Application of Optimization to Machine learning

Look at artificial neural networks (ANNs)

Aim: N measurements

$\underline{x}_1$ ⋮ $\underline{x}_N$ } Inputs

$\underline{y}_1$ ⋮ $\underline{y}_N$ } Outputs

Unknown functional relationship between inputs and outputs. Aim of ML is to "learn" the functional relationship.

What this means: A general input $\underline{x}$, a general input $\underline{y}$. Approximate the functional relationship as

$$\underline{y} \approx f(\underline{x}, \underline{p})$$

Where the $\underline{p}$'s are adjustible parameters and f is a function to be determined.

NEW NOTATION FOR THIS PART!

- $\underline{p}$ for parameters
- f for function TBC (not cost function!)

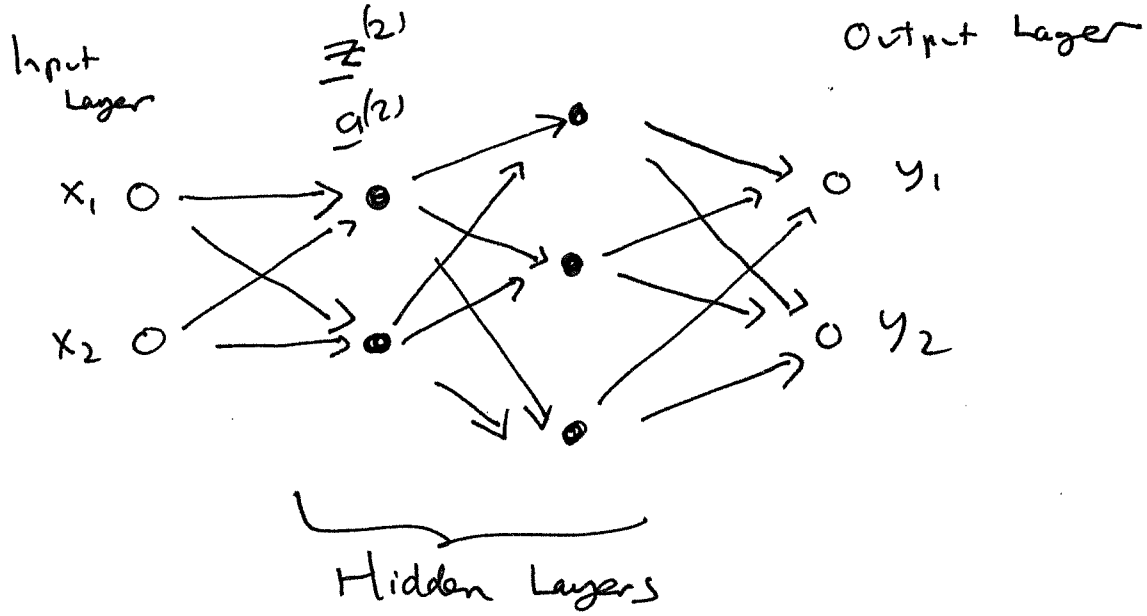
In Artificial Neural Networks (ANNs) $f(\underline{x}, \underline{p})$ is approximated by a composition of:

- Activation functions
- Linear functions — described by their weights and biases

$\underline{p}$

The idea is inspired by how the human brain works — a network of nodes which are either on or off — neurons ~~of connected~~ and connections between nodes — synapses. The first ANN was created by Frank Rosenblatt in 1957 and was implemented on an IBM 704 machine. He called his ANN the "perceptron".

Example: A small neural network:



- The input layer: $\underline{a}^{(1)} = \underline{x}$
- Gets transformed according to
  $\underline{z}^{(2)} = W^{(2)} \underline{a}^{(1)} + \underline{b}^{(2)}$
- Gets activated according to:
  $\underline{a}^{(2)} = \sigma(\underline{z}^{(2)})$ POINTWISE
- Gets transformed:
  $\underline{z}^{(3)} = W^{(3)} \underline{a}^{(2)} + \underline{b}^{(3)}$
- Gets activated:
  $\underline{a}^{(3)} = \sigma(\underline{z}^{(3)})$ ⟶

