Regularization and Best Practices

Regularization :- Any modification we make to learning algorithm to reduce its generalization error rather than training error. Trade increased bias for variance by adding some prior knowledge

Strategies

a) Extra constraints and penalties

b) Restrict param values

c) Data Augmentation

d) Adversarial training

e) Ensemble methods | Dropout

→ Better to have an overcapacity model (overfit) and then regularize it to required level rather than searching for correct capacity.

Parameter Norm Penalties

→ Limit capacity of model by adding parameter norm penalty $\Omega(\theta)$ to objective function J

$$\tilde{J} = J(\theta; X, y) + \alpha \Omega(\theta)$$

$\alpha = 0$ → unregularized model ⇒ α = some large number → underfit

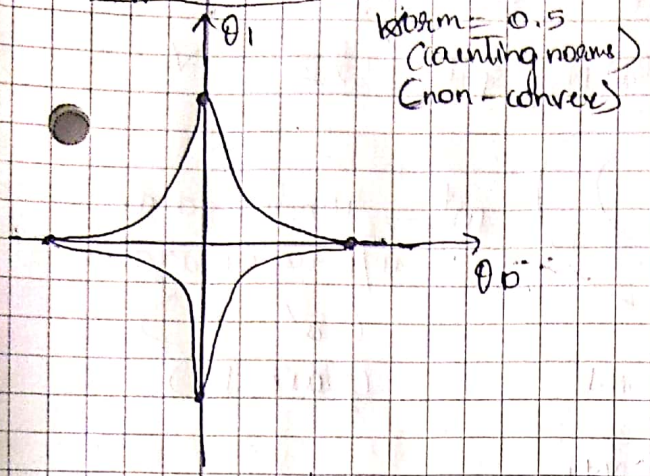
→ Typically we omit bias from regularization $\Omega(\theta)$ because bias term is much more simpler & depends/induced by one value rather than weight which is induced by multiple values

→ Usually Each layer must have a separate penalty 'α' but we use a single penalty 'α' to ease computation

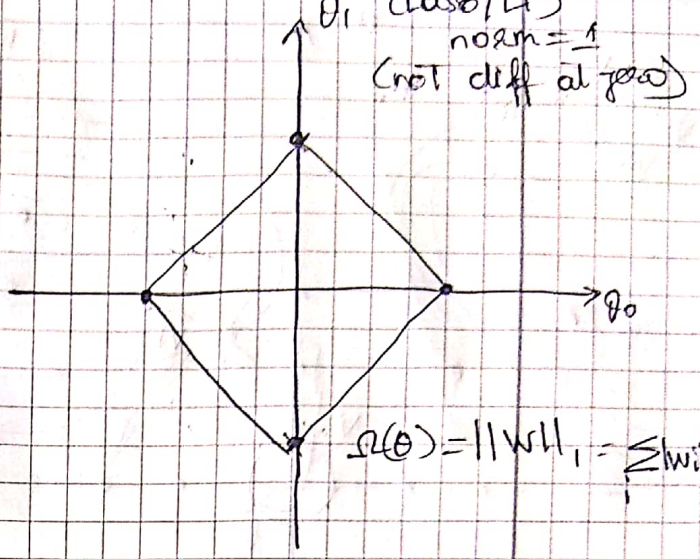
Need for parameter sparsity :- if parameters are sparse (Mostly near one / zero and not between). It is easier to store and compute

→ Most useful params would be params near one & useless ones tend near zero. Norms tend to provide sparsity in various ways

Different Parameter norms

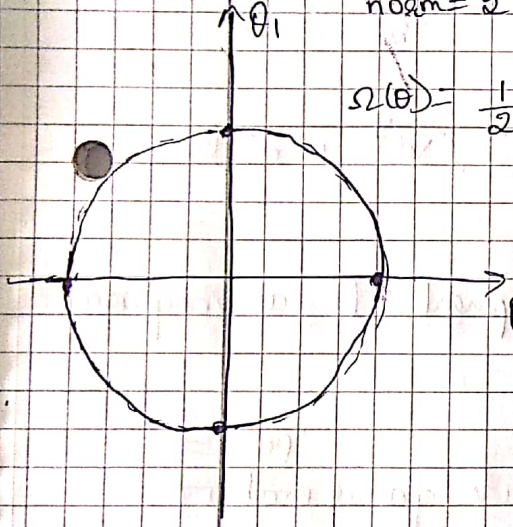


norm = 0.5 (counting norms) (non-convex)



