Deep MLP (1980's to 2018)

1

1980's, 1990's — 2006 - 2012 → 2,3 layered network.

problems with growth of Deep MLP.

1. Vanishing gradient
2. too little data (easy to overfit)
3. too little Computation power. (hard to run for too many epoch because require lot of time).

2010 → lots of data → Internet → labelled data (Image net) Comptition.

→ Computers (GPU, CPU)

→ NVidia (super useful for solving deeplearning algorithm)

→ v.v. good for deeplearning.

→ new Idea's & Algorithm.

Morden DL

classical → ML [SVM (90's), RF (2000's)] {Theory, Experimental} old new

* Building new theory is Very hard.
  So they do lot of experiments and if it is successful then build the theory.

great result

lot of new idea's in DL → University of toronto } theory
Standford
Companies } experiment.

② Dropout & regularization.

Deep NN → overfit easily
↳ many layers → many weights

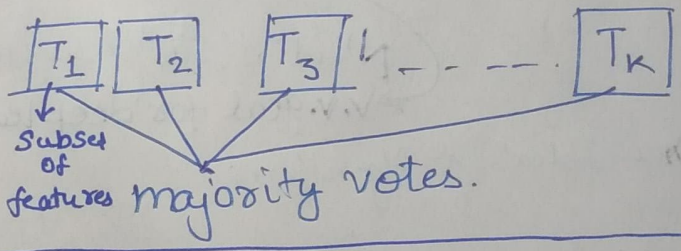
* So we must avoid overfitting we use regulizer like L1, L2 or **DROPOUT**.
↳ extremly simple & elegant

2012 :- Nitesh S. Hinton
↳ Master's thesis

Random forest → sample a subset of columns/features
↳ many trees (DT) → fully grown

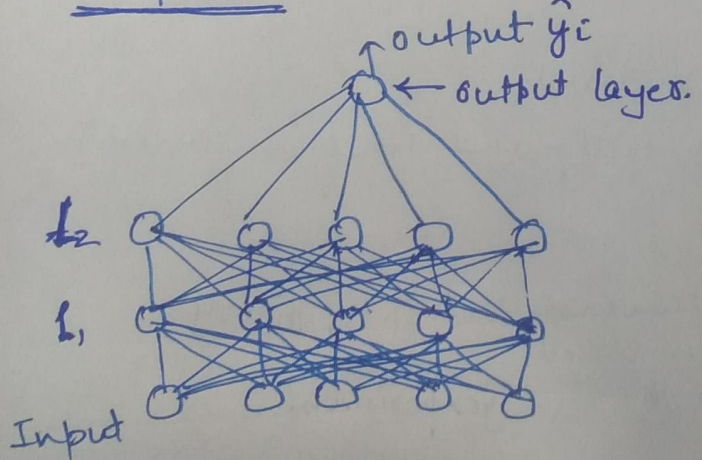
→ DT → overfit Models

$T_1, T_2 \ldots T_K$ are partially grown many trees.

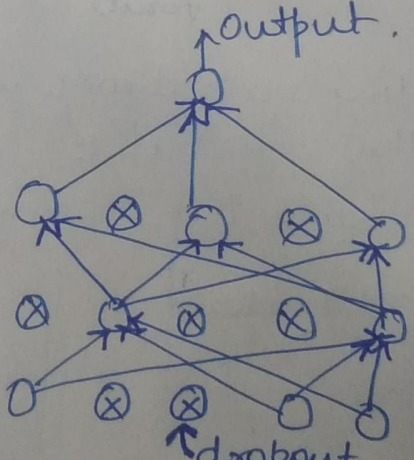


① Random subset of feature and randomization of regulization through randomization.

