Supervised learning

Overview: What is Supervised learning?

- The aim is to develop a mathematical model that takes in an input $\vec{x}$ and returns an output $\vec{x}$
- The model is simply a mathematical equation with a fixed form. $y = f[x]$
- Inference: process of making predictions using the model.
- The model $y = f[x]$ represents a family of relations b/w the i/p and o/p. We also add parameters to the model to determine a particular relation, so our model is $y = f[x, \phi]$ where ϕ are the model parameters.
- we want to find a model that makes the most sensible mapping from i/p to o/p. This is called Learning or training a model.
  - What do we want to learn?: The parameters
  - How do we learn?: with the help of the training data set I, pairs of i/p & o/p $\{\vec{x}_i, \vec{y}_i\}$. Thus the name supervised. We try to map the training i/p to its associated o/p as closely as possible.
  - we need a way to quantify the mismatch in this mapping. This is using a loss function. selection of a param ϕ will result in a "loss" value. which quantifies how bad the mapping is. So the loss is a fN. of the parameters.
  - So, to train/learn a model is to find parameters that minimizes the loss function. or. $\hat{\phi} = \underset{\phi}{\operatorname{argmin}}[L[\phi]]$
- basic method is to choose a random set of param values then "walk down" the loss fN to reach the bottom.

- Once trained, we want to see how well the model performs on novel data [test data] and how well it generalizes.
  - It's possible that the model might underfit: model does not capture the true mapping or overfit: Model learns the pecularities of the training set

Example: Linear Regression.

- 1D linear regression model. (i.e scaler x y), a straight line
- $y = f[x, \phi]$
  $= \phi_0 + \phi_1 x$

y intercept of the line (parameter) — ϕ_0; (Parameter) Slope of the line — ϕ_1

This eqn describes the family of all possible lines, we want to find the parameters that best fit our training data.

- Loss function: (remember, it assigns a num. value to each param selection indicating how bad the mismatch is in that mapping)
  - lower loss → better fit.
- One way to calculate this is to find out the deviation b/w the prediction made by the model (i.e $f[x_i, \phi]$) for an input and its ground truth (i.e. y_i) [for the whole training data set]

or $L[\phi] = \sum_{i=1}^{I} (f[x_i, \phi] - y_i)^2$ ← squaring makes the direction of deviation unimportant.

[this is the Square loss].

To find the best param, we want to minimize the squaredloss

$$\hat{\phi} = \underset{\phi}{\operatorname{argmin}}[L[\phi]]$$
$$= \underset{\phi}{\operatorname{argmin}}\left[\sum_{i=1}^{I} (f[x_i, \phi] - y_i)^2\right]$$
$$= \underset{\phi}{\operatorname{argmin}}\left[\sum_{i=1}^{I} (\phi_0 + \phi_1 x_i - y_i)^2\right]$$

Shallow Neural Networks

What is a neural network and why do we need one?

Linear regression give a linear relationship b/w x & y which is not always true in real life. Neural N. describes piecewise linear functions which are much more expressive and can approximate complex relation b/w x & y.

Intuition behind Neural N/w using an example

Consider a model $y = f[x, \phi]$ described by 10 parameters.

$\phi = \{\phi_0, \phi_1, \phi_2, \phi_3, \theta_{10}, \theta_{11}, \theta_{20}, \theta_{21}, \theta_{30}, \theta_{31}\}$

The model is described as:

$$y = \phi_0 + \phi_1\, a(\theta_{10} + \theta_{11}x) + \phi_2\, a(\theta_{20} + \theta_{21}x) + \phi_3\, a(\theta_{30} + \theta_{31}x)$$

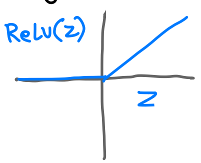
notice that we are doing 3 steps here:

1. We compute 3 linear functions: $\theta_{10} + \theta_{11}x$, $\theta_{20} + \theta_{21}x$, $\theta_{30} + \theta_{31}x$
2. We pass the result through a function a called the activation fn $a[\bullet]$
3. Add weights to the resulting activation and add an offset ϕ_0 (ϕ_1, ϕ_2, ϕ_3)

Activation fn: many types. Most common: Retified Linear Unit (ReLU)

$$a[z] = \begin{cases} 0 & \text{if } z < 0 \\ z & \text{if } z \geq 0 \end{cases}$$

(Clips -ve values to 0)



Which family of eq^n does this model belong to?

A family of continious piecewise linear fns with upto 4 linear regions.

