Neural networks

Alexander Khanov

PHYS6260: Experimental Methods is HEP
Oklahoma State University

November 3, 2023

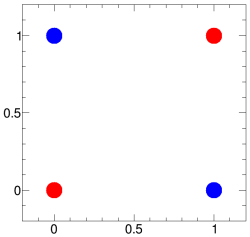
Formulation of the problem

- Let $\mathbf{x}$ be the input variables ("features"), and $\mathbf{y}$ be the outputs ("labels")
  - the labels can be binary (classification), or continuous (regression)
- Consider a function $\mathbf{y} = \mathbf{f}(\mathbf{x}, \mathbf{p})$ that depends on a vector of parameters $\mathbf{p}$
- Given a training set $\{(\mathbf{x}_i, \mathbf{y}_i)\}$ of $\mathbf{x}$, $\mathbf{y}$ pairs, we want to find the values of $\mathbf{p}$ that minimize the distance between $\mathbf{f}(\mathbf{x}_i, \mathbf{p})$ and $\mathbf{y}_i$ ("loss function")
  - an example of distance is mean squared loss $\chi^2 = \sum_i (\mathbf{y}_i - \mathbf{f}(\mathbf{x}_i, \mathbf{p}))^2$
- The task is to come up with a "model" (the actual function $\mathbf{f}(\mathbf{x}, \mathbf{p})$) and "train" it (calculate the optimal values of $\mathbf{p}$)

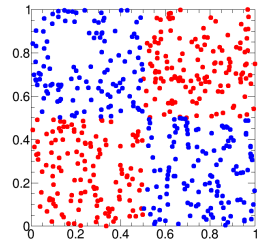
Limitations of MVA methods

- The MVA methods we have considered so far are fairly limited in terms of problems they can solve
- Canonical example: the XOR problem $(x_1, x_2) \to y$