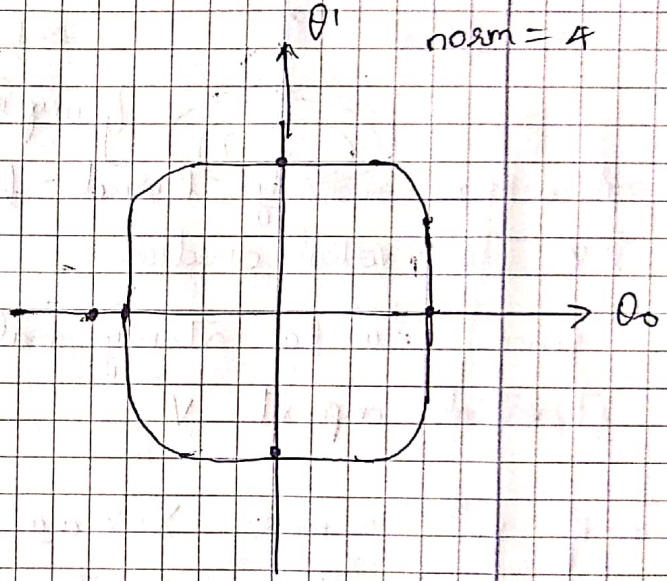
θ1 (Lasso/L1) norm = 1 (not diff at 0)

$\Omega(\theta) = \|w\|_1 = \sum |w_i|$



norm = 2 (Ridge/L2)

$\Omega(\theta) = \frac{1}{2}\|w\|_2^2$



norm = 4

L2 norm

$$\tilde{J}(\theta; X, Y) = J(\theta; X, Y) + \frac{\alpha}{2}\|w\|_2^2$$

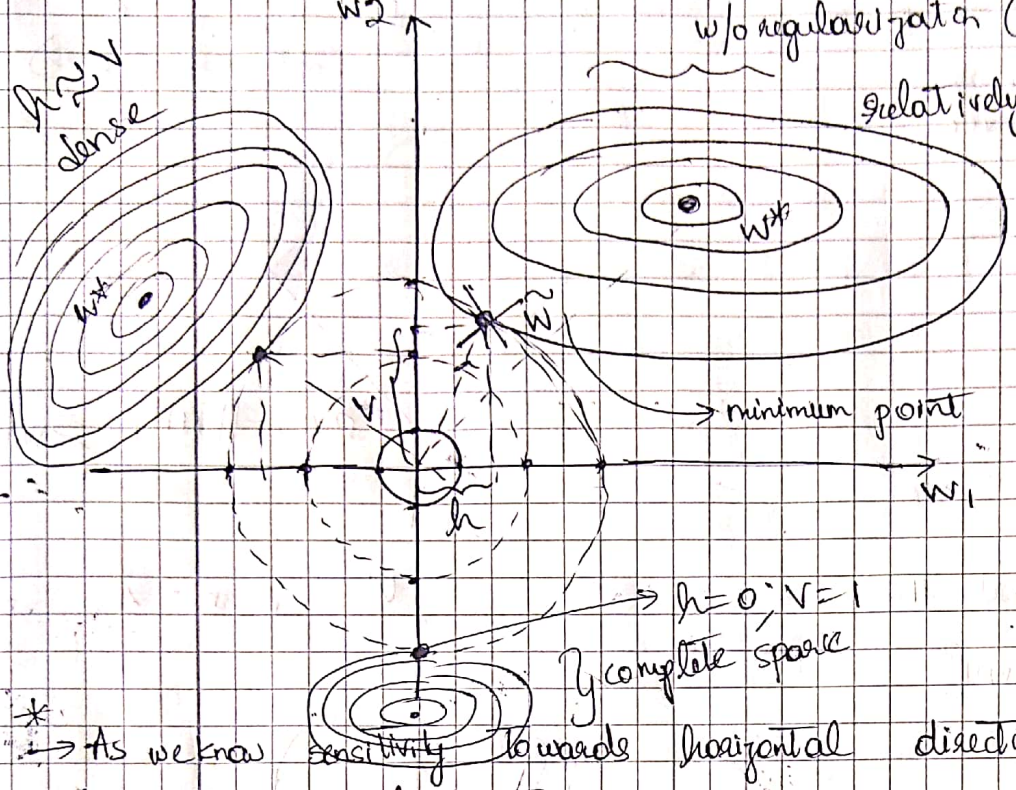
$$\nabla_w \tilde{J} = \alpha w + \nabla_w J(w; X, Y)$$

$$w := w - \epsilon \nabla_w \tilde{J}$$

$$w = w - \epsilon \alpha w - \epsilon \nabla_w J(w; X, Y)$$

$$w = (1 - \epsilon\alpha) w - \epsilon \nabla_w J(w; X, Y)$$

** → At each Training step we decrease w by factor of $(1 - \epsilon\alpha)$ ✓



w* → Min loss param w/o regularization ($\alpha = 0$) (plain J)

