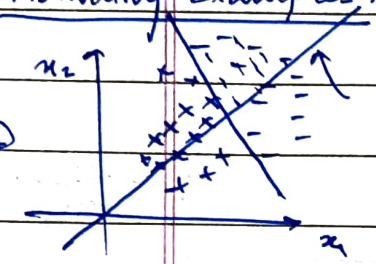
Week 8 - Feature Selection:

PAGE No.
DATE / /

5.1.) Motivation & creating new features:

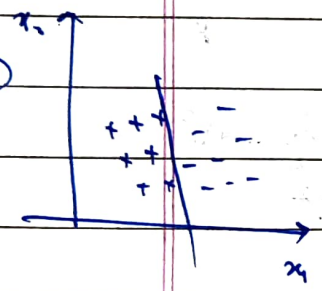
- Articulate why right features are important & wrong are harmful
- Be able to generate new ones from existing ones
- Be aware of common features for common data types
- Be able to apply feature selection methods.

Motivating Examples:



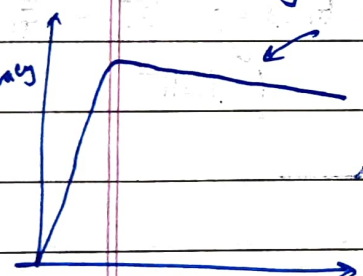
① Any decision boundary of x_2 / x_1 parallel will misclassify.

But a projection line will seperate them.

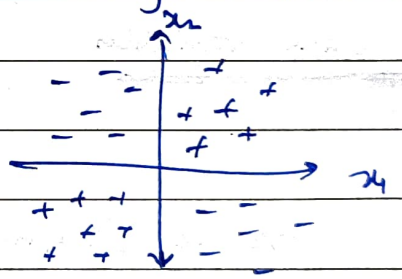


② Even though x_1 can classify on its own x_2 affects slant of the decision boundary & acts as a confounder.

So, we essentially have :-



Generating new features:-



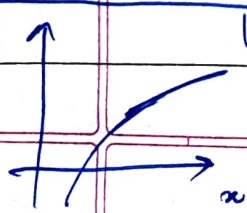
$x_3 = x_1 x_2$ plane elevates +ve samples

Transforming variables:

Power: $\hat{x} = x^p$ | Log: $\hat{x} = \log(x + c)$ | exp: $\hat{x} = e^{ax+b}$

- Transforming big range $\longleftrightarrow$ small range etc.

Objective is to transform to an easier distribution:



log for example

Objective:

$\hat{x}$ should have an easier dist. in the analytical sense.

Lets say $\hat{x} = f(x)$ assuming they are bijective

$\Rightarrow f^{-1}(\hat{x}) = x$

If $p(x)$ and $q(\hat{x})$ are the distributions of the two,

$$q(\hat{x}) = p\left(f^{-1}(\hat{x})\right)\left|\frac{d}{dx} f^{-1}(\hat{x})\right| \Rightarrow q(\hat{x}) = \frac{p\left(f^{-1}(\hat{x})\right)}{\left|\frac{d}{dx} f(\hat{x})\right|}$$

Derivation :-

$$P(f(x) \le \hat{x}) = P(x \le f^{-1}(\hat{x})) = F(f^{-1}(\hat{x}))$$

Method of transformations:

$Y = g(x)$ g is differentiable | g is strictly increasing.

Now, $F_Y(y) = P(Y \le y) = P(g(x) \le y) = P(X \le g^{-1}(y))$

$= F_X(g^{-1}(y))$

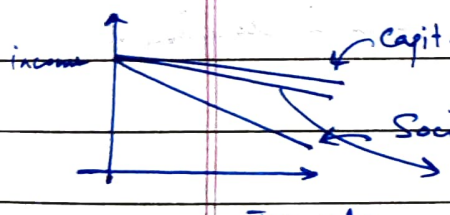
$$\Rightarrow \frac{d}{dy} F_Y(y) = f_Y(y) = \frac{d}{dy} F_X(g^{-1}(y)) = \frac{f_X(x_1)}{g'(x_1)} \quad [x_1 = g^{-1}(y)]$$

Dummy Variables:

- Convert discrete variables with N levels to N-1 binary variables.

Apple, orange, Banana, Grapes $\Rightarrow W_1 x_1 + W_2 x_2 + W_3 x_3 + b$ (4th)

- Interaction variables

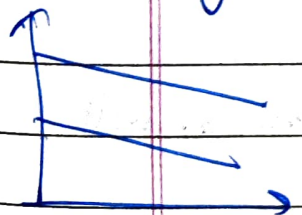


Socialist & Capitalist countries on seperating.

