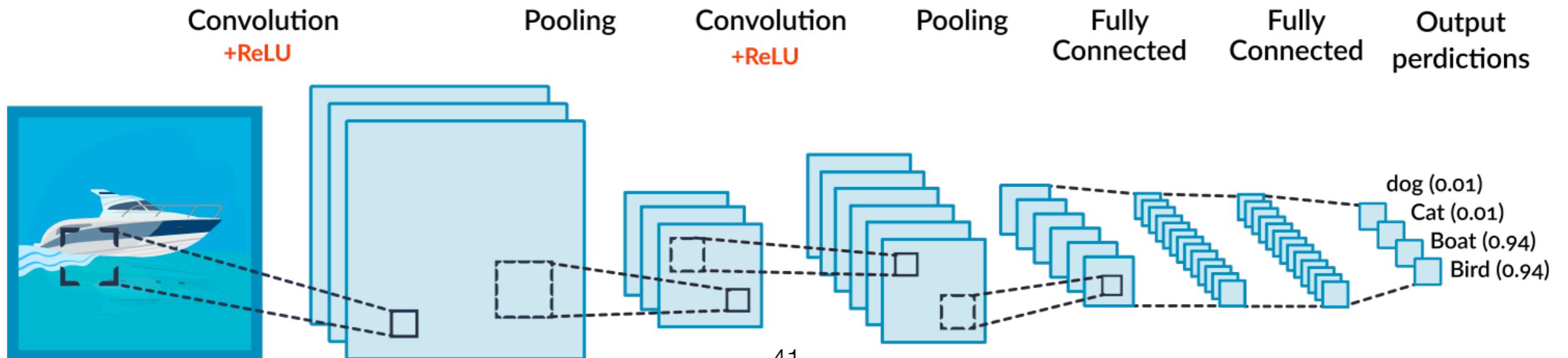
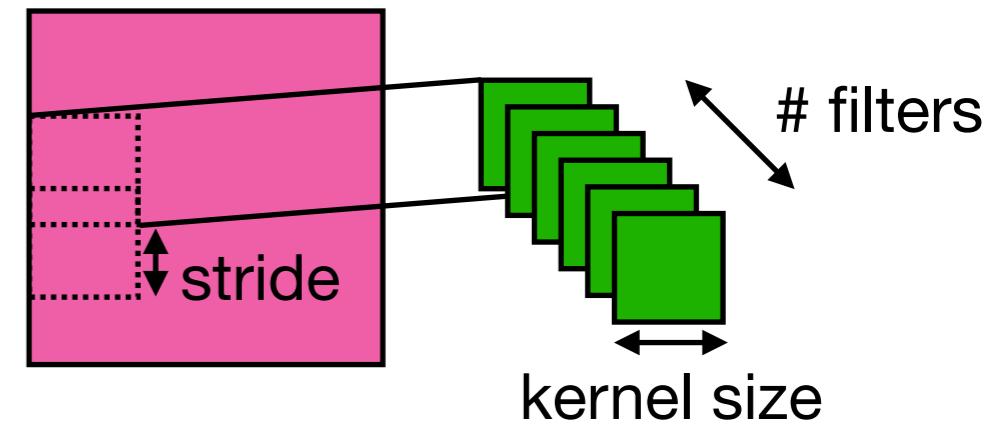
Unsupervised Learning

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



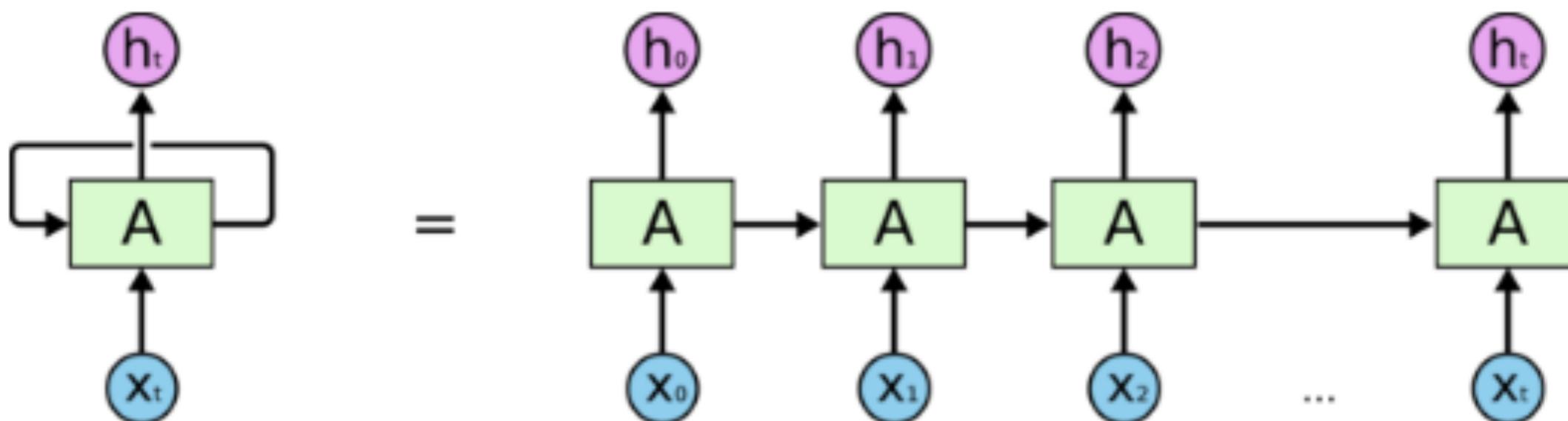
Convolutional Neural Network (CNN)