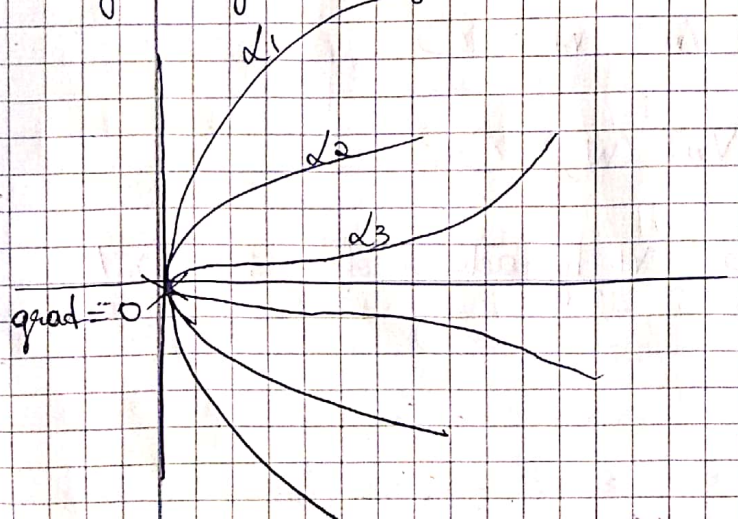
* → As we know sensitivity towards horizontal direction w^* is much lesser than vertical direction

** → Hence $\tilde{w}$ has extremly small horizontal component h as compared to vertical component V

→ if w^* is near X, Y axis we would get much sparser matrix (para)
if w^* is at 45° to center then we would get dense params

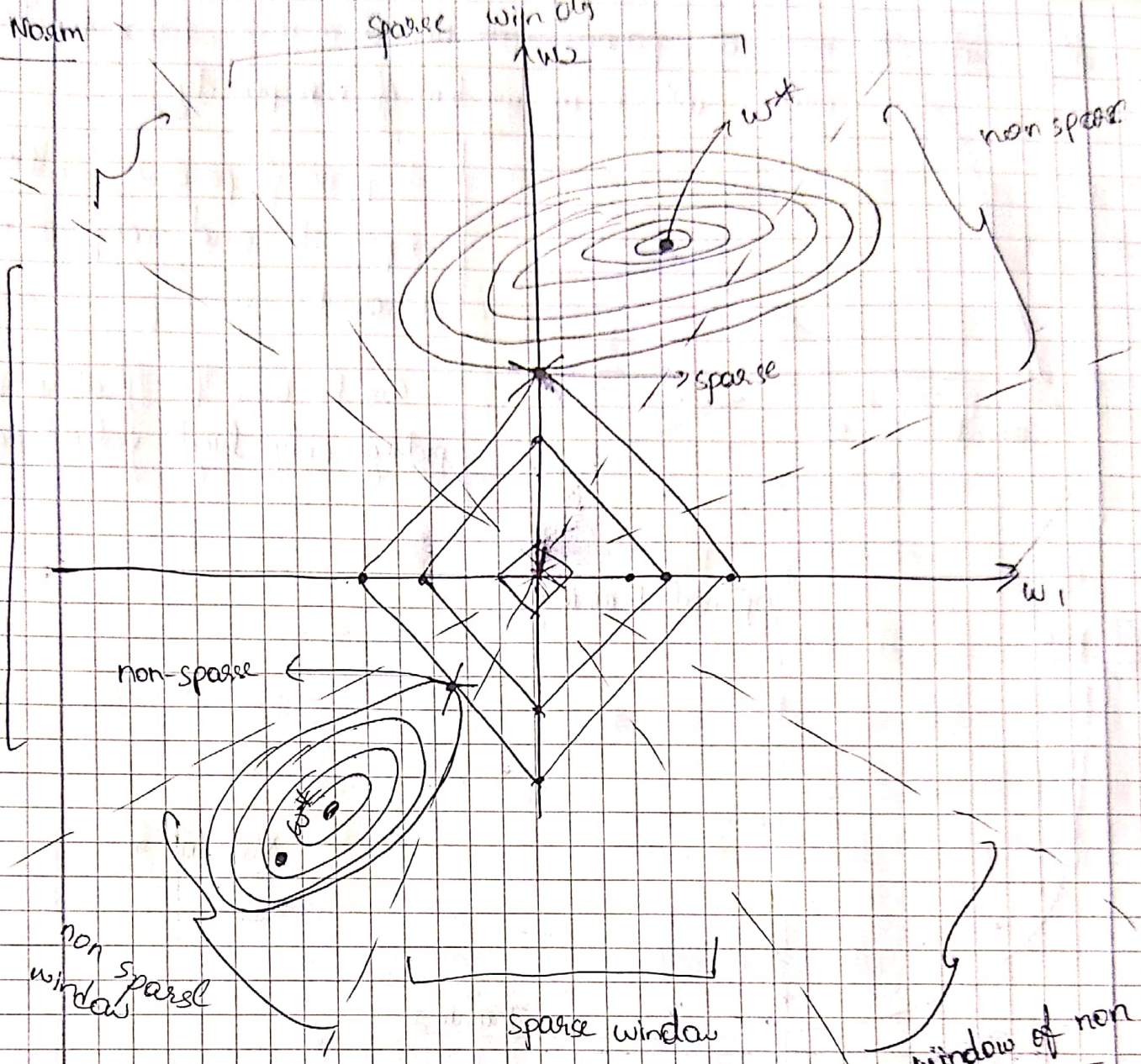
→ L2 norm penalizes large weights due to "$(1-\alpha\epsilon)w$" term in gradient if w is large then $-\alpha\epsilon w$ is large, hence it pulled down faster