Average effect. (If we don't include type of economy)

$y = W_1 x_1 + W_2 x_2 + b \quad (+ W_3 x_1 x_2)$

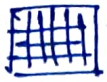
x_1: Tax rate; x_2: social economy type; $x_1 x_2$: Models the original graph with different slopes.



If combination is important $\Rightarrow$ Interaction term needed.

x_1 x_2 $\leftarrow$ $x_1 x_2$ are correlated and can have some effects.

5.2) Features for images:



2D array + colour dimension

PAGE No.
DATE

Common Image Features:

- Pixel level statistics
  - Gray scale histogram
  - Color histograms
- Texture based stats.
  - Fourier descriptors
  - GLCM
    Gray level co-occurrence Matrix
- Hybrid
- Shape based.
  - Hu invariant moments ⇒ $\eta_{ij} = \sum\sum (x-\bar{x})(y-\bar{y})^j I(x,y)$

5.3) Features for audio & text:

- Meaningful features - power in different frequency bands

Signal ⟶ fourier transform → Useful data.

Time windows also important. Window → Fourier Transform

MFCC is a default feature selection ① Windows of time (Overlapping)

② DFT of window - Power Spectral Density

③ Filter bank - log energy ④ DCT (Discrete Cosine Transform)

Common Features to Extract from text:

"The cat was chasing the rat" or "I got lucky in the test today"

"I was not very lucky today"

- Histogram of words from a dictionary (Feature vector) → $\begin{bmatrix} 0 \\ \vdots \\ \vdots \end{bmatrix}$ are not ordered.

Problem: Too much importance to all words.

- TF-IDF: Term frequency - Inverse Document Frequency

$$tf(t,d) = \frac{f(t,d)}{\sum_{\tau \in d} f(\tau,d)}$$

Term frequency over a document

$$idf(t,D) = \log \frac{|D|}{1 + \{d \in D : t \in d\}}$$

↑ corpus

Entire feature = $tf \times idf$

(Also, pretrained deep neural network help for features as well.)

5.4) Feature reduction 1 of 2 :

PAGE No. / DATE / /

Features on pretrained deep neural networks :-

filter based methods / (Filter / Wrapper / Embedded)

x_{11} ... x_{1000}

$x_{1000,1}$ ... $x_{1000,100}$

2^{100} subsets so too much

Filter based: For each feature decide keep vs discard → Train ML with kept features.

Wrapper: Generate subset, train & validate ML model on subset.

Embedded: - Lasso regularization. Subset selection & ML integrated

Correlation-based elimination:

Blocks around diagonal - Feature reduction [Correlation based feature reduction]

Utility based subset selection:

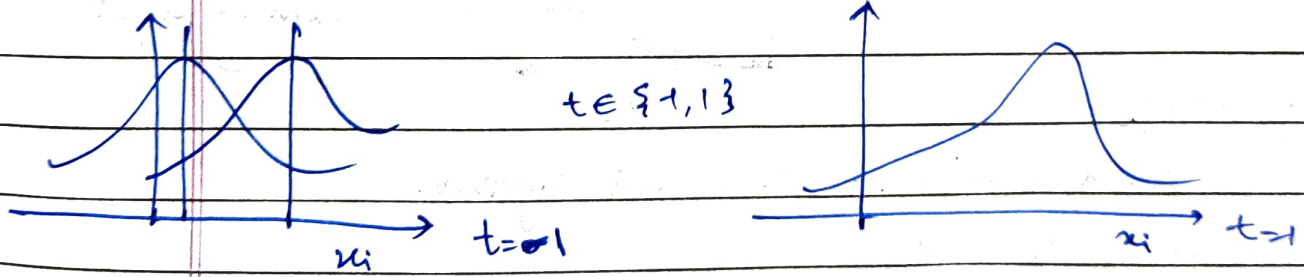
- Regression : Correlation | - Classific^n : t-test | - AIC & BIC

How the features related to target output.

1.) x_i correlation ⟹ Highly correlated with t may be good subset for prediction.

- Doesn't take in acc. interaction of x_i & x_j - Filter based method

2.) Classif^n : Two class classification.



