(1)

* Dropout :- It is one of the mostly used and effective regularization Technique in DL;

Concept :- Dropout :- Randomly turning off (or) swith off neurons (which lies in I/p (or) HL) (setting their output to 0) during training with some probability;

epoch 1
epoch 2
⋮
(n)

that means if you did 10-epochs, that means you worked on 10-diff NN'

- This forces the network to not dep' too heavily on specific ↓ neurons, but instead learn redundant, robust rep';

In this epochs, you are training on a same data, but on a different NN';

- Dropout as an Ensemble method :-

* It gives (2%) improvement gey

* In normal ensemble learning (like Random forest) you train many models separately & then Combine them;

* epoch 1, 2, 3 ... - n are subset of main NN' ✓

* with dropout, you don't literally train multiple networks, but you simulate this effect inside a single network;

* reduce the nodes;

*

How it works :-

@ training :-

- At each step, dropout randomly "removes" (sets zero) some neurons with probability (p)
- i.e., each mini-batch is processed by a diff' sub-network (a small version of the full model)
- over many training iterations, lot of different sub-networks get trained

Ex :- If you have (100) neurons in a layer & use dropout rate (p = 0.5) each forward pass effectively trains a network with only ~50 neurons;

@ Testing / prediction / Inference :-

- We don't drop any neurons; All are active;
- But since each neurons was only active a fraction of time during

training, we scale their o/p's (mult. by 1-p) so that the expected o/p matches training;

Training – randomly drop neurons, scale survivors up by (1/q)
Testing – keep all neurons, no scaling needed, because expectation already matches;

Ex:-
Training :- many weak teams practicing seperately;
Testing – the entire team together, but already calibrated to match the average of all smaller teams;

Mathematical explanation:-
$h = [h_1, h_2, h_3 \ldots]$ be the o/p (activations) of a layer;

Dropout rate (p)
keep probability $q = 1 - p$ ✓

we define a random mask vector $(\gamma) = [\gamma_1, \gamma_2 \ldots \gamma_n]$ where

$\gamma_i \sim \text{Bernoulli}(q)$

i.e $\gamma_1 = \begin{cases} 1 & \text{with prob. } q \\ 0 & \text{with } \text{" } p \end{cases}$

The dropped-out activations are:-

$$\tilde{h}_i = \frac{\gamma_i}{q} \times h_i$$

(*) division by (q) – inverted dropout;
(*) @ training time, dropout is applied
(*) @ test time, all neurons are active, so we just use (h)

h = be the o/p's of a layer;   layers. Dropout (0.5)

say (one) hidden layer with (4) neurons;

$$h = [h_1, h_2, h_3, h_4] = [2, -1, 3, 4]$$

say ($p = 0.5$) i.e ($q = 0.5$)

⊛ each forward pass = different sub-network;

Pass (1):- Random mask (γ) = [1, 0, 1, 0]   its vector ✓

apply inverted dropout scaling $\left(\frac{1}{q} = \frac{1}{0.5} = 2\right)$

$$\bar{h} = [(2 \times 1 \times 2), (-1 \times 0 \times 2), (3 \times 1 \times 2), \ldots]$$

$$= [4, 0, 6, 0]$$ ✓ means only 1, 3 neurons actives ✓

Pass (2):- $\gamma = [0, 1, 1, 0]$

then $\bar{h} = [0, -2, 6, 0]$ means neurons 2, 3 are active

pass (3):- $\gamma = [1, 1, 0, 1]$ then $\bar{h} = [4, -2, 0, 8]$ means 1, 2, 4 are active ✓

over the whole training process, many diff. sub-networks are trained;

step (2):- testing :- We don't drop anything → full output;

$$h = [2, -1, 3, 4]$$

Since we scaled during with (1/q), no extra scaling is needed @ interference;

this way, the expected activation is consistent ✓

this is like an (ensemble) :-

Training - Every forward pass = different "mini-network"

Testing — using all neurons = like averaging across all those sub-network say all once;

⊛ more Robust, prevents overfitting, improves generalization;

(ES) Early stopping :- It is a strategy to stop training before the model starts overfitting

- When training a NN', the training loss usually keeps dec', But the validation loss dec' @ first, then starts increasing after point (overfitting);

- ES, halts training when validation loss stops improving, so the model parameters are saved @ point of best generalization;
  ↳ A model's ability to perform well on new, unseen data after being trained on a specific dataset

overfitting = model memorizes training data, losses ability to generate

By stopping early, we don't allow the network to reach the overfitting stage;

It acts like a form of implicit regularization;

Basically Regularization techniques (2) categories :-

- (1) Explicit Regularization :- We add penalties to loss func' to modify training process;

$L'(\omega) = L(\omega) + \lambda R(\omega)$ where $R(\omega)$ is a penalty

L_1, L_2, dropout & data augmentation;

- (2) Implicit Regularization :- No extra penalty is added, but the training ~~dyn~~ dynamics themselves acts as a regularization;

Ex :- (1) Early stopping (just stop training earlier, no term added in the loss)

(2) SGD with small batch sizes;

(3) Network architecture choices;

SGD is fundamentally an optimiser aimed @ minimising the loss func', but the inherent randomness in its gradient updates acts as a form of implicit R;

(1) keras.callbacks.EarlyStopping(----)

→ When you train a DL model, it learns by going through the data again & again (called epoch). But sometimes, after a certain point, the model stop improving & may even start to overfit.

→ Early stopping is like saying: "Hey model, if you're not improving anymore, stop training early instead of wasting time";

terms:-

(1) monitor = 'val_loss' ⇒ validation loss; if val_loss stops improving, then training may stop;
↓
means keep an eye on

(2) min_delta = 0 ⇒ (i) Min. improvement we care about;
(ii) Zero means even the tiniest improvement counts;

(3) patience = 0 ⇒ (i) Zero: stop immediately if no improvement;
say patience = 3, then it means; ⇒ wait for 3 more epochs to see if the model improves again before stop;

(4) verbose = 0 :- (i) Controls how much info is printed;
(ii) Zero means silent
(iii) 1 = prints messages like: "epoch 10: early stopping"

(5) mode = 'auto' → decides what direction is better ✓

(6) baseline = None → A ref score you expect ✓
Ex:- If baseline = 0.8 for accuracy → training will stop if accuracy never goes above 0.8;
(ii) None means no baseline check

7) restore_best_weights = False :-

(i) After stopping, should it go back to the (best weights) (i.e point where performance was best)
(ii) False = keep the last weights
(iii) True = restore the best weights before things started getting worse.

8) start_from_epoch = 0 :-

(i) which epoch to start checking for improvement;
(ii) Ex:- start_from_epoch = 5 means, "don't check for stopping until after 5 epochs"

Ex:-

monitor = 'val_loss' → the teacher is checking test performance, not practice performance;
min_delta = 1 → improvement must be atleast 1 mark, otherwise it doesn't count
patience = 2 → give the student 2 more days to improve before stopping practice
restore_best_weight = True → If the student was doing best on 7th day & got worse later, we roll back to day 7's learning.

Simply:-

Early stopping :- stop training when the model stops improving, but be a little patient & keep the best version if needed ✓

callbacks :-

It is like a (helper / assistant) that watches your training process & does something automatically @ certain points;

(*) "Extra rules or actions" you attach to your training ✓