→ Only converges to zero at zero or when w^* is exactly on X or Y axis



$\alpha_3 > \alpha_2 > \alpha_1$

L1 Norm



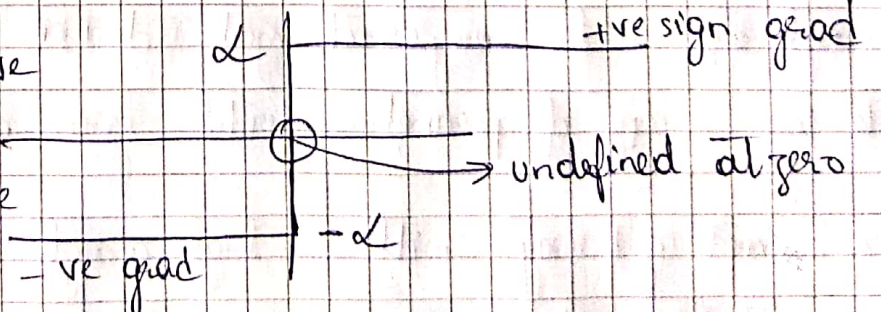
L1 Norm :- $\Omega(\theta) = \|w\|_1 = \sum_i |w_i|$

$$\tilde{J}(w, x, y) = J(w, x, y) + \alpha \|w\|_1$$

$$\nabla \tilde{J}(w, x, y) = \nabla_w J(w, x, y) + \alpha \, \text{sign}(\nabla \|w\|_1)$$

