Sparse Autoencoders → unsupervised learning ← convert to ← Neural Networks → Supervised

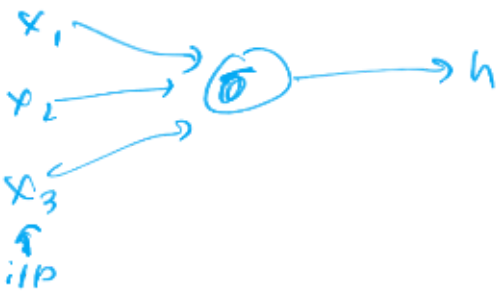
Neural NW:-

$(x^{(i)}, y^{(i)})$

Hypothesis: $h_{w,b}(x)$

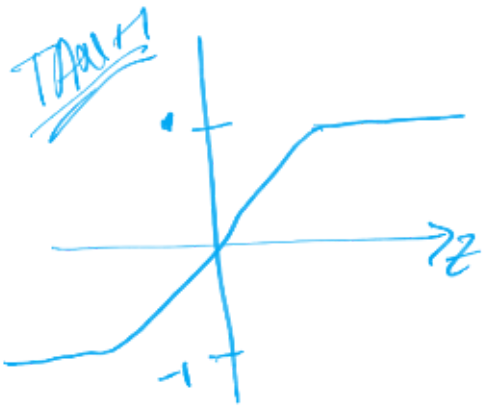
parameters: w, b

"Neuron"



f = activation function

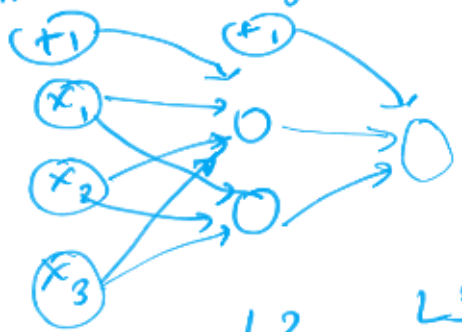
$f(z) = \frac{1}{1+e^{-z}}$ ⇒ sigmoid AF.



$f(z) = \tanh(z)$

$= \frac{e^z - e^{-z}}{e^z + e^{-z}}$

Combine lots of these neurons to make a neural network



"+1" ⇒ bias units 'b'

L1 aka i/p layer — L2 hidden layer — L3 aka o/p layer

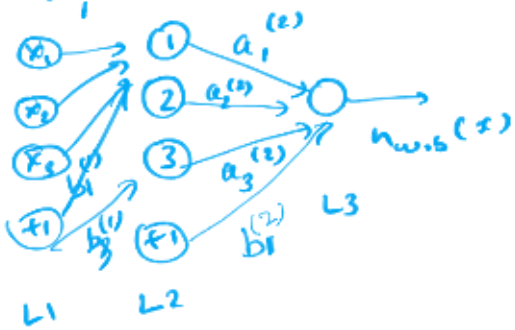
5

parameters (W, b)

$W_{ij}^{(l)}$ = weight for connection b/w unit j in layer l and unit i in layer $l+1$

$b_i^{(l)}$ = bias associated with unit i in layer $l+1$

$a_i^{(l)}$ = activation (O/P value) of unit i in layer l



$$a_1^{(2)} = f(W_{11}^{(1)} x_1 + W_{12}^{(1)} x_2 + W_{13}^{(1)} x_3 + b_1^{(1)})$$

$$a_2^{(2)} = f(W_{21}^{(1)} x_1 + W_{22}^{(1)} x_2 + W_{23}^{(1)} x_3 + b_2^{(1)})$$

$$a_3^{(2)} = f(\ldots\ldots)$$

$$h_{W,b}(x) = f(W_{11}^{(2)} a_1^{(2)} + W_{12}^{(2)} a_2^{(2)} + W_{13}^{(2)} a_3^{(2)} + b_1^{(2)})$$

$h_{W,b}(x) \Rightarrow$ Parameters W, b

Online learning :- $(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}) \cdots$ ↑ no end

stochastic gradient descent :-

For $i = 1, 2, 3, \ldots$

Get example $(x^{(i)}, y^{(i)})$

update $W_{jk}^{(l)} := W_{jk}^{(l)} - \alpha \frac{\partial}{\partial W_{jk}^{(l)}} J(W, b; x^{(i)}, y^{(i)})$

