

Question 1:

- What is ML?
- What is it used for in HEP?
- Why is it useful?

Answer 1:

- Algorithm to minimize Loss function
- Machine learn and self-corrected w/o user input
- Series of functions that transfer input data to some output data.
- Goal: Show a machine some data to train on and the ability to generalize to unseen data
- Classification:
 - separate SM from BM
 - jet tagging
- Generation:
 - parameterize PDFs
 - hadronization
 - detector unfolding
 - reweighting MC
 - event generation
- Unsupervised (no parameterization)
 - Make unbiased guesses
 - Better $\propto \sqrt{B}$ from NN
 - Higher tagging efficiency
 - Better unweighting efficiencies

Loss function

Given 2 classes (signal, bkg) we need metric to know how well the computer separates them

Binary Cross Entropy:

$$L_b = - \sum_i (y_i \log(p_i) + (1-y_i) \log(1-p_i))$$

p_i is the predicted probability ($0 = \text{bkg}$, $1 = \text{signal}$)
 y_i the truth label ($0 = \text{bkg}$, $1 = \text{signal}$)

Question 2:

How to generalize for n-classes?

Answer 2:

$$L = - \sum_{c=1}^n \sum_i y_{ci} \log(p_{ci})$$

$y_{ci} = \begin{cases} 1 & \text{if } i \text{ is class } c \\ 0 & \text{otherwise} \end{cases}$

Kullback-Leibler:

$$L = \sum_i p(x_i) \log \left(\frac{p(x_i)}{q(x_i)} \right) \rightarrow \int p(x) \log \left(\frac{p(x)}{q(x)} \right) dx$$

$$= \sum_i p(x_i) \log \frac{1}{q(x_i)} - \sum_i p(x_i) \log \frac{1}{p(x_i)}$$

$$= L_B(p, q) - L_B(p, p)$$

EMD:

Two piles of sand. The EMD is the amount of sand moved times the distance the sand is moved.

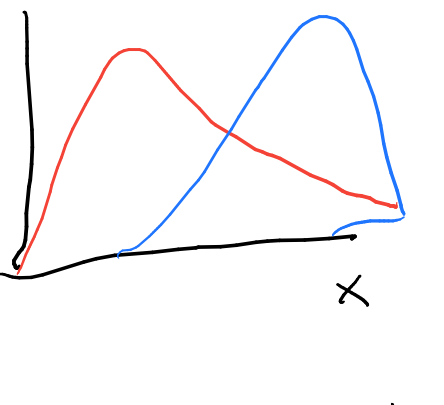
$$P = \{ (p_1, w_{p1}), \dots, (p_n, w_{pn}) \}$$

$$Q = \{ (q_1, w_{q1}), \dots, (q_m, w_{qm}) \}$$

$$EMD = \frac{\sum_{i=1}^n \sum_{j=1}^m f_{ij} d_{i,j}}{\sum_{i=1}^n \sum_{j=1}^m f_{i,j}}$$

Classification:

Neyman-Pearson Lemma:

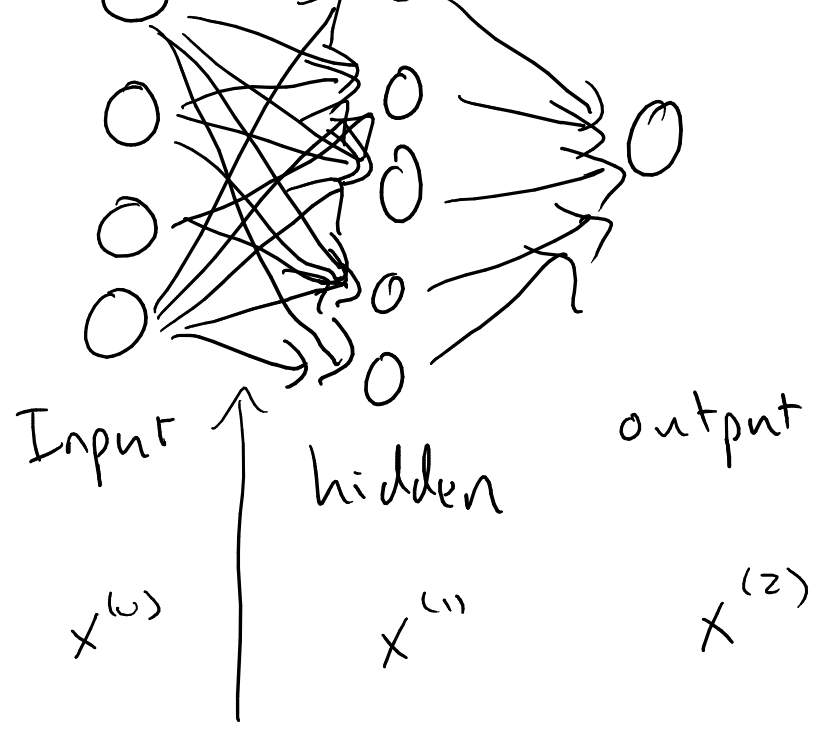


The likelihood ratio provides the optimal separation of signal + background when used as a test statistic

⇒ Ultimate goal: "learn" likelihood ratio

Neural Networks:

$$f: \mathbb{R}^n \rightarrow \mathbb{R}^p$$



weights (w): matrix
 bias (b): vector

activation function (a)
 typically non-linear

$$z_j^{(0)} = w_{jc}^{(0-1)} x_c^{(0-1)} + b_j^{(0-1)}$$

$$x^{(0)} = a(z^{(0)})$$

Ex:

Binary step: $\begin{cases} 0 & x < 0 \\ 1 & x \geq 0 \end{cases}$

Logistic (sigmoid): $\sigma(x) = \frac{1}{1+e^{-x}}$

ReLU (Rectified Linear Unit): $\begin{cases} 0 & \text{if } x < 0 \\ x & x \geq 0 \end{cases} = \max(0, x)$

Leaky ReLU: $\begin{cases} 0.01x & x < 0 \\ x & x \geq 0 \end{cases}$

Question 3:

- ① Allowing arbitrary width layers, How many layers are needed to approximate any continuous function?
- ② Allowing arbitrary # of layers, w/ ReLU activation what is the min width to approximate any continuous function?

Answer 3:

- ① one layer hidden
~~∞ layers hidden~~
~~2 layers hidden~~
- ② size of input + 1
~~2~~
~~size of output~~

Universal approximation theorems:

Gradient Descent:

Minimize loss by adjusting weights + biases

Given: $g(x, \theta)$
 ↑ NN ↑ weights + biases

$$\text{Let } \theta^{i+1} = \theta^i - \eta \nabla L(x, \theta)$$

η = learning rate

η too small → never converges

η too large → jumping around, diverge

How do we calculate gradients?

- Finite differences: numerically inaccurate
- Analytically: Θ (million) parameters, impossible by hand

New numbers:

$$A = a + b\epsilon, \quad \epsilon^2 = 0$$

Question 4 (HW): $X = x + \epsilon, \quad Y = y + \epsilon$

- ① x^2
- ② $x+y$
- ③ xy
- ④ $\frac{x}{y}$
- ⑤ $\cos(xy)$

$$\cos(A) = \cos(a) - \sin(a)b\epsilon$$