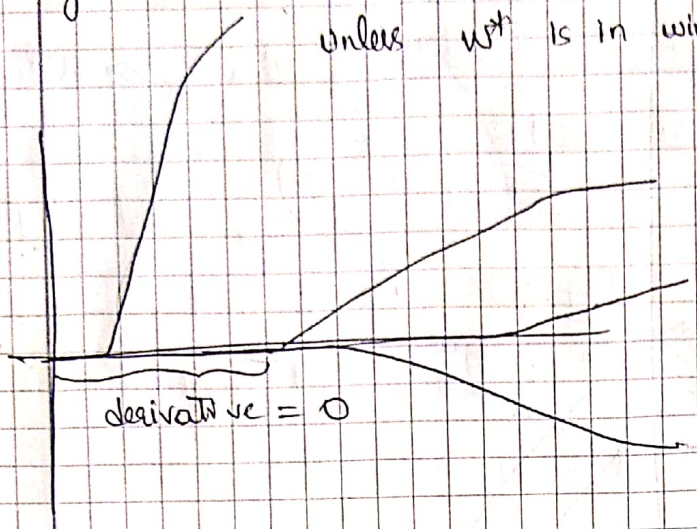
if sign is +ve

if sign is -ve



→ Undefined at exactly zero

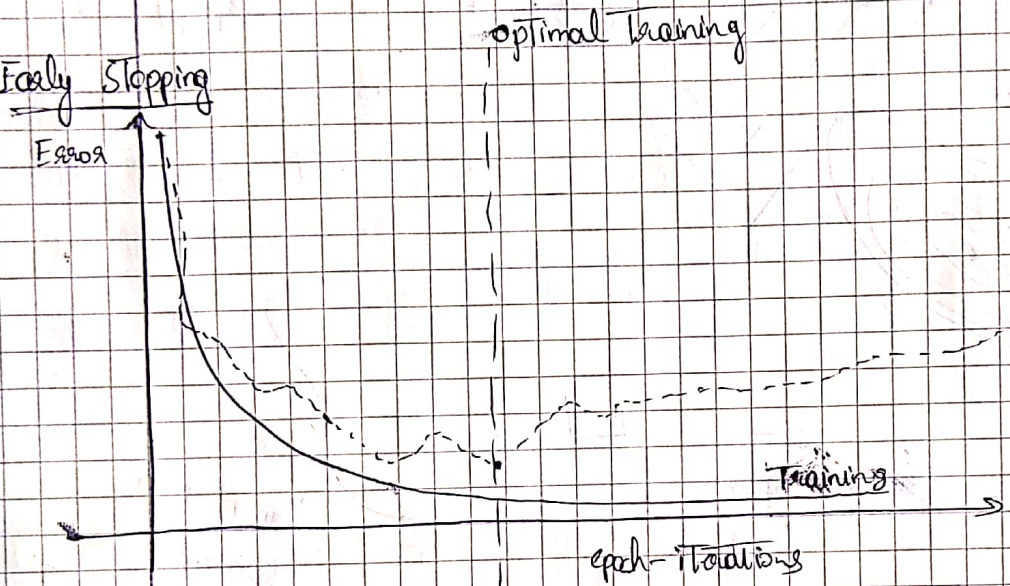
L1 generally Tends To sparse params as its derivative is constant and sign unless w^* is in window of non sparsity



→ Sparse params could be compressed and computation can be saved

→ Can be used for feature selection picking only high valued params

II) Early Stopping



→ When training with sufficient overcapacity on Training data The error decreases but validation error starts increasing after certain point/epochs

→ Why not monitor and stop at this point ?

a) To monitor and make a stop decision we must have a good validation set with enough size else we cannot make decision on small validation set

b) Need To save copy of parameters until error decreases for some iterations

for (a) we cannot get away with high enough validation step and second stage retraining. (b) → save in cpu/ inexpensive storage

Retraining

→ If our data set was small and we had taken validation data from it To perform early stopping, Then we can reintegrate back in Two ways

a) After early stopping, Combine old validation + old training data to new Training data. Re initialize & reset the n/w. Start Training from scratch only till epoch where we stopped early

b) Instead of reinitializing network since its costly, Simply Treat validation as part of Training set and keep on Training Till avg error decreases below certain limit. But this still doesn't guarantee minimal as (a).

How early stopping acts as regularizer :- It slows down network growth and keeps it To small neighbourhood of initial parameter θ_0

→ Much better Than L1/L2 as we are not changing weights and allowing natural fit of network. Also no hyperparameter Tuning like (α) is required in early stopping

Other regularizations

Ensemble, Dropout, Adversarial Training, parameter sharing etc

Best Practices (Machine learning Yearning)

1) Setting up dev and Test sets

Trainingset :- On which we Train our algorithm

Dev set :- Tune parameters, select features, (hold out crossvalidation)

Test set :- Evaluate final performance, but don't use for making any decisions like Tuning hyperparams

→ Dev & Test allows Team To see how algo is doing.

→ To/30 split may not be prevalent in modern deeplearning era as Neural net are data hungry & perform well on large data (>95-1.)

→ Choose Dev and Test set To reflect data you expect To get in future and which you are planning To do well on. Collect better dev/Test set

** → Choose dev and Test from same distribution, if you don't have this data now, then proceed to deploy. Later once you have collected relevant data from users retrain the model with new dev/test set and redeploy it

Example :- Training set :- Cat pictures from website + Japan + India + USA
Dev set :- Cat pictures from USA + India } Different distribution
Test set :- Cat pictures from China

if $Error_{dev} >> Error_{Test}$ then reasons could be

i) Overfit on dev set

ii) Even though Test & dev are from distro, Test set could be much harder to learn (Cannot do anything about this)

** iii) Dev & Test from different distribution ⇒ (Wasted Optimization effort)

* iv) Size of Dev and Test set

$.01\% = \frac{0.01}{100} = 1 \times 10^{-4}$

→ Dev set should be large enough for us to detect improvements b/w two classifiers. For example :- if Dev set has 1000 examples then Classifier A = 90.1% Classifier B = 90.01% ⇒ Cannot be detected as $0.01\% = 10^{-4}$ and we have only 1000 (10^3) examples we need atleast 10^4 examples to detect 0.01% change