$b_j^{(l)} := b_j^{(l)} - \alpha [\ldots]$

$b_j^{(l)} := b_j^{(l)}$

Cost Function :- $J(W, b; x, y)$

$= \frac{1}{2} \| h_{W,b}(x) - y \|^2 + \frac{\lambda}{2} \sum_l \sum_i \sum_j \left(w_{ji}^{(l)} \right)^2$

$\| h_{W,b}(x) - y \|^2$ → squared error

$\sum_l \sum_i \sum_j \left(w_{ji}^{(l)} \right)^2$ → weighted decay → regularization term to prevent overfitting

λ = weight decay parameter

If $(x^{(i)}, y^{(i)}) \sim D$ (IID) ↳ Identically independently distributed.

$\min_{W,b} E[J(W, b; x, y)]$

For a NN it is NOT ok to initialize all parameters with zeroes

So instead we initialize with random values

$w_{ij}^{(l)} \sim N(0, \varepsilon^2)$ {eg} ↳ Normal distribution

& not all $w_{ij}^{(l)} = 0$

bcoz if we do so, all hidden units will be computing exactly the same function & in each iteration $a_1^{(2)} = a_2^{(2)} = a_3^{(2)} = \cdots$

i.e., .·. weights randomly for symmetry Breaking

So we init

Bias values can be initialized with 0s or randomly.

Backpropagation :-

Intuition: For node i in layer l
compute $\delta_i^{(l)}$ that measure how responsible that node is for errors in o/p.

⇒ For o/p node :-
(In layer n_l)
compute $\delta_i^{(l)}$ as a function of $h_{w,b}(x)$ & y

⇒ For hidden units :-
(can't observe the o/p directly)

$$\delta_i^{(l)} = \left(\sum_j w_{ji}^{(l)} \delta_j^{(l+1)} \right) \cdot (\dots)$$

Algorithm :-

(1) Feedforward pass to compute activations of all units

(2) For each unit in o/p layer (layer n_l)
compute $\delta_i^{(n_l)} = -(y_i - a_i^{(n_l)}) \cdot f'(z_i^{(n_l)})$

(3) For each layer $l = n_l - 1, n_l - 2, \dots 2$

$$\delta_i^{(l)} = \left(\sum_j w_{ji}^{(l)} \delta_j^{(l+1)} \right) \cdot f'(z_i^{(l)})$$

(4) [illegible]

(4) Update parameters :-

$$W_{ij}^{(l)} := W_{ij}^{(l)} - \alpha \left(a_j^{(l)} \delta_i^{(l+1)} + \lambda W_{ij}^{(l)} \right)$$

$$b_i^{(l)} := b_i^{(l)} - \alpha \delta_i^{(l+1)}$$

Unsupervised learning :- → unlabelled examples ⇒ No $y^{(i)}$, only $x^{(i)}$ part given