Dropouts →



a) standerd NN

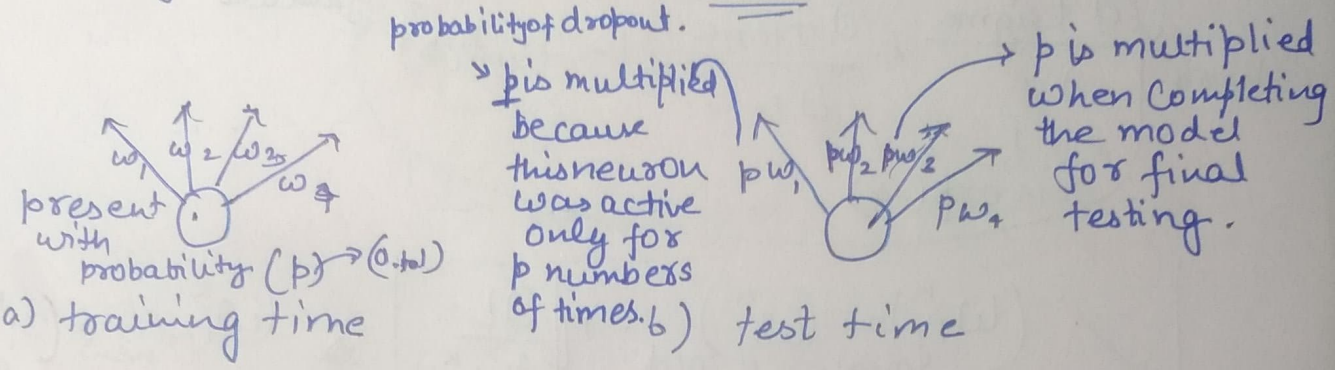


b) Appy dropout for 1 iteration

dropout rate
$a \le p \le 1$
↳ % of point dropout or inactive.

⇒ dropout ≈ random subset of features in radom forest but in NN we have dropout {random subset of neurons active}.

Q training is done but now test will done.



p is hyperparameter generly keep low if overfiting.

③ Rectified Linear Unit (ReLU) ← new Activation function.

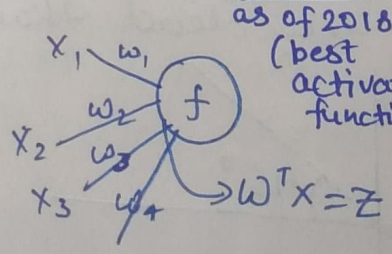
→ Classical NN → vanishing gradient (Sigmoid or tanh)

↳ convergence slow

$$(w^k_{ij})_{new} = (w^k_{ij})_{old} - \eta \left[\frac{\partial L}{\partial w^k_{ij}}\right]$$ ← very small.

very slow changes. when training large network cause lot of time.

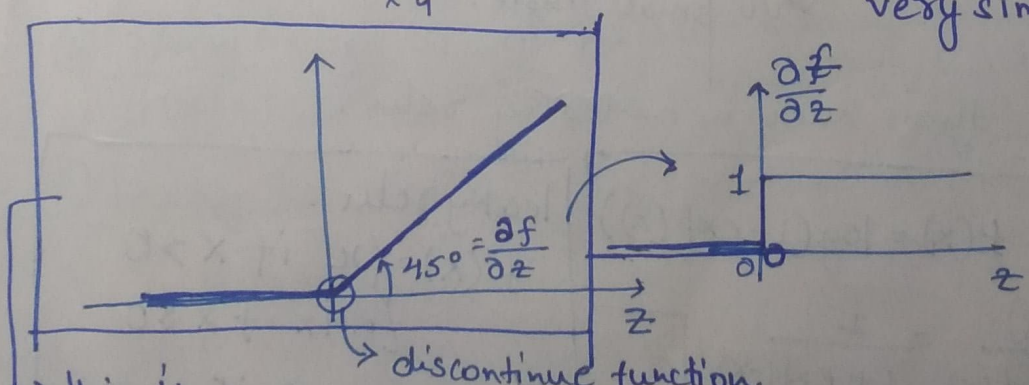
To solve this comes ReLU as of 2018 (best activation function)



f(z) is said z^+ which is max(0, z)

$$f(z) = \max(0,z) = \begin{cases} 0 & \text{if } z \le 0 \\ z & \text{if otherwise} \end{cases}$$

very simple. still powerful.



discontinue function.
i.e. not differnciable at 0. means $\frac{\partial f}{\partial z}$ at 0 is not defined.
which will be optimized in later chapter.