→ Test set must be large enough to give us confidence about overall performance of your system

III) Single number evaluation metric

→ Usually having a single number evaluation metric helps to iterate quicker and optimize better than multiple target

Example :- Instead of Precision + Recall Optimize / Use F1-Score

→ In case we need or have some wierd combination of metric like F1-Score, Running Time, Size, Mem required, False positives, False-ves etc

→ keep only one metric as optimizing, on which you try to improve as much as possible like F1-score

→ Running Time, Memory, Model Size could be satisfying metric and classifier just needs to be satisfying a cut off criteria

Select classi	Classifier	F1-Score	Running Time(Sec)	Memory RAM (MB)	Model Size (GB)
X	A	0.89	400	100	2
X	B	0.92	450	200	3
✓	C	0.77	150	50	0.7
✓	D	0.85	170	70	1
✓	E	0.72	150	70	1.1
X	F	0.71	1,00	90	0.7
X	G	0.81	200	60	4

satisfying metric

- Running Time ≤ 200sec
- Memory ≤ 75 MB
- Model size ≤ 1.5 GB

Selected classifiers C, D, E whose F1-score would be optimized further

**

→ Select dev/test set and setup a basic metric as apt unless you are changing any app already running in prod or really matured.

II) Changing Dev/Test sets and metric

→ Lets say you took initial cat images uploaded by users as your dev/test set. Now users are uploading killer images, hence our dev/test set is no longer valid. Change them accordingly

Problems

i) The actual distribution has changed from dev/test sets we used.

ii) You have tuned so much on dev set such that there is significant error b/w dev & Test sets indicating that you have overfit on dev set
Throw away dev set and get new dev set

iii) Business changed the metrics they told earlier
Ex:- Classifier A = 90% accuracy + lets through porno images
Classifier B = 85% accuracy + doesn't let porno images

→ we had used A as accuracy was our metric, but it's letting porno image through.

→ Include penalty on porno in our metric and reiterate process

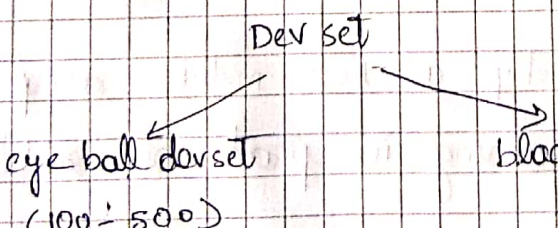
new metric = Accuracy − penalty * % of porno images let in

5) Basic Error Analysis

Error analysis:- Process of finding/examining dev set examples that our classifier got wrong to better understand its underlying cause

** → Only fix errors that may give you max boost in your metric

→ Make an excel sheet of errors and their plausible reasons



→ Instead of looking at each error manually, make a separate dev set called eyeball from existing one which is representative of all categories but still small enough to be manually examined.

→ Analyse and make spread sheet on eyeball devset

Image id	Mislabeled	Cat	Dog	Mouse	Error reason
1	✓	✓			Instagram filter
2			✓		
3	✓				Instagram filter
4					
:					
99					
100	✓			✓	Too small and behind elephant
Total misclassified	20%	5%	70%	25%	

→ Instead of fixing cat / Mouse, if we fix dog related error we can get more gains and create data

** → Use this as eyeball dev set and Tune your hyper params on blackbox dev set. Don't Tune it on eyeball set unless it's absolutely necessary. If you see overfit then throw this away and get new dev set

→ For smaller devsets ≤1000 make eyeball & blackbox set the same

VI) Bias and Variance

* Bias
  - avoidable → Can be bridged by bigger models
  - unavoidable → even humans make mistake on this.

Variance : Difference of error b/w dev v/s Train. Regularize well

Example :- Audio recognition : Some audio clips are loaded with noise that even us humans make mistakes. Hence human error = 8%

⇒ accuracy = 92% (Optimal bayes error ⇒ Unavoidable bias)

optimal error = 8%.

Training error = 20%.

Dev error = 22%.

Reason:- High bias & low variance

Correction:- Build bigger model

Training error = 10%.

Dev error = 20%.

Reason:- High variance & low bias

Correction:- Regularize model
Add more Training data

Training error = 20%.

Dev error = 30%.

Reason:- High bias & ~~low~~ high variance

Correction:- Bigger Model + Regularize

Training error = 10%.