- Gets transformed to the output layer:

$$\underline{z}^{(4)} = W^{(4)} \underline{a}^{(3)} + \underline{b}^{(4)}$$

- Gets activated:

$$\underline{a}^{(4)} = \sigma(\underline{z}^{(4)}) \simeq \underline{y}$$

Write as a composition of functions:

$$\underline{y} = f(\underline{x}) \simeq \sigma\left(W^{(4)} \underline{a}^{(3)} + \underline{b}^{(4)}\right)$$

$$= \sigma\left[W^{(4)} \sigma\left(W^{(3)} \underline{a}^{(2)} + \underline{b}^{(3)}\right) + \underline{b}^{(4)}\right]$$

$$= \sigma\left\{W^{(4)} \sigma\left[W^{(3)} \sigma\left(W^{(2)} \underline{a}^{(1)} + \underline{b}^{(2)}\right) + \underline{b}^{(3)}\right] + \underline{b}^{(4)}\right\}$$

$$= \sigma\left\{W^{(4)} \sigma\left[W^{(3)} \sigma\left(W^{(2)} \underline{x} + \underline{b}^{(2)}\right) + \underline{b}^{(3)}\right] + \underline{b}^{(4)}\right\}$$

Parameters: $p = \{W^{(2)}, W^{(3)}, W^{(4)}, \underline{b}^{(2)}, \underline{b}^{(3)}, \underline{b}^{(4)}\}$

Counting:

- $\underline{z}^{(n)}$

  $\underline{z}^{(2)} \in \mathbb{R}^2$, $\underline{z}^{(3)} \in \mathbb{R}^3$, $\underline{z}^{(4)} \in \mathbb{R}^2$

- $W^{(n)}$

  $W^{(2)} \in \mathbb{R}^{2\times 3}$, $W^{(3)} \in \mathbb{R}^{3\times 2}$, $W^{(4)} \in \mathbb{R}^{2\times 2}$ 16

- $\underline{b}^{(n)}$

  $\underline{b}^{(2)} \in \mathbb{R}^2$, $\underline{b}^{(3)} \in \mathbb{R}^3$, $\underline{b}^{(4)} \in \mathbb{R}^2$ 7

23 parameters.

Training data: $\begin{cases} \underline{x}_1, \ldots, \underline{x}_N \\ \underline{y}_1, \ldots, \underline{y}_N \end{cases}$

Cost function ("loss function"):

$$C(p) = \sum_{i=1}^{N} \frac{1}{2} \| f(\underline{x}_i, p) - \underline{y}_i \|_2^2$$

$$= \sum_{i=1}^{N} C_i(p)$$

Steepest Descent:

$$p^{I+1} = p^I - \eta \nabla C(p^I)$$

Problem is computational cost of computing gradient (23 dimensions $\times$ N = # observations)

Instead, use stochastic gradient descent:

- At iteration I, choose $i \in \{1, 2, \ldots, N\}$ at random.
- Update p^I according to:

$$p^{I+1} = p^I - \eta \nabla_p C_i(p^I).$$

Back-propagation : Nature of approximating $f \hat{=}$ (compositions) means that the gradients $\nabla_p C_i$ can be computed ANALYTICALLY, via back-propagation.

For counter = 1 up to Niter

Stochastic gradient descent

Choose an integer k uniformly at random from $\{1, 2, \ldots, N\}$

$\underline{x}^{\{k\}}$ is set as the current training point.

Set $\underline{a}^{(1)} = \underline{x}^{\{k\}}$.

Feed forward steps :

For $\ell = 2$ up to L # ℓ and L label layers

$$\underline{z}^{(\ell)} = W^{(\ell)} \underline{a}^{(\ell-1)} + \underline{b}^{(\ell)}$$

$$\underline{a}^{(\ell)} = \sigma(\underline{z}^{(\ell)})$$

$$D^{(\ell)} = \mathrm{diag}\left(\sigma'(\underline{z}^{(\ell)})\right)$$

end

Back propagation steps :

Initialize with $\delta^{(L)} = \partial C / \partial z^{(L)}$:

$$\delta^{(L)} = D^{(L)}\left(\underline{a}^{(L)} - \underline{y}^{\{k\}}\right)$$

For $\ell = L-1$ down to 2

$$\delta^{(\ell)} = D^{(\ell)}\left(W^{(\ell+1)}\right)^T \delta^{(\ell+1)}$$

end

Here, $\delta^{(\ell)} = \partial C / \partial z^{(\ell)}$. ⟶

Steepest-descent update

For $l = L$ down to 2

$$W^{(l)} \longrightarrow W^{(l)} - \eta\, \delta^{(l)} \left(a^{(l-1)}\right)^T$$

