

* Sigmoid neuron → building block of Deep Neural Networks.

↓
or the foundational block of DNN.

↓
not limited to sigmoid, can be something like ReLU neuron or any other non-linearity

↓
generalised case: sigmoid neuron.

↓
till now i/p's: real i/p's

task: biclassification

model: linear (can't operate with non-linear data)

↓
weights and bias were tuneable (perceptron)

indicator fn.
or 1-0 loss

← loss: $\sum_i 1_{\{y_i \neq \hat{y}_i\}}$ or MSE

training: perceptron learning algo.
or brute search (M.P)

evaluation: accuracy

Note: o/p plotted along with the features would result in a kind of a ramp function (for perceptron.

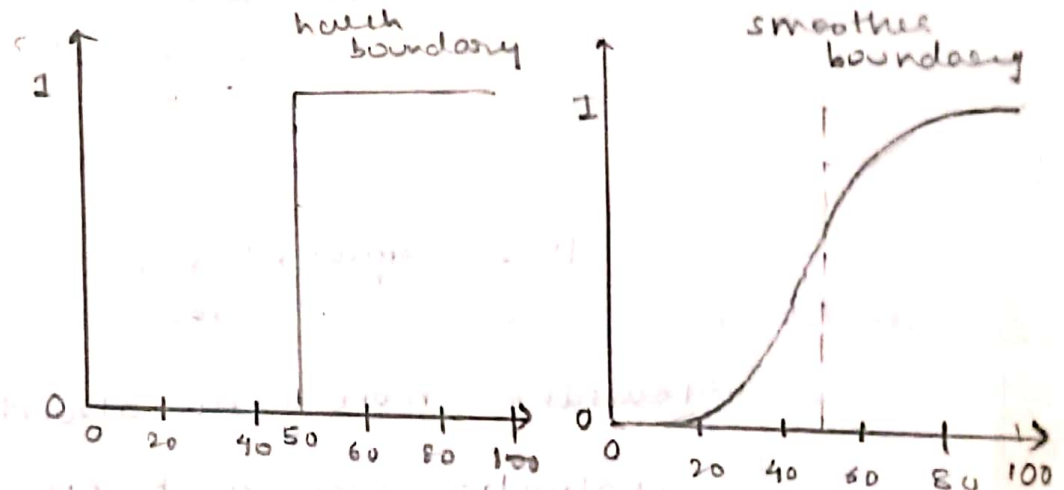
① if linearly separable)

↓
also the decision boundary is very sharp. for perceptron, rather practically the decisions are much smoother

•) a person with salary 50.1k can afford a car but 49.9k can't?

↓
boundary is harsh, and we would rather want a more smooth boundary, where there is some uncertainty associated with a range of i/ps.

salary	buy?
80	1
20	0
65	1
15	0
30	0
49	0
51	1
87	1



therefore person with a strategy near to 50k also has a prob. of being able to afford a car

the probability on the right of 50 is now ~~low~~ approaching 1 and prob. on the left is not equal to 0.

↓
which is logical practically.

↓
These probabilities are continuous, and hence can also be employed with regression ~~in some way~~, and are not just constrained to classification.

↓
real valued o/p.s.

for sigmoid neuron: i/p: real

task: classification/regression

↓
real o/p ✓

model: smooth at boundaries
non-linear

(can deal with non linear
fns to a limit, only first
step in that direction)

loss: squared error
loss

↓
network of
neuron

learning: more generic algorithm

evaluation: accuracy + RMSE

(root mean square
error)

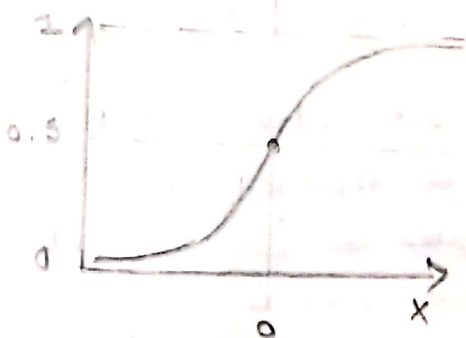
↓
since this would be
used for regression

parameters are $[w], b$

↑
the mapping b/w y and x

→ approximated using sigmoid

(*) model: a smoother fn. → sigmoid family of fns.



substitute values in fn. to
visualize.

$$y = \frac{1}{1 + e^{-(wx+b)}}$$

→ single feature

↓
logistic function

↓
always +ve

↓
between 0 + 1 (probability)

3. evaluate $\hat{\pi}(s, a)$ → single sample
 4. evaluate policy gradient
- $$\nabla_{\theta} J(\theta) \approx \nabla_{\theta} \log \pi_{\theta}(s, a) \cdot \hat{\pi}(s, a)$$

- *) hue / color can be used to visualize probability smoothness for two feature data, rather than plotting 3D graphs, (with certainty along the ends and uncertainty in the middle)
- *) The power of the exponential is the aggregation of the features with their weights.

$$y = \frac{1}{1 + e^{-(w^T x + b)}}$$

→ scalar quantity.

- *) In perceptron model, there is no distinction between the points very close to the decision boundary and the points very far to the decision boundary as the points far from the decision boundary are very sure to be classified correctly, but the ones close would / should still have some uncertainty associated with them.

↓

can be attributed to the harsh boundary and the o/p being either 1 or 0.