- 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



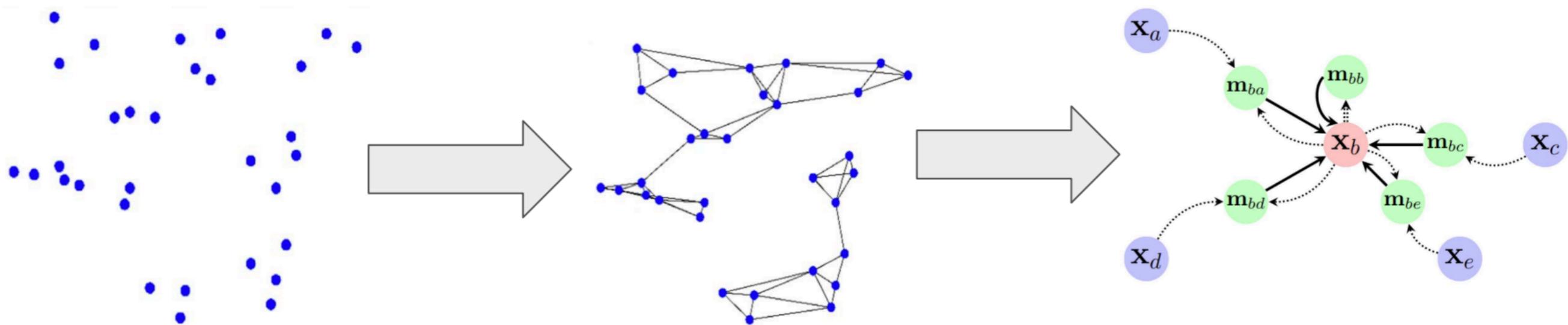
Recurrent Neural Network (RNN)

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



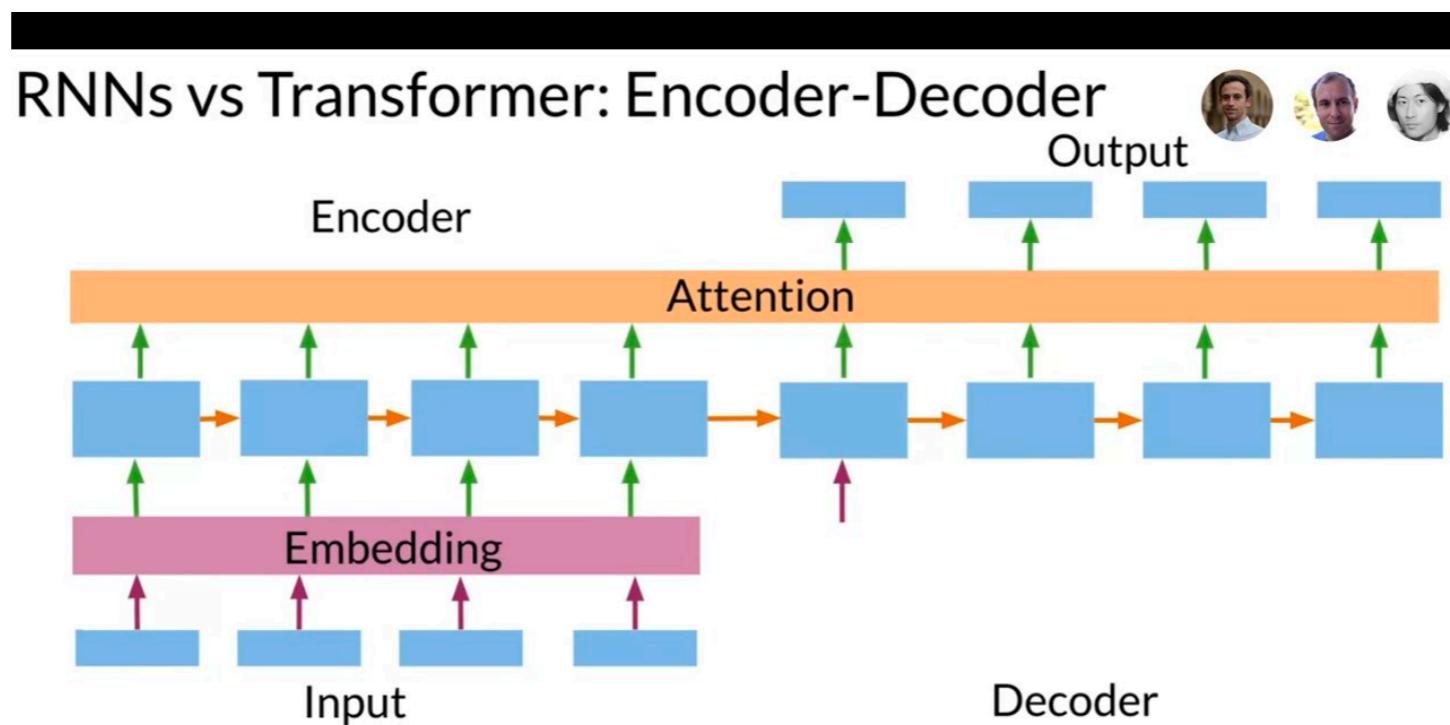
Graph Neural Network (GNN)

- 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