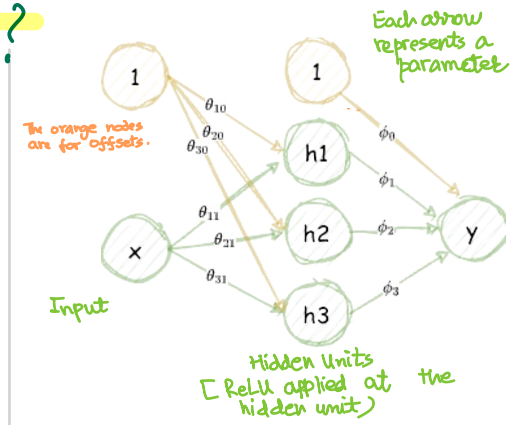
Another way to write this eq:

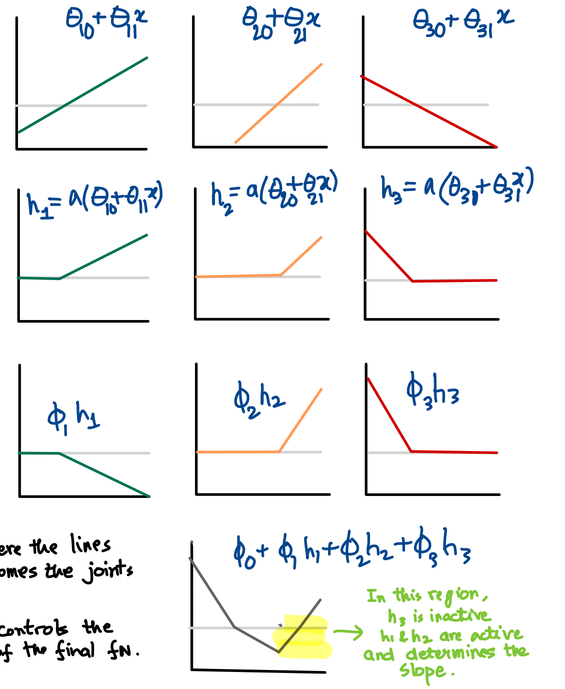
let $h_1 = a(\theta_{10} + \theta_{11}x)$, $h_2 = a(\theta_{20} + \theta_{21}x)$, $h_3 = a(\theta_{30} + \theta_{31}x)$

then $y = \phi_0 + \phi_1 h_1 + \phi_2 h_2 + \phi_3 h_3$.

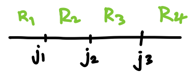
lets call h_1, h_2, h_3 "hidden layers". To compute y, we first compute the hidden layers, then combine their result with a linear fn.

This flow of computation can be described in a network. nodes are in input & hidden layer and edges are the parameters.





- In each region a hidden layer is either clipped (inactive) or present (active)
- Each hidden layer introduces one joint ⇒ there are 4 linear regions
- The positions where the lines crosses zero becomes the joints in the final o/p.
- The offset ϕ_0 controls the overall height of the final fn.



How can neural networks describe complex relations? (The Universal approximation theorem)

A general SHALLOW Neural Network with D hidden Layers:

- A layer i is defined by $h_i = a[\theta_{i0} + \theta_{i1}x]$
- All d layers combined, gives the o/p

$$y = \phi_0 + \sum_{d=1}^{D} \phi_i h_i$$

- D is the capacity of the N/W
- The piecewise fn has at most D joints (& D+1 linear regions)
- As more layers are added, complexity increases

A shallow n/w can describe any continious 1D function.

As we add more and more hidden layers, we add more and more smaller linear regions that represent a part of the fn increasigly well approximated by a line.

This can be proven using the universal approximation theorem.

Shallow networks for Multi Variate data

i.e. when input is $\vec{x} = [x_1, x_2 \dots x_i]^T$ & o/p is $\vec{y} = [y_1, y_2 \dots y_j]^T$

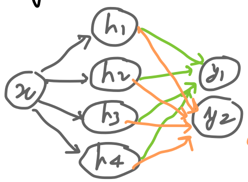
let's consider them individually:

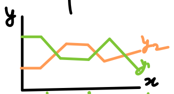
Multivariable output : i.e x; $y = [y_1, y_2]^T$: Each output is a different linear function of the hidden layers.

Example : 4 hidden layers :

$h_1 = a(\theta_{10} + \theta_{11}x)$, $h_2 = a[\theta_{20} + \theta_{21}x]$

$h_3 = a(\theta_{30} + \theta_{31}x)$, $h_4 = a(\theta_{40} + \theta_{41}x)$





$y_1 = \phi_{10} + \phi_{11}h_1 + \phi_{12}h_2 + \phi_{13}h_3 + \phi_{14}h_4$

$y_2 = \phi_{20} + \phi_{21}h_1 + \phi_{22}h_2 + \phi_{23}h_3 + \phi_{24}h_4$

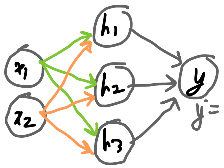
Multivariate input : i.e $x = [x_1, x_2 \dots]$: extend the relation b/w i/p & hidden layers.