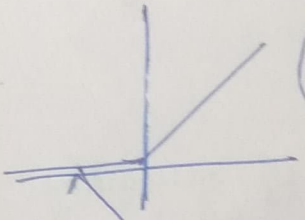
↳ this is graph of Relu activation function i.e. $f(z) = \max(0, z)$

Computing derivative relu.

easy $\begin{cases} \frac{df}{dz} = \begin{cases} 0 & \text{if } z<0 \\ 1 & \text{if } z>0 \end{cases} \end{cases}$ → easy to diff.
→ No problem of vanishing gradient because now derivative will be 0 or 1.
→ No problem of exploding gradient

hard ✗ $\frac{\partial f_{sigma}}{\partial z}$, $\frac{\partial f_{tanh}}{\partial z} = 1 - \tan^2(z)$

$= \frac{1}{1+\exp(-z)}$



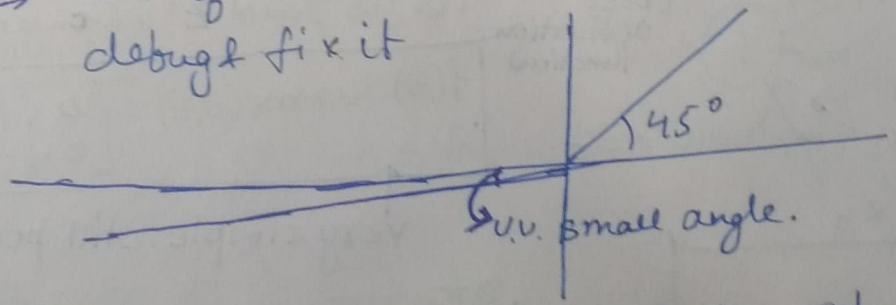
$(w_{ij}^{k})_{new} = (w_{ij}^{k})_{old} - \eta \boxed{\frac{\partial L}{\partial w^{k}_{ij}}} \Rightarrow (w_{ij}^{k})_{new} = (w_{ij}^{k})_{old}$

If z is negative every $\frac{\partial L}{\partial w^{k}_{ij}}$ this will 0. ⓪ ↑ dead activation.

$\Rightarrow 0 * 1 * 1 * 1 \Rightarrow 0.$

* Relu also converge faster for same number of epochs. than tanh function. (because it does not have vanishing gradient problem).

* This is because if z is negative means weights are highly negative and derivation is always 0.
the we can use variation called leaky relu.
⇒ See if there are more dead than active.
debug & fix it



this is known as leaky relu.

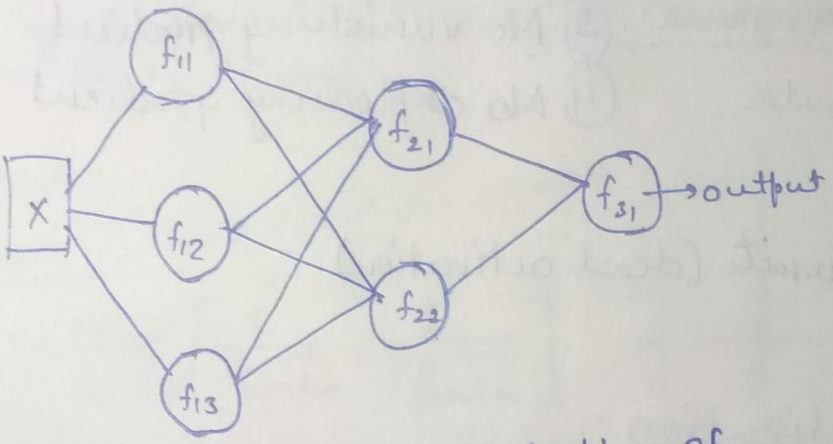
Softplus function: $f(x) = \log(1+\exp(x))$

derivative of Softplus function → $f'(x) = \frac{\exp(x)}{1+\exp(x)} = \frac{1}{1+\exp(-x)}$

leaky relu.