binary: $(0,0) \to 0$, $(1,0) \to 1$, $(0,1) \to 1$, $(1,1) \to 0$

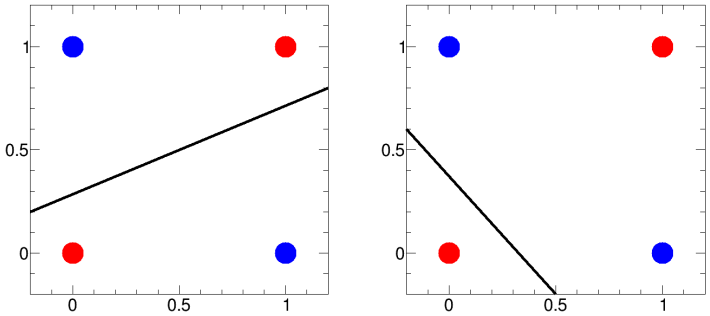


continuous: 0 if $(x_1 - 0.5)(x_2 - 0.5) > 0$, 1 otherwise

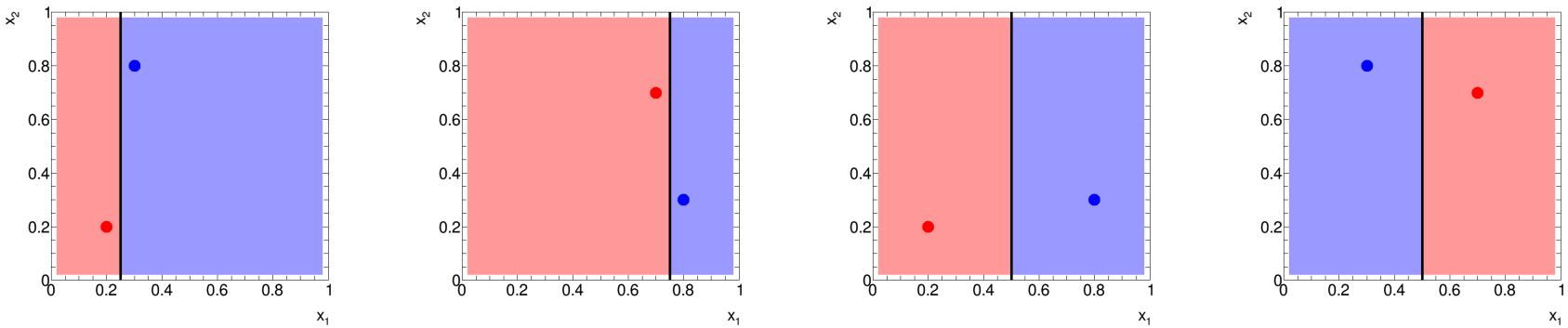


Limitations of MVA methods (2)

- XOR problem is not linearly solvable
  - no matter how you draw the separation line, it will only correctly classify 3 out of 4 points at most



- Decision trees will also be in trouble since they look at one variable at a time



meaningless – the order is arbitrary

good for $x_2 < 0.5$

good for $x_2 > 0.5$

Limitations of MVA methods (3)

- Many machine learning methods fail as the number of input variables becomes high
  - the curse of dimensionality: the number of possible distinct configurations increases exponentially with the number of variables
  - can't evaluate statistical density for cells without training examples

"Deep learning," I. Goodfellow, Y. Bengio, and A. Courville

- Machine learning algorithms need some idea on what kind of function they are trying to learn
  - a typical assumption is smoothness (local constancy)
  - this approach does not work well for small training examples created by complicated underlying functions

Neural network structure

- A simple neural network consists of "layers" which are collections of "neurons"
  - a neuron i in layer j is characterised by its "activation value" a_i^j
  - the activation values of neurons in layer $j+1$ are related to those in the previous layer j as $a_i^{j+1} = \sigma\left(\sum_k w_{ik}^j a_k^j + b_i^j\right)$, where $\sigma(t)$ is "activation function", w_{ik}^j are "weights", and b_i^j are "biases"
- In order for the NN to work, $\sigma(t)$ must be nonlinear

Example: linear activation

- For the XOR problem, consider linear activation function $\sigma(t) = t$
- The NN with linear activation will look like $y = b + w_1 x_1 + w_2 x_2$ (two inputs x_1, x_2, one output y)
  - minimizing the mean squared loss $\chi^2 = \sum_i (y_i - (b + w_1 x_{1i} + w_2 x_{2i}))^2$,

    we arrive at

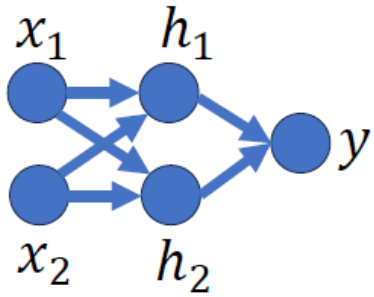
$$\begin{cases} b\sum_i 1 & + & w_1 \sum_i x_{1i} & + & w_2 \sum_i x_{2i} & = & \sum_i y_i \\ b\sum_i x_{1i} & + & w_1 \sum_i x_{1i}^2 & + & w_2 \sum_i x_{1i}x_{2i} & = & \sum_i x_{1i} y_i \\ b\sum_i x_{2i} & + & w_1 \sum_i x_{1i}x_{2i} & + & w_2 \sum_i x_{2i}^2 & = & \sum_i x_{2i} y_i \end{cases}$$

  - using the set $\{(x_{1i}, x_{2i}, y_i)\} = \{(0,0,0), (1,0,1), (0,1,1), (1,1,0)\}$, we find $b = 0.5$, $w_1 = w_2 = 0$
- Our "NN" always returns 0.5 – linear activation doesn't work