Example : 3 hidden layers, $x = [x_1, x_2]$, y is scaler.

$h_1 = a[\theta_{10} + \theta_{11}x_1 + \theta_{12}x_2]$

$h_2 = a[\theta_{20} + \theta_{21}x_1 + \theta_{22}x_2]$

$h_3 = a[\theta_{30} + \theta_{31}x_1 + \theta_{32}x_2]$



$y = \phi_0 + \phi_1 h_1 + \phi_2 h_2 + \phi_3 h_3$

In this case the input is 2D i.e a plance, the activation fn clips the negative part of the plane which are combined as continious linear piecewise planes (convex polygon regions)

Shallow neural n/w : general case

$\vec{y} = f[\vec{x}, \Phi]$, $\vec{x} \in R^i$, $\vec{y} \in R^j$ & D hidden layers

$$\text{hidden layer } d = h_d = a\left[\theta_{d0} + \sum_{t=1}^{i} \theta_{dt} \cdot x_t\right]$$

$$\text{and each output } y_j = \phi_{j0} + \sum_{t=1}^{D} \phi_{jt} h_t$$

The model has an activation fn a and parameter set ϕ
most common being ReLU

The network divides the i/p space into convex polytopes defined by the intersection of hyper planes computed by the "joints" in the ReLU functions.

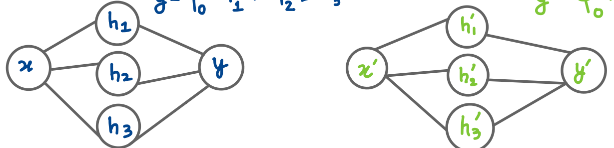
- pre activations : the value of inputs to hidden layers before ReLU functions are applied [ie $\theta_{10} + \theta_{11}x_1 + \theta_{12}x_2$]
- activations : value at the hidden layer.
- n/w with atleast 1 hidden layer are called multilevel perceptron.
- n/p with 1 hidden layer : Shallow N.N ;
  with multiple hidden layers : Deep N.N
- Feed forward n/w : N.N where the connections form an acyclic graph.
- Fully connected : when all elements of one layer connects to all the elements of the next layer.
- Connections represent slope parameters or Network weights the offset parameters (usually not shown in the n/w) are called biases.

Deep Neural Networks

Composing shallow N.N into Deep N.N

- o/p of 1 as i/p of 2nd
- Eg:

$y = \phi_0 + \phi_1 h_1 + \phi_2 h_2 + \phi_3 h_3$ $\quad$ $y' = \phi'_0 + \phi'_1 h'_1 + \phi'_2 h'_2 + \phi'_3 h'_3$

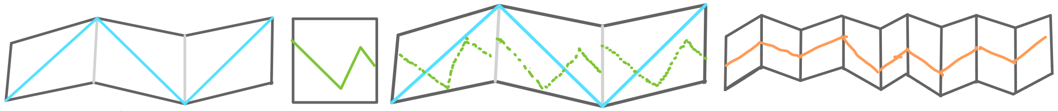


Q_1: when o/p of one is fed as i/p to other, how many region exists in the piecewise fn.

Q_2: How does composing N/w compare to a shallow N/w with the same total hidden units.

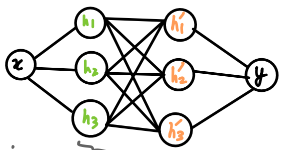
Composing S NN results in more linear regions. Consider a/w 1 with 3 regions. when composed with another similar N/w, each of the regions in the first N/w are further divided into 3 more region. resulting in a total of 9 region [compare this with N/w with 6 hidden layers which can give atmost 7]

Another way to think of it is that the 1st n/w folds the input space into 3 regions. The 2nd n/w further folds each fold.



n.w 1. $\quad$ nw 2 $\quad$ composing N/w 1 & 2

This is equivalent to a Deep Network with 2 hidden layers.



The result is again a piecewise continuous linear fn.

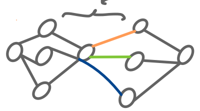
$h_1 = a(\theta_{10} + \theta_{11} x)$
$h_2 = a(\theta_{20} + \theta_{21} x)$
$h_3 = a(\theta_{30} + \theta_{31} x)$
$l_1 = \phi_0 + \phi_1 h_1 + \phi_2 h_2 + \phi_3 h_3$

$h'_1 = a(\alpha_{01} + \alpha_{11} l_1)$
$h'_2 = a(\alpha_{02} + \alpha_{12} l_1)$
$h'_3 = a(\alpha_{03} + \alpha_{13} l_2)$

$y = \psi_0 + \psi_1 h'_1 + \psi_2 h'_2 + \psi_3 h'_3$

(expand the full thing) (it's still linear)

Equivalent to:



Pre activations of all nodes of this layer are same.