$f(x) = \begin{cases} x & \text{if } x>0 \\ 0.01x & \text{if } x \le 0 \end{cases}$

$f(x) = \begin{cases} 1 & \text{if } x>0 \\ 0.01 & \text{if } x \le 0 \end{cases}$

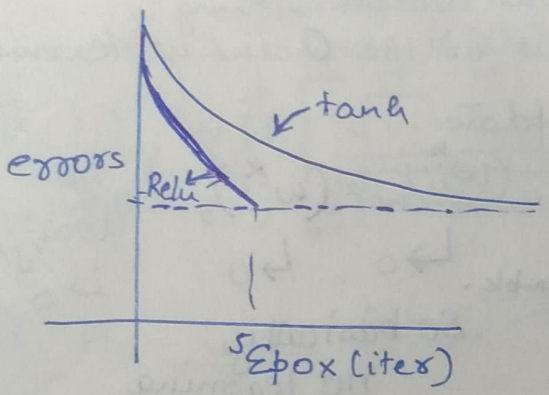


No problem of.

$\frac{\partial f}{\partial z} \in \{0, 1\}$ ✗ exploding ✗ Vanishing

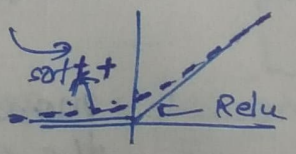
but can cause dead-activation

Experimentally how fast is ReLU.



So, for equal number of epoch relu converge faster than tanh.

To solve diff at 0 problem solution was Softplus + function. $f(x) = \log(1 + \exp(x))$



this can also solve dead activation to certain level.

Varients of Relu

① Noisy Relu's

$f(x) = \max(0, x + Y)$, with $Y \sim N(0, \sigma(x))$ ← this is like adding some bias

② Leaky Relu (avoid Dead activation) like mixture of Vanishing & Relu.

$$\frac{\partial f_{LR}}{\partial z} = \begin{cases} 1 & \text{if } x > 0 \\ a \text{ (0.01)} & \text{if } x < 0 \end{cases}$$

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ ax & \text{if otherwise} \end{cases}$$

Advantage of ReLU.

1. speed up convergence.
2. easy to compute.
3. No vanishing gradient
4. No exploding gradient

disavantage

1. dead Relu units (dead activation).

4) Weight initialization:

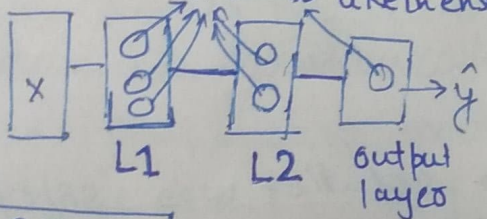
Logistic reg → ① initialize the weight randomly
↳ or ② $w_{ij}^{k} \sim N(0, \sigma)$ (gaussian)

1. init $w_{ij}^{k} = 0 \quad \forall i,j,k$ → v.v. bad idea.
   ↑ all
   → all neuron will start will same thing and always input will be 0 and no learning.
   → same gradient update.

$$(w_{ij}^{k})_n = (w_{ij}^{k})_0 - \eta\left(\frac{\partial L}{\partial w_{ij}^{k}}\right)$$
↳ 0 ↳ 0 ↳ 0

So basically, no learning.

different weight acts as different models like in ensemble.



RF, GBDT
↳ ensemble → more different models are better the output, the better is output of ensembling.

2. if w_{ij}^{k} = large -ve number

ReLU $w^T x = z$ = -ve large
↑ normalized data
↳ Vanishing gradient or dead activation.

Solution

idea ① properties of weights.

1. Weight should be small (not too small)
2. not all zero
3. good Variance ← $Var(w_{ij}^{k})$

$w_{ij}^{k} \sim N(0, \sigma^2)$ ← Gaussian/Normal initialization.
this σ is also very small but not too small.

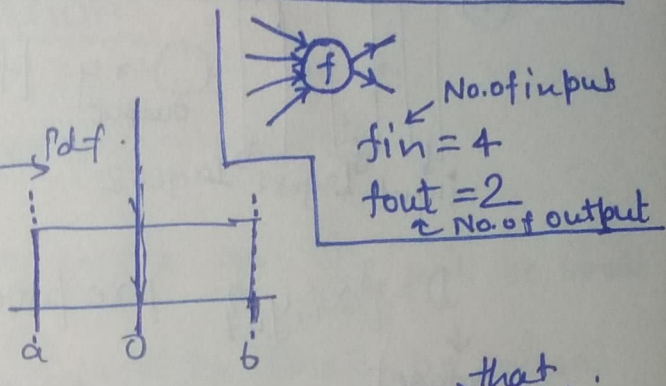
better init strategies.