Dev error = 12%.

Reason:- Low bias & Low variance

Correction:- Not needed / good model

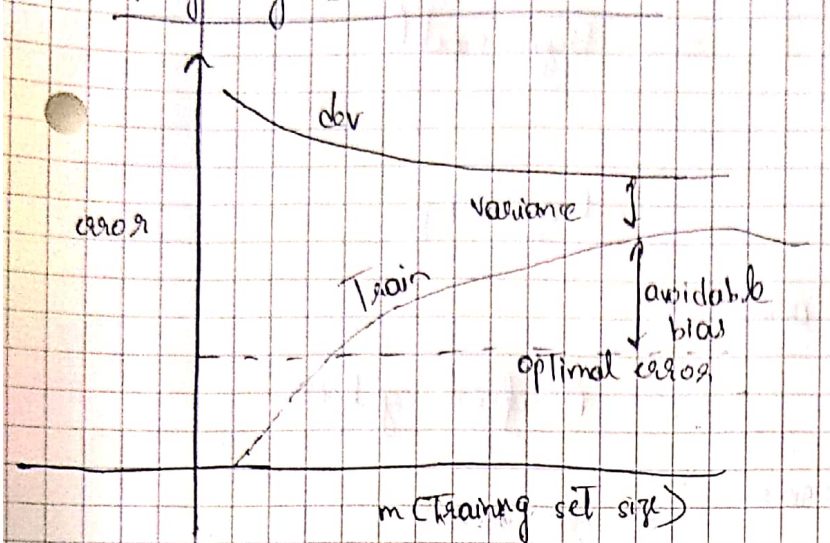
Techniques To reduce avoidable bias

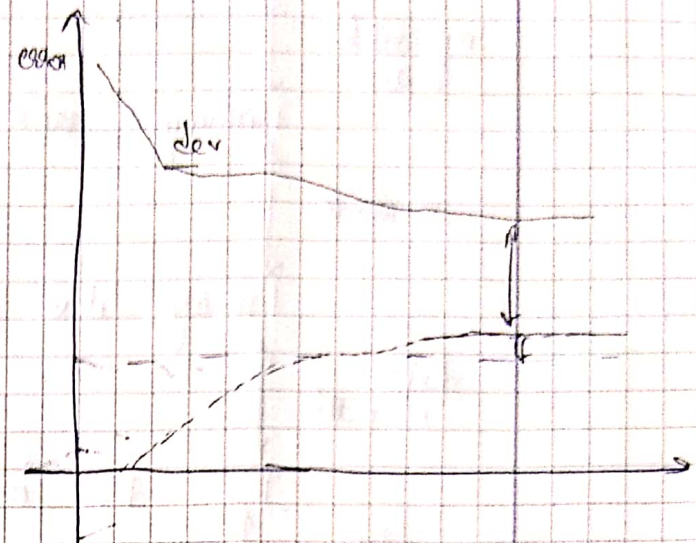
→ Increase model size
→ Modify i/p feature based on error analysis
→ Tone down regularization
→ Modify model architecture

Techniques for reducing Variance

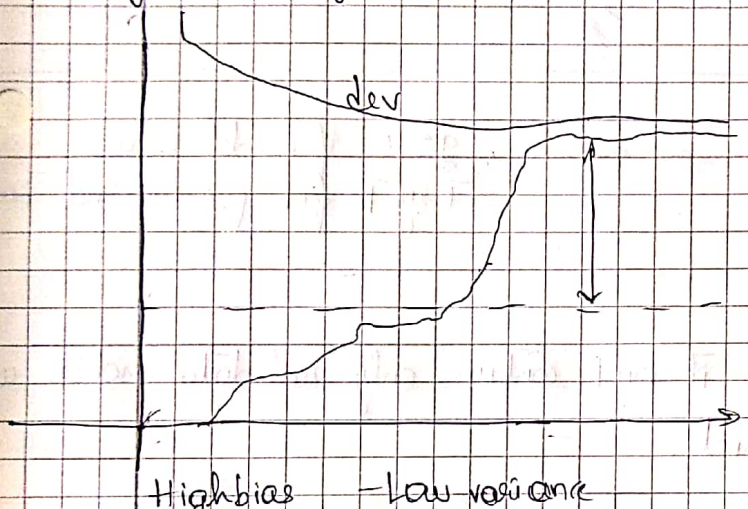
→ Add more varied Training data
→ Add regularizer
→ Early stopping
→ Feature selection (Select lesser features)
→ Decrease model size
→ Modify model architecture
→ Error analysis