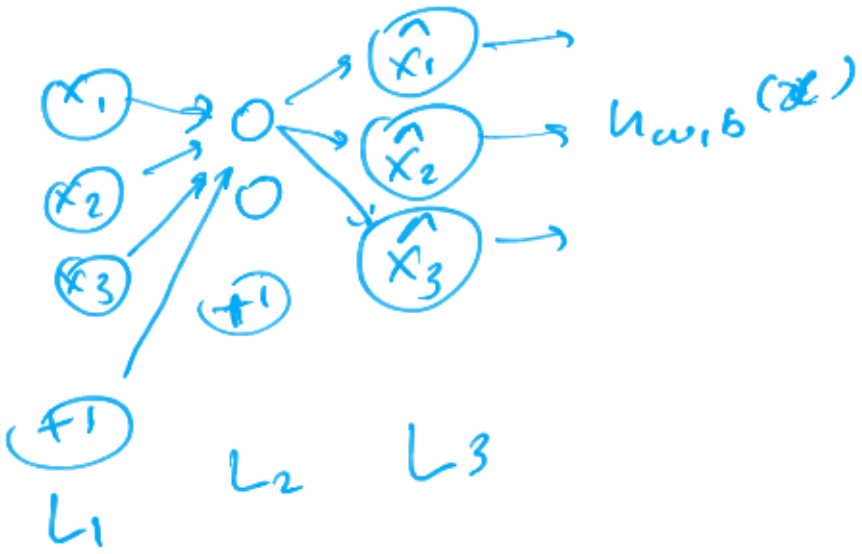
$x^{(1)}, x^{(2)}, x^{(3)}, \ldots$

create $y^{(i)} = x^{(i)}$ so that we can give it to NN.

ans: NN need labelled examples.

Autoencoder :- → learn identity function.

$h_{W,b}(x) \approx x$ → approximation to ID function.



eg: i/p 10×10 image

⇒ $x \in \mathbb{R}^{100}$

50 hidden units (say)

100 O/P units

In a real image all pixels are not completely independent (i.e., not i.i.d.)

... to compress 100 pixels image do a

If we draw 50 ...

50 units hidden layer then we well be (~~100~~)
learning many (redundancies), so we make

in 50 units ... not a meaningful compression.

→ limiting the ~~size~~ hidden layer is one way to force autoencoders to learn an interesting function.

→ another constraint ⇒ sparsity

Sparsity :- (Informally)

constraint $a_i^{(2)} \approx -1 \ \forall i$

Neuron is "active" if $a_k^{(2)} \approx +1$

"inactive" if $a_i^{(2)} \approx -1$

⇒ most neurons should be inactive most of the times.

Sparsity (Formal defn)

Imagine $x^{(1)}, x^{(2)}, x^{(3)}, \dots$ are drawn IID from $\mathcal{D}$

constraint : $E_{x\sim\mathcal{D}}[a_i^{(2)}] = \rho \quad \forall i$

where $\rho = -0.9$ (say) {i.e., close to -1 }

Algo :-

(1) keep running estimate of $\hat{\rho}_i$ of $E_{x\sim\mathcal{D}}[a_i^{(2)}]$

(2) On each iteration of gradient descent, update parameters to make expectation closer to $\rho = -0.9$ (say)

Explanation :-

b''

(1) On each iteration, update

$$\hat{\rho}_i := 0.999 \cdot \hat{\rho}_i + 0.001 \cdot a_i^{(2)} \quad \{\text{for each } i\} \rightarrow (\text{eq 1})$$

Initialize $\hat{\rho}_i = 0$ (say)

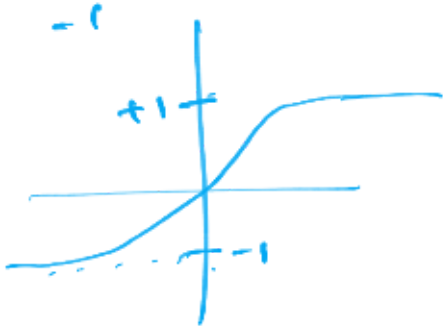
(2) Recall $a_i^{(2)} = f\left(\sum_{j=1}^{n} w_{ij}^{(1)} x_j + b_i^{(1)}\right)$

If $\hat{\rho}_i > \rho$ ⇒ we want hidden unit i to be less active (want o/p values closer to -1)

eg $(-0.7) > (-0.9)$

So decrease $b_i^{(1)}$

bcoz decreasing $b_i^{(1)}$ will move value of 'f' closer to -1



If $\hat{\rho}_i < \rho$

↓

we want hidden unit i to be more active

So increase $b_i^{(1)}$

putting both cases together

update rule:

$$b_i^{(1)} := b_i^{(1)} - \alpha\beta(\hat{\rho}_i - \rho) \quad \forall i \rightarrow (\text{eq 2})$$

learning rate

additional learning rate

Summarised algo. for enforcing sparsity constraint

→ On each iteration (get new (x, y)):

(1) Run forward pass to compute all activations.

(2) Use backprop to perform 1 iteration of stochastic gradient descent.

(3) Use (eq 1) to update $\hat{p}_i$ and

(eq 2) to update $b_*^{(1)}$