Relative to own variances: Assumes they are distributed as Gaussian

t-test formula : $\left(\frac{\mu_1 - \mu_{-1}}{\sqrt{\frac{\sigma_1^2}{n_1} + \frac{\sigma_{-1}^2}{n}}} \right)$ ← width of Gaussian important

AIC & BIC : Akai Info. Criteria & Bayesian Info Criteria

5.5.) Forward Selection & Backward Elimination.

PAGE No.
DATE / /

$x_1 \longleftrightarrow x_{100}$

Subset $S \leftarrow \{\phi\}$ initially $R - (x_1, \ldots, x_{100})$

for $i = 1 \to n$ (100) we measure the maginal utility.

for x_j in R

measure maginal utility of x_j in ML model

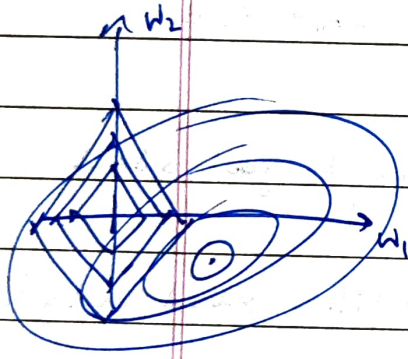
Include x_k with largest Maginal Utility in S & out of R.

Backward Elimin^n works in opp. dir^n :

Issues: If x_i & x_j correlated, we may not be able to find that

Backward elimination - correlated variables may already be in S

LASSO & elastic net:

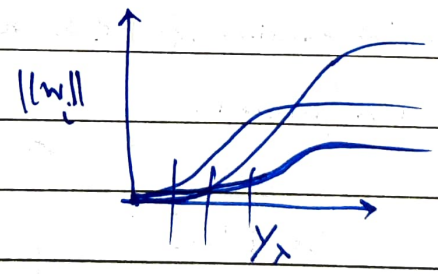


Contours will graze to one of the corners.

$$Z(w) = E(w) + \lambda \|w\|_1 + \lambda_2 \|w\|_2$$

↑ Elastic-Net

summing weights



Elastic Net:

keeps correlated variables kept in or out together.

Assume: $x_1 = x_2$

$w_1 x_1 + w_2 x_2 + w_3 x_3$ we want $w_1 \simeq w_2$

but $(w_1 + \alpha) x_1 + (w_2 - \alpha) x_2 + w_3 x_3$ is also the same

but l_1 norm is same. How to diff.? $\Rightarrow$ l_2 Norm

l_2 penalty large for $(w_1 + \alpha)$ & $(w_2 - \alpha)$ case.

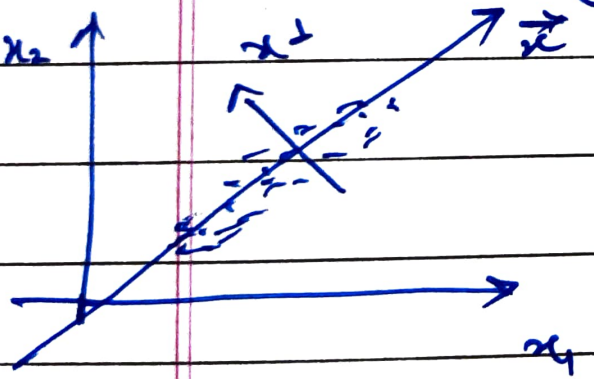
L1 SVM : (Or L2 SVM is original SVM)

$$\frac{1}{2}\|w\|_2^2 + C\sum_i [1 - t_i y_i]_+$$

$$\lambda\|w\|_1 + C\sum_i [1 - t_i y_i]_+$$ for some values of λ you have certain w to be 0.

PAGE No.
DATE / /

Principal Component Analysis :



$(N \to d)$

Captures essence of x_1 & x_2.

Eigen vectors corresponding to higher Eigen values. Each captures contribution from all underlying contribution.

Week 9-10 (7-1-1)—(7-2-3)

PAGE No. / DATE / /

(7-1-1) Introduction to Neural Networks (12:21)

- Objectives
  - Adding layers ↑ modelling power
  - Mathematical expression — Backpropagation algorithm
  - List ways to improve backprop.