Diagnosing bias and variance

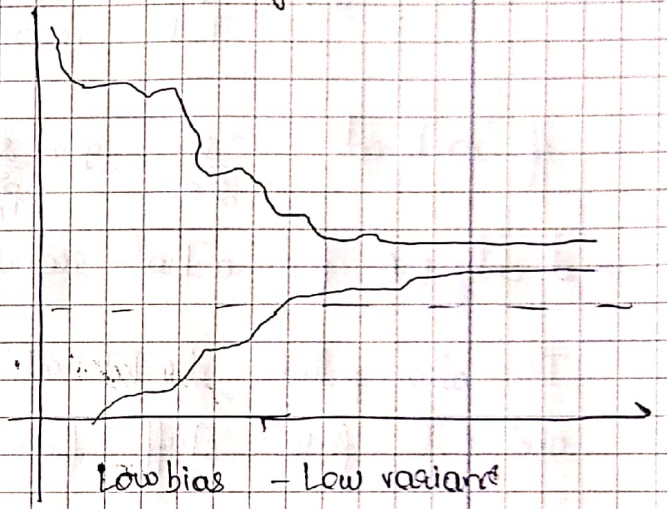




High bias - High variance



Low bias - High variance



High bias - Low variance

Low bias - Low variance

VII) Human level performance

Collect data from humans if they can solve tasks much better than machines (Image segmentation, Audio recog) and not on ones they cannot (Recommendation, Ad search)

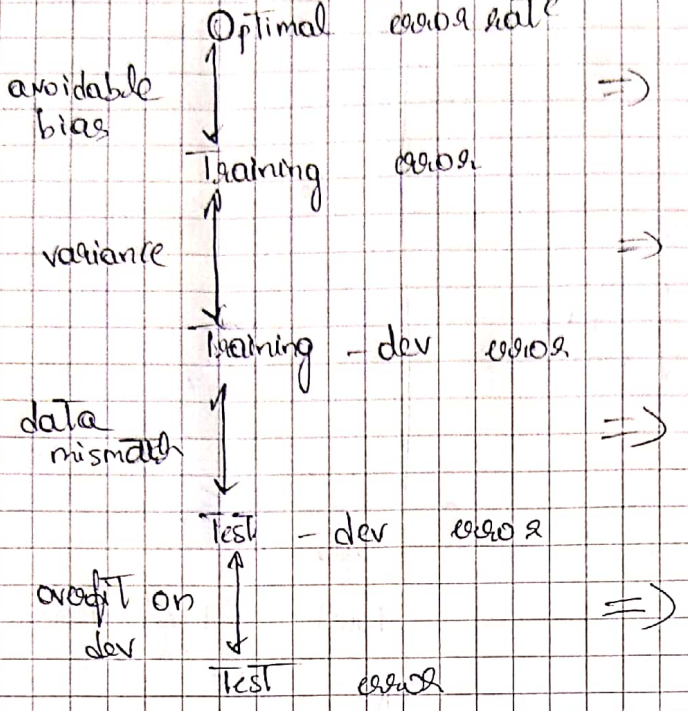
VIII) Train and Test/dev from different distros

**

→ Carve out a subset of Training data as Train dev set to help see variance

→ ~~If~~ It's better not to mix Train-dev-Test data and randomly select if Size(Train data) >>> Size(dev-Test)

Ex:- 200,000 internet cat image 20k mobile cat image



→ if combined 10 200 ≈ 92% 20 / 220 ≈ 8%. 92% of model would try to fit for internet image and only 8% for actual required

→ This also pollutes dev-test set as it must contain only real data (mobile uploads) and not fake stuff from internet.

→ Even if train is entirely different from dev-test, keep it this way and don't mix. Find any other methods for fixing

~~if~~ Penalty score (weighing data)

→ if size (Training) ≈ size of (dev-Test) Then it better to mix both and add a penalty weight more for dev-test rather than train

→ so if classifier misclassifies internet image it must get lesser loss compared to misclassifying mobile uploads.

Ex: 50k internet images 20k mobile images

penalty > 1

$$\text{Cost} = \sum_{\text{mobile image}} \text{Loss} * \beta + \sum_{\text{internet}} \text{Loss} * 1.0$$

Artificial data synthesis

→ Data mismatch can be addressed by artificially synthesizing dev like data and use it to augment training

Ex:- Adding car noise to clear audio of training.

Problems

→ if we synthetically add same car noise (Audi / Volkswagen) to training then if dev has diffent car (Ford / Ferrari), it still doe give mismatch

→ Careful synthesis required