Generalizing this,

Hyperparameters:

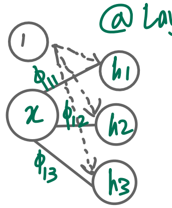
- width: # of hidden units in each layer
- depth: # of hidden layers
- Capacity: total # of hidden units in the N/w
- These are called hyper.p. These values are chosen before we learn the model parameters.

- K = # Layers, $D_1, D_2 \dots D_K$: Nodes in each layer.

Matrix Notation:

$$\vec{h} = a[\vec{\theta}_0 + \vec{\theta}_1 \vec{x}]$$
$$\vec{h}' = a[\vec{\psi}_0 + \vec{\psi}_1 \vec{h}]$$
$$\vec{y} = \vec{\phi}'_0 + \vec{\phi}'_1 h'$$

@ Layer 1

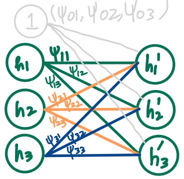


$$\begin{bmatrix} h_1 \\ h_2 \\ h_3 \end{bmatrix} = a\left(\begin{bmatrix} \phi_0 \\ \phi_1 \\ \phi_2 \end{bmatrix} + x \begin{bmatrix} \phi_{11} \\ \phi_{12} \\ \phi_{13} \end{bmatrix}\right)$$

$$\vec{h} = a(\vec{\theta}_0 + x \vec{\theta}_1)$$

@ Layer 2

(Parameter names: $\psi_{\text{to-from}}$)



$h'_1 = a(\psi_{01} + \psi_{11} h_1 + \psi_{12} h_2 + \psi_{13} h_3)$
$h'_2 = a(\psi_{02} + \psi_{12} h_1 + \psi_{22} h_2 + \psi_{23} h_3)$
$h'_3 = a(\psi_{03} + \psi_{13} h_1 + \psi_{32} h_2 + \psi_{33} h_3)$ $\Rightarrow$

$$\begin{bmatrix} h'_1 \\ h'_2 \\ h'_3 \end{bmatrix} = a\left(\begin{bmatrix} \psi_{01} \\ \psi_{02} \\ \psi_{03} \end{bmatrix} + \begin{bmatrix} \psi_{11} & \psi_{21} & \psi_{31} \\ \psi_{12} & \psi_{22} & \psi_{23} \\ \psi_{31} & \psi_{32} & \psi_{33} \end{bmatrix}\begin{bmatrix} h_1 \\ h_2 \\ h_3 \end{bmatrix}\right)$$

$$\vec{h}' = a(\vec{\psi}_0 + \vec{\psi} h)$$

In general, $h_1 = a(\beta_0 + \Omega_0 x)$
$h_2 = a(\beta_1 + \Omega_1 h_1) \dots$
$h_K = a(\beta_{K-1} + \Omega_{K-1} h_{K-1})$ and $\vec{y} = \beta_K + \Omega_K h_K$

Parameters θ are all the weights & Biases $\phi = \{\beta_k, \Omega_k\}_{k=0}^{K}$

- if K^{th} layer has D_K units, Bias vector β_{K-1} will be of size D_K
- size of $\beta_K = D_0$ (the output)
- Size of $\Omega_K = D_K \times D_{K+1}$, except for last $\Omega_K (D_0 \times D_K)$ and first : Ω_0 : $D_0 \times D_i$ (size of input)

Shallow vs Deep N.N.

Property	Shallow	Deep
Ability to approx diff. fns	can approx well with enough capacity	Can approximate any continious fn arbitraly closely (given sufficient capacity)
No. of linear regions per parameter	Given 1 i/p, 1 o/p & D>2 hidden units, will have atmost D+1 linear regions Defined by 3D+1 params.	Given 1 i/p, 1 o/p, K layers with D>2 units, $(D+1)^K$ linear regions, $3D+1+(K-1)D(D+1)$ parameters. Linear Regions increases as the # of params increases ⇒ For a fixed param budget D.NN creats more complex fns
Depth Efficiency		Some fn are require exponentially more hidden units shallow N.N than equivalent Deep N/W (This is called Depth Eff. of D.N.N)
Large structured i/p :		fully connected N/W may not be very practical. or the # of parameters may be prohibitive. Helps in local-to-global processing (more later!)
Training & Generalization		- easier to train moderately deep NN. (difficult for over parameterized ones) - seem to generalize new data better.