↓

The sigmoid neuron deals with this issue as it assigns a ~~higher to much~~ ~~probability to~~ ~~more~~ higher level of confidence to points far from the decision boundary as compared to the ones close to it.

- *) o/p of sigmoid can also be interpreted as probability.
- *) does not separate the i/p, rather giving a more graded o/p.

and we probably need to use parameter

* variation according to w and b for the model

$w \rightarrow -ve$, then the slope of the curve is $-ve$, as the denominator \uparrow , and total value decreases as $w \uparrow$

more $-ve$, more steeper slope of curve.

$+ve$, reverse happens

$$\frac{1}{1+e^{-w}} = \frac{1}{1+\frac{1}{e^w}}$$

$$\text{or } \frac{1}{1+e^w}$$

$$\frac{1}{1+\frac{1}{e^w}} > \frac{1}{1+e^w}$$

⑥ \rightarrow decreasing the value pushes the curve to the right

$$y = \frac{1}{1+e^{-(wx+b)}} = \left(\frac{1}{2}\right) \rightarrow \text{at the middle}$$

$$\Rightarrow e^{-(wx+b)} = 1$$

$$\Rightarrow wx+b=0$$

$$\Rightarrow x = \left(-\frac{b}{w}\right) \rightarrow \text{as } b \text{ decreases, with constant } w, x \uparrow \text{ (hence shift to the right)}$$

4. evaluate policy π

$$\pi(\theta) \approx \nabla_{\theta} \log \pi$$

$$15) \cdot \hat{\pi}(\theta) > 1$$

→ real x/p. feature

* Data and Task

(.) sigmoid neuron can be used for binary classification as well by highlighting which class the feature vector is more closer to. i.e. we are now getting a softer o/p.

↓
to take a decision, some threshold can be applied to these values. ~~to decide~~

(.) regression: given a feature vector, fitting a real value of y to that based on training data.

↓
for sigmoid, we are regressing the probability.

↓
could be used to o/p the rating for the phone (normalised btw 0 and 1)

* Loss function:

$$\sum_{i=1}^N (y - \hat{y})^2 \rightarrow \text{squared error loss fn.}$$

true prediction

↓
can also be used with binary labels.

↓
for this case the point closer to the labels contribute less to the loss whereas the points away from it contribute more.

* learning algorithm

objective: give the data, we want to generalise the mapping between x and y , using sigmoid function whose parameters are w and b

- i) initialize w, b
- ii) iterate over the dataset, and compute loss on predicted value

iii) update w and b , and repeat → the generalised roadmap.

w and b need to be updated and approximated in order to minimize the loss over the dataset.

a) method 1: guesswork, : initialize $w=0, b=0$,
(by visualizing the points and the function)

and accordingly taking the decisions to vary w and b .

but, yes not scalable to higher dimensions.

guesswork or random changes would take a time to actually approximate

straight line ($y=0.5$)

change $w=1, b=0$,
somewhat better (the slope)

going from $w=1$ to $w=3$
(slope more steeper)

moving the fn. to the right
(decreasing b)

again changing w and b
a little works.

in other words, for each iteration,
 $w = w + \Delta w$
 $b = b + \Delta b$

3. evaluate $\hat{\pi}(S, a)$
4. evaluate policy gradient $\nabla_{\theta} J(\theta) \approx \nabla_{\theta} \log \pi(S, a) \cdot \hat{\pi}(S, a)$

* for guesswork were we guided by the loss function?

yes in a ~~weak~~ way the guesswork was guided by the loss function, but it can only get us so far.

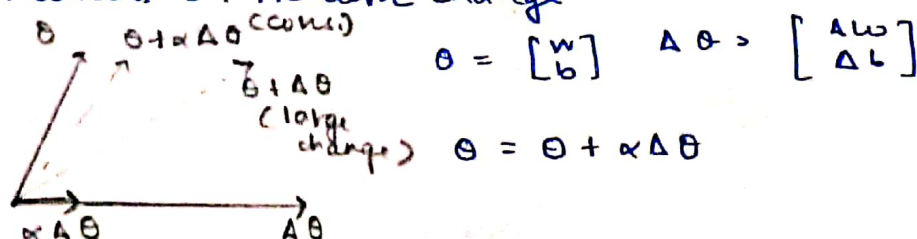
(it is easier when there are only 2 or 3 features, but for high dimensional data, this would be next to impossible to determine)

observation: Also after one point of decreasing the loss, the loss began to increase, and we had to move back a little to reach the optimal w and b for minimal loss.

for guesswork, the trajectory of optimizing w and b to minimize loss has been very random and full of fluctuations, and in high end dimensionality, where the guesswork will move in random directions and hence reaching the minima will require a large no. of steps.

so this motivates the need for a more principled algorithm that allows us to move such a way that there is gradual decrease in the loss associated with w and b .

b) method 2: iterative change



how to compute $\Delta \theta$?

change or diff. vector

gradient of loss with respect to θ

frameworks like TensorFlow and pyTorch
have functions that give these gradients
~~and~~ and these are used to tune θ
at each iteration.

* Taylor series .

$$f(x + \Delta x) = f(x) + f'(x) \Delta x + \frac{1}{2!} f''(x) \Delta x^2 + \frac{1}{3!} f'''(x) \Delta x^3 + \dots$$

↓
value in close proximity
to x