DL

RNN for speech recognition & machine translation

CNN for image & audio recognition etc.

structured data → datasets → easily organized in rows & columns

unstructured " → audio, images, text docs, emails, social media posts, etc.

- Relu over sigmoid, due to gradient descent performance being higher using relu func^n
It introduces non-linearity. rest of the complexity & how model captures underlying patterns

Binary classification is algo

logistic reg^n and when output labels are 1 or 0

ie.; $\hat{y} = 0$ or 1 for given x

ie., $\hat{y} = P(y=1 \mid x)$ inter-spectrum

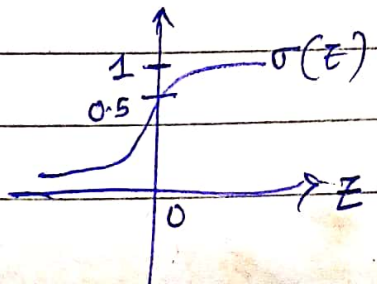
Parameters of L.R. → $w \in \mathbb{R}^{n_x}$, $b \in \mathbb{R}$

output $\hat{y} = \sigma(\overbrace{w^T x + b}^{z}) = \sigma(z) = \frac{1}{1+e^{-z}}$

sigmoid func^n

; $z >>> \longrightarrow \frac{1}{1+e^{-z}} \approx \frac{1}{1+0} = 1$

$z <<< \longrightarrow \quad " \quad \approx \frac{1}{1+(\text{Big no.})} \approx 0$



$y = 1 \longrightarrow L(\hat{y}, y) = -\log \hat{y}$

avg of L.F. of entire training set

---

w, b → real nos. J(w, b) → cost funct
→ ht. of the convex funct

global ~~local~~ optimum of this convex = min w & b to minimize J

~~start~~ iterations of gradient descent help ~~to~~ ~~get~~ the pt. step down to global ~~o~~ optimum

learning rate → how far of a step do we take for each iterat

: ⟶ update /

derivative — update of the change ~~we~~ we want to make to the parameters w,

dw is the variable used in code for representing the derivative

~~dw~~ $\frac{dJ(w)}{dw}$ is slope, i.e., $\frac{ht.}{width} = \frac{J}{w}$

1 step of back propagat on a Computat graph yields derivative of final output var.

~~1st~~ — dJ/dvar = dvar in code

Training with Mini Batch G.D.

Cost func graph -

- Cost func is supposed to reduce (exponentially) as the iterations increase.
- For mini-batch, this func is different for the diff. batches (can be higher than last or previous func, too.), it too decreases tho. with more noise, noise can be reduced by using smaller learning rate.

Size of Mini - Batch :

size is between 1 and m, cus,

if, 1 ⟶ every training eg. is a mini batch so vectorisation needs to be done for each, which's inefficient.

This is Stochastic G.D. (ie., if mini size = 1) tho, program is run regularly, this is inefficient

If m ⟶ Batch G.D. it is. too long to iterate. again, inefficient, ∴ we land in middle b/w m & 1.

If size $\in (1, m)$, ① vectorisation (per 1000 egs.)
② Faster learning, compared to last 2 G.D.
③ program is good.

→ Tips for size -

If small datasets ($size \leq 2000$): use Batch G.D.

For larger: mini batch size must be power of 2.
(usual sizes of mini-batch for larger datasets is 2^4 till 2^9)
↳ eg. In the last eg. (lec-1), we would use 1024, not 1000

Make sure the mini-batch size fits CPU/GPU memory, by empirical way (try diff. powers of 2)

~~~~~~~~ ~~~~~~~~ ~~ofe~~

~~Better~~

] Exponentially weighted averages

— better optimization technique to G.D.

Eg: Temperatures : $T_1 = 4°C$, $T_2 = 9°C$, ... $T_{180} = 15°C$,
If Temp vs days graph looks noisy,
do this,

(1) $V_0 = 0$, $V_1 = 0.9 V_0 + 0.01 \theta_1$; $\theta_1 \rightarrow$ Day 1 Temp.
i.e., avg v_i with a weight of 0.9 times value, and adding 0.01 times current day's temp.

Sly, $V_2 = 0.9 V_1 + 0.1 \theta_2$, ... $V_t = 0.9 V_{t-1} + 0.1 \theta_t$

general ~~fla~~ formula,

$$V_t = \beta V_{t-1} + (1-\beta)\theta_t$$

To get no. of days over which β, the avg has been calculated use the ~~below~~ ~~fla~~ forla, $(1/1-\beta)$

eg: here, $\beta = 0.9$ $\therefore$ $1/1-0.9 = 10$ (days)

greater no. of days → lesser noise.
→ smoother the ~~exponential~~ curve.

cus, more weight to v_{t-1} compared to θ_t,
so, if

Scanned with CamScanner
~~~~~~~~

If the days were 2, curve looked like stacks to one. Blue is for 10 days. - larger window

Also, blue one spreads out as days hold more weight