Increasing non linearity in models:

1. Linear models (Regressor / Classifier) $\hat{y} \propto \sigma(Wx)$
2. Support vector machines → Fixed features $\hat{y} \propto \sum_i t_i a_i K(x, x_i)$
3. Neural Networks → Trainable features $\hat{y} \propto \sigma(W_2 \sigma(W_1 x))$
   [or multiple layers (But not too deep)]
4. Deep Neural Networks $\hat{y} \propto \sigma(W_n \cdots (\sigma(W_1 x))$
   [σ is a non linear function, not necessarily sigmoid]

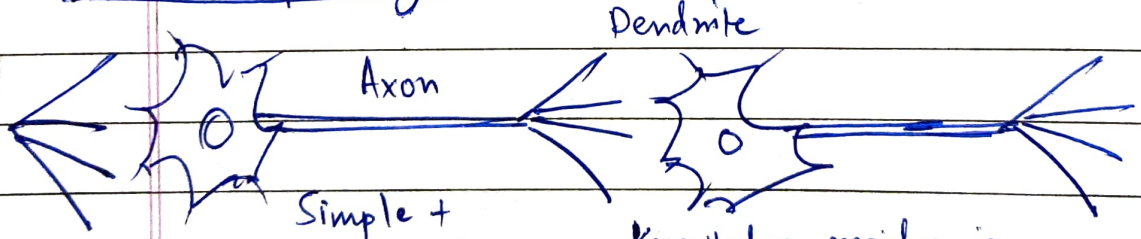
(7-1-2) Layered functional representation of NN:

W are matrices for neural networks

Transform feature vectors into other vectors.

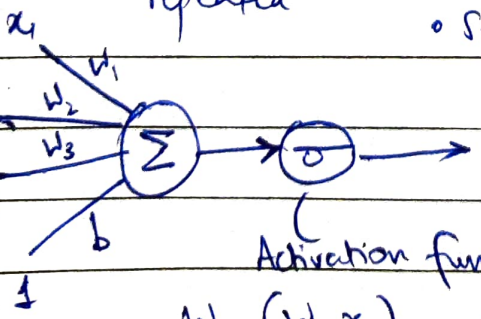
Update based on $\partial L / \partial W_{ij}$ (chain rule of derivatives)

Structure of Biological Neuron:



Simple + repeated

Knowledge resides in
- Structure of connect^n & Strength of conn^n



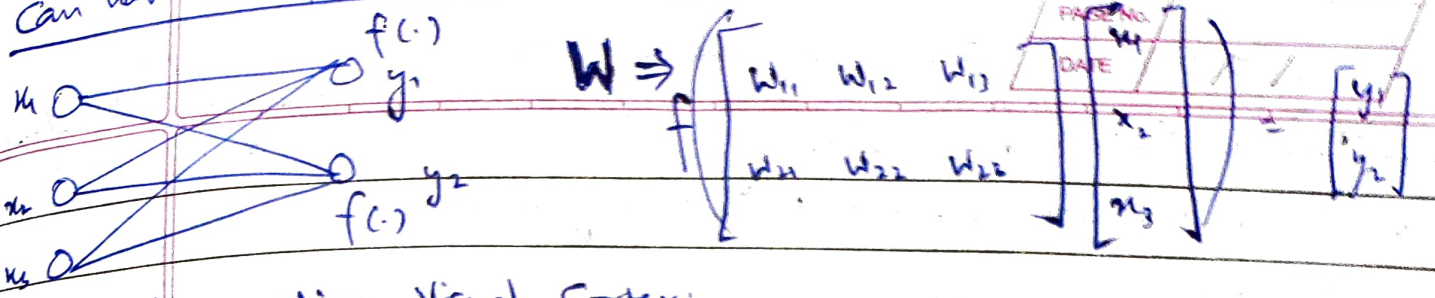
Simplification of Biological Neuron.

Activation function

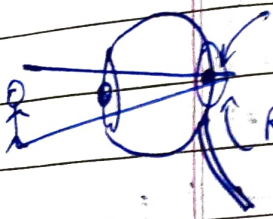
$W_2(W_1 x) = (W_2 W_1) x = W_3 x$

Which is why every layer but the last has some non linear function 'σ'

Can have multiple outputs as well:

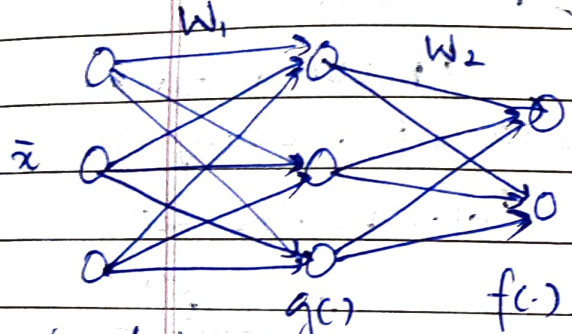


(7-1-3) Mammalian Visual Cortex:

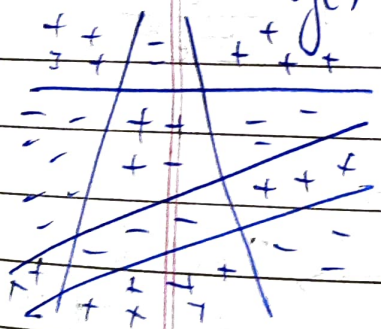


Specialised sensory cells → optic nerve → V_1 ("shapes") ← V_2 ← ... (More complex features) So on. [ONE DIR^N FLOW]

(7-1-4) Introducing Hidden layer in NN:



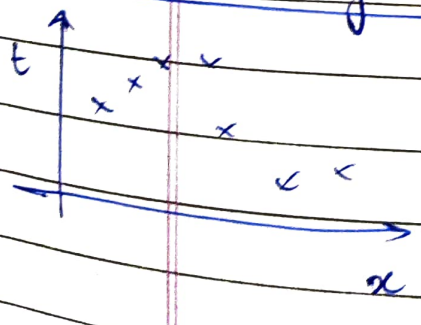
$$\bar{y} = f(W_2(g(W_1 x)))$$



Each linear boundary is one neuron

Hence can do non-linear accurate classification

What hidden layers are doing:-



Basic $f(x) = c$

Lin. $f(x) = W_1 x + W_0$

Non lin. : $f(x) = W_1 \phi_1 x + W_2 \phi_2 x + b$

SVM $f(x) = \sum W_i K(x, x_i)$

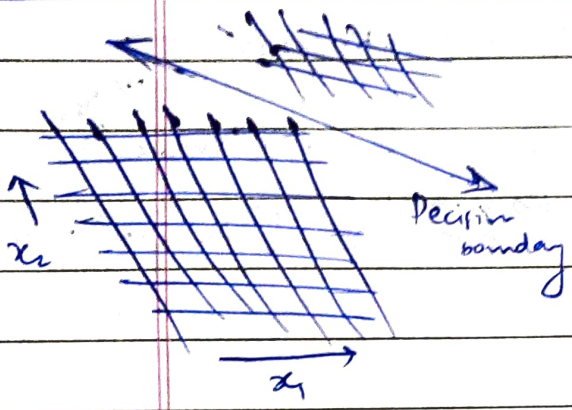
NN fcn = Summation of activation function with weights & biases.

Repeated continuation of canonical function with W & b & layers.

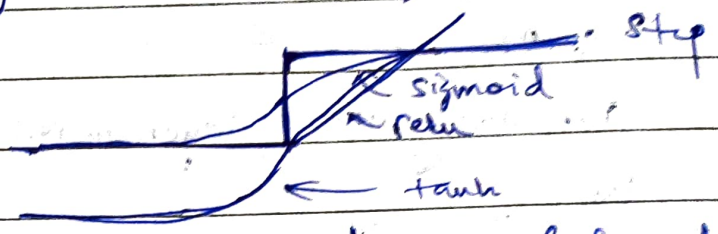
Universal Approximation Theorem:

- $f \in C(W_2 \sigma(W_1 x))$ can approximate with error ϵ in a compact interval any smooth $f(x)$
  - Provided size of W is arbitrary
  - σ is also smooth, but NOT a polynomial

(7-1-5) Activation function:



We need a smoother version of a step function with non zero gradients ⟹ sigmoid etc.



Step: Classification only. Not used anymore	σ & tanh: trainable approx. of step	ReLU: Preferred as fast convergence

Softmax: Generalized σ for multiclass classification net

$\frac{sgn(x)+1}{2}$	$\frac{1}{1+e^{-x}}$	$\tanh(x)$	$\max(0,x)$	$\frac{e^{x_i}}{\sum e^{x_i}}$

Problems with too many layers:

— Too many parameters — Gradient dilution

Hyperparameter: No. of hidden layers. → Input & output dim fixed
Dimension of layers.