Here: $\partial C / \partial W^{(l)} = \delta^{(l)} \left(a^{(l-1)}\right)^T$,

outer product

Also, $\partial C / \partial b^{(l)} = \delta^{(l)}$:

$$b^{(l)} \longrightarrow b^{(l)} - \eta\, \delta^{(l)}$$

end

end

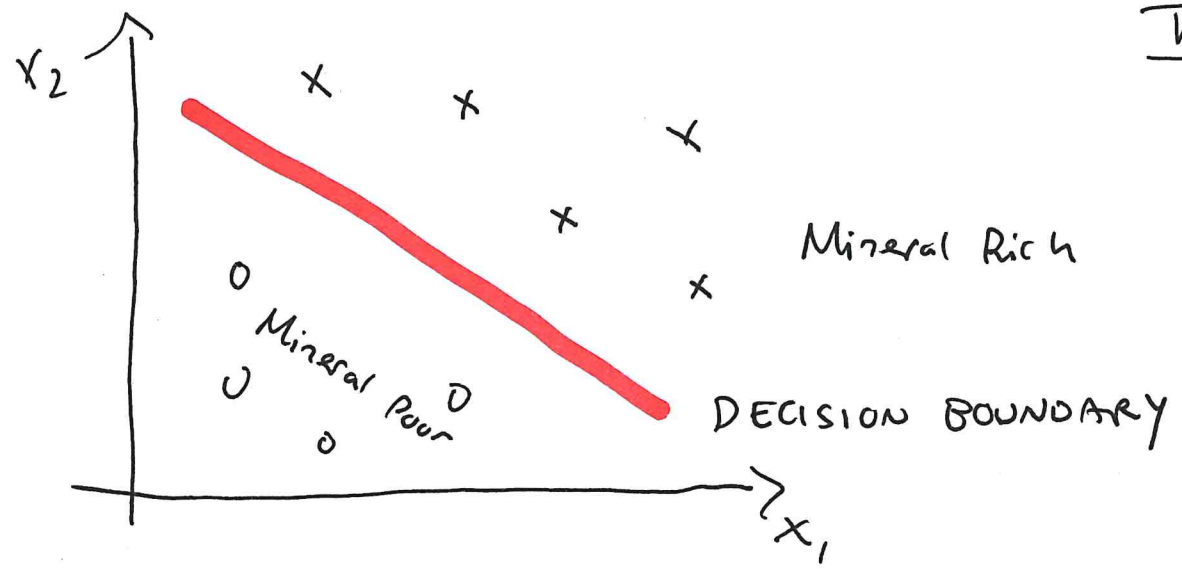
Example:

$$\begin{cases} \underline{x}_1, \ldots, \underline{x}_N = \text{points in } \mathbb{R}^2 \\ y_1, \ldots, y_n \in [0,1] \end{cases}$$

Could be samples from various places where:

- $\underline{x}_i$ are latitude / longitude
- $y_i = \begin{cases} 1 \text{ if } x_i \text{ is a mineral-rich region} \\ 0 \text{ if } x_i \text{ is mineral poor} \end{cases}$

We are looking for the decision boundary separating mineral-poor and mineral-rich regions. This then tells us where to explore for minerals. ⟶



Instead of writing our own code, we have used MATLAB's feedforwardnet function to create an ANN with two hidden layers:

- First hidden layer has 2 nodes
- Second " " " 3 nodes

Structure of input / output layers set by training data:

- Input layer has 2 nodes (x_1, x_2)
- Output layer has 1 node (y)

- ANN is supplied with synthetic training data
- ANN is trained using train function in Matlab
- Result is plotted and decision boundary is constructed as the 0.5 contour of $f(x, p)$.

LINK TO MATLAB CODE

DEMO ON LAPTOP

Summary

- Whistlestop tour of Continuous optimization
- Described problem of training an artificial neural network in mathematical terms
- Showed how an ANN can be used to construct a decision boundary.

Key problems in ANN

- Non-convex optimization problem — not clear if global min. is reached
- Understanding best network architecture (# nodes, # layers) for the problem in hand
- Understanding the best choice of activation function for each layer
- Understanding which cost function is best for the problem in hand
- Requires good knowledge of Mathematics to be a responsible user of ANNs.