Example: RELU activation

- The activation function does not have to be complicated
  - RELU: $R(t) = \begin{cases} 0, t < 0 \\ t, t \geq 0 \end{cases}$
- Consider a NN with one hidden layer of two neurons:

$$\begin{cases} h_1 = R(W_{11}x_1 + W_{21}x_2 + b_1) \\ h_2 = R(W_{12}x_1 + W_{22}x_2 + b_2) \\ y = w_1 h_1 + w_2 h_2 \end{cases}$$



- $W = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$, $\mathbf{b} = \begin{pmatrix} 0 \\ -1 \end{pmatrix}$, $\mathbf{w} = \begin{pmatrix} 1 \\ -2 \end{pmatrix}$ does the trick:

x_1	x_2	h_1	h_2	y
0	0	0	0	0
1	0	1	0	1
0	1	1	0	1
1	1	2	1	0

The power of NN

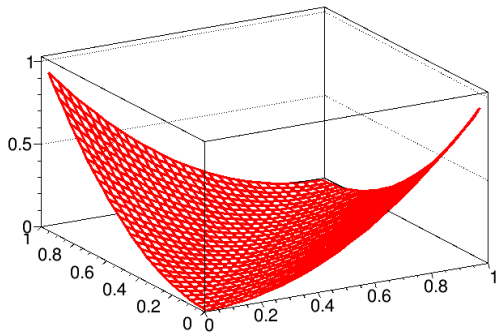
- Can NN solve any problem?
  - yes
- Universal approximation theorem (G. Cybenko): arbitrary decision regions can be arbitrarily well approximated by continuous feedforward neural networks with a single hidden layer and any continuous sigmoidal nonlinearity
  - namely, if σ is a "sigmoid" ($\lim_{t\to-\infty}\sigma(t)=0$, $\lim_{t\to\infty}\sigma(t)=1$), then for any continuous function $f(\mathbf{x})$ defined on a unit cube, for any ε there is a sum of form

$$G(\mathbf{x})=\sum_{j=1}^{N}\alpha_j\sigma(\mathbf{y}_j^{\mathrm{T}}\mathbf{x}+\theta_j)$$

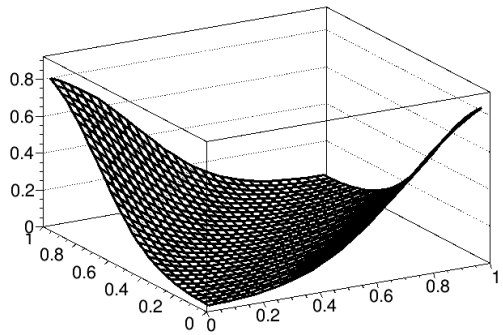
    such that $|G(\mathbf{x})-f(\mathbf{x})|<\varepsilon$ for all $\mathbf{x}$
  - $\sigma(t)$ does not have to be a sigmoid, it just has to satisfy certain conditions

NN training

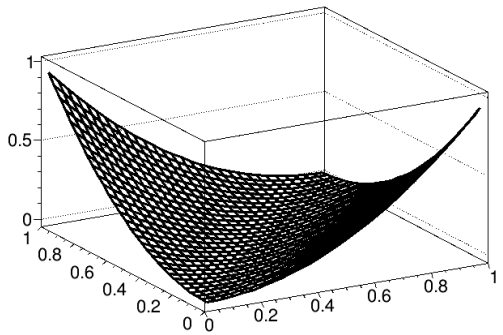
- For a simple NN with one hidden layer of n neurons, the number of parameters is small, and they can be found by a universal minimizer (like Minuit)
  - example: use the formula for $G(\mathbf{x})$ to approximate $f(x_1, x_2) = (x_1 - x_2)^2$ (takes care of the XOR problem)
  - for two inputs, one output, and n neurons in the hidden layer, the number of parameters is $4n$
  - let $\sigma(t) = 1/(1 + \exp(-t))$



target function



NN, $n = 2$



NN, $n = 4$

- In practice, large NNs are trained using iterative gradient-based optimizing procedures