The training process is gradient-descent

Gradient of vectors: ∇f $\nabla = dL/dW_{ij}$ matrix

(7-2-1) Chain rule & backpropagation:

$f(x) = g(h(x))$

$f'(x) = g'(y)\, h'(x)$ $y = h(x)$. so on.

eg $\frac{d(\sin(x^2))}{dx} = \cos(x^2)\, 2x$

Algorithm:

- Make a forward pass & store partial derivatives
- During backward pass, multiply ∂ [Library handles this]

(7-2-2) Backpropogation for vector variables :-

$$f(x) = \begin{bmatrix} f_1(x) \\ f_2(x) \end{bmatrix} = \begin{bmatrix} f_1(x_1\ x_2\ x_3) \\ f_2(x_1\ x_2\ x_3) \end{bmatrix} \xrightarrow{x_j}$$

$$J(f) = \begin{bmatrix} \frac{\partial f}{\partial x_1} & \frac{\partial f}{\partial x_2} & \frac{\partial f}{\partial x_3} \end{bmatrix} = \begin{matrix} f_i \downarrow \\ \end{matrix} \begin{bmatrix} \partial f_1/\partial x_1 & \partial f_1/\partial x_2 & \partial f_1/\partial x_3 \\ \partial f_2/\partial x_1 & \partial f_2/\partial x_2 & \partial f_2/\partial x_3 \end{bmatrix}$$

$$J(f) = \left[\partial f_i / \partial x_j \right]$$

i - row index
j - col index

$[f_i \times x_j]$
$|I| \times |J|$

Questions:

- Scale all weights & biases by factor? ⟹ Sign no change
  NOT UNIQUE !! Need to regularize
- Gradients in deep neural networks?

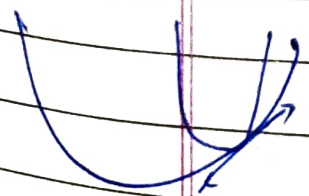
$\prod_i W_i$ - if $|W|<1$ then we have vanishing ∇
if $|W|>1$ then we have exploding ∇ (depends on initialization)

Vanilla gradient descent:

- Iteration loop n
  - Sample loop i
    - $\nabla_\theta L \leftarrow \nabla L_i + \nabla L_i$
  - $\theta \leftarrow \theta - \eta \nabla_\theta L$

Slow model: We have to go through all training samples before we can update even once

(7-2-3) Role of step size & learning rate :-



Same gradient ⟹ Same step for same η

- Same value + Same gradient BUT:
- Different Hessian • Different step sizes needed

⟹ We need learning rate scheduling or some function to model η

Issues with gradient descent:

- Need to find good step size η
- Lots of computation before each update
- Can get stuck in local minima

Solutions: Stochastic GD or Batch GD

1.) Stochastic Gradient Descent:

- Pick any training point randomly

Iteration loop
- Select point
- L_i (compute)
- $\nabla_\theta L_i$
- $\theta \leftarrow \theta - \eta \nabla_\theta L_i$

Need more updates but updates much quicker
$\Rightarrow$ Noisy updates are possible
But unlikely to settle in local minima

2.) Batch gradient descent: (GPU utilization)

[Hybrid of Vanilla GD & Stochastic GD] Most popular.

- Batch formation loop:
  - Shuffle training pt
  - Divide into batches
- Epoch Loop
  - Batch loop
    - Batch
    - $\nabla_\theta L_{Batch_j}$
  - $\theta \leftarrow \theta - \eta \nabla L_{\theta' batch_j}$

Keep increasing batch till GPU error.
Faster than stocho

Double derivative speed up of Hessian & Jacobian:

Let $f(x) = ax^2 + bx + c$

Assuming $a < 0$ [Inverse parabola]

Minima at: $-b/2a$

For any 'x' the perfect step :-

$$\left(-b/2a - x\right) = -\left(\frac{2ax+b}{2a}\right) = -\frac{f'(x)}{f''(x)}$$

step

$$\Rightarrow \eta = \frac{1}{f''(x)}$$

Same logic for:

$$x \leftarrow x - \left(H(f(x))^{-1}\right) \nabla(f(x))$$

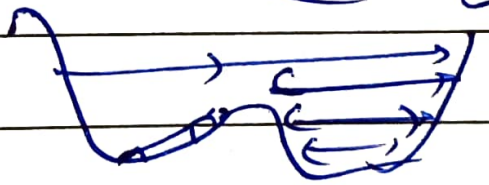
[Not used but approximations of these speed up]

→ Note that it is NOT important to find global minima.
(Since Global Min of traning data ≠ That of testing data)

Adding momentum instead of 2nd derivative :-

$$\theta^n \leftarrow \theta^{n-1} - \eta \nabla_\theta L + \alpha(\theta^{n-1} - \theta^{n-2}) \quad \alpha =$$

0 & 1 (0.8 & 0.9)



Like momentum intuition

[L2 - Weight decay] in NN