↳ lots of experiments (strong) ✓ on how to initialise weight.
↳ some theory (weak)

Technique -

idea (2) Uniform init → v good for sigmoid

$$W_{ij}^{k} \sim \text{Unif distrib}\left[\frac{-1}{\sqrt{fanin}}, \frac{1}{\sqrt{fanin}}\right]$$

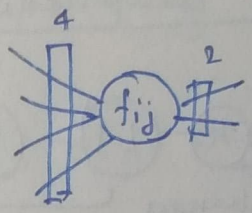


no concrete agreement amongst all researchers. → that which is best initializer.

idea (3) Xavier/Glorot init (2010) → v good for sigmoid.

(a) $W_{ij}^{k} \sim N(0, \sigma_i)$ (Normal) ← normal distribution

$$\sigma_i = \sqrt{\frac{2}{fanin + fanout}}$$



(b) $W_{ij}^{k} \sim U\left[\frac{-\sqrt{6}}{\sqrt{fanin + fanout}}, \frac{+\sqrt{6}}{\sqrt{fanin + fanout}}\right]$ (uniform) ← uniform distribution

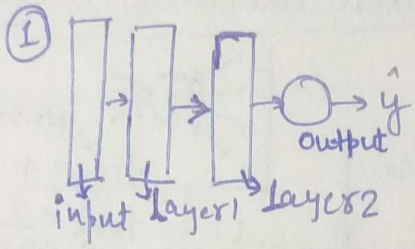
idea (4) He - init. (2015) → V. good for ReLU.

a) $W_{ij}^{k} \sim N(0, \sigma)$ (Normal) $\sigma = \sqrt{\frac{2}{fanin}}$

b) $W_{ij}^{k} \sim U\left[-\sqrt{\frac{6}{fanin}}, +\sqrt{\frac{6}{fanin}}\right]$ (Uniform)

He init is good for relu.

5) Batch Normalization :- (2015)



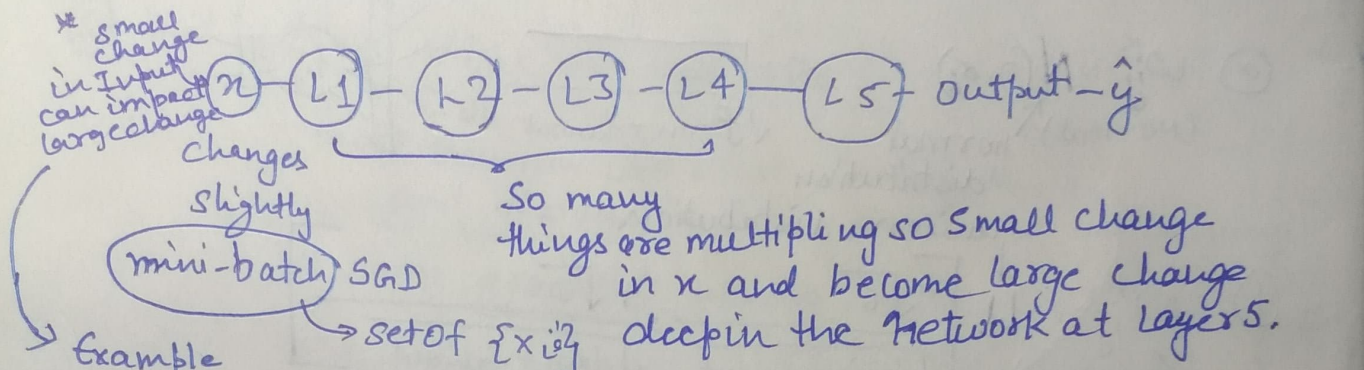
$D = \{x_i, y_i\}$ ↓ Dataset — pre processing step → data Normalizat^n on x_i i.e. mean centring and var. scaling

$$x_i = \frac{x_i - \mu}{\sigma}$$

μ ← mean(x_i)

σ ← std dev(x_i)

ex. of deep network



Example

1 → 1.01 ← slight change

1.0201 ← a^2 → 1.0406 ← a^2 → 1.375

2 layer — more layer then after 5 times

1 will can to 1.375

37% with slight change on sq. funct^n neuron of 5 layer

Problem is called Internal Covariance Shift.

Solution :-

