Machine Learning: Lecture #1

NOT on BDT, Boltzmann, etc. NN

Question 1:

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

Answer 1:

- Algorithm to minimize Loss function
  - Machine learns 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:
  - supervised SM from BM
  - jet tagging

Generation:
- parameterize PDFs
- hadronization
- detector unfolding
- reweighting MC
- event generation

- Unsupervised (no parameterization)

Make unbiased guesses
Better $S/\sqrt{B}$ than NN
Higher bagging efficiency
Better reweighting efficiencies

Loss function

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

Binary Cross Entropy:

$$L_\theta = -\sum_i \left(y_i \log(p_i) + (1-y_i)\log(1-p_i) \right)$$

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:

$$L = \sum_{c=1}^{M} L_\theta^{(c)}$$

$$= \sum_{c=1}^{M} \sum_{i=1}^{N^{(c)}} L_\theta^{(c)}$$

$$= -\sum_c \sum_{i=1}^{N^{(c)}} y_i \log(p_{ic})$$

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

Kullback-Leibler:

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

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

$$= L_\theta(P, Q) - L_\beta(P, P)$$

EMD:

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

$$P = \{(p_1, w_{p_1}), \ldots, (p_n, w_{p_n})\}$$

$$Q = \{(q_1, w_{q_1}), \ldots, (q_m, w_{q_m})\}$$

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

Classification:

Neyman-Pearson lemma:

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

$\Rightarrow$ Ultimate goal: "learn" likelihood ratio

Neural Networks:

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

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

activation function (α) typically non-linear

$$z_j^{(l)} = w_{jk}^{(l-1)} x_k^{(l-1)} + b_j^{(l-1)}$$

$$x^{(l)} = \alpha\left(z^{(l)}\right)$$

Ex: Binary step:

$$\begin{cases} 0 & \text{if } x < 0 \\ 1 & x \geq 0 \end{cases}$$

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 (uniform) 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
   ~~=~~
   ~~size of output~~

Universal approximation theorems:

Gradient Descent:

Minimize loss by adjusting weights + biases

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

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

η = learning rate

η too small $\longrightarrow$ never converges

η too large $\longrightarrow$ jumping around, diverge

How do we calculate gradients?

- Finite differences: numerically inaccurate
- Analytically: $O(1\text{million})$ parameters, impossible by hand

New numbers:

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

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

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

1. X^2
2. $X+Y$
3. XY
4. $\frac{X}{Y}$
5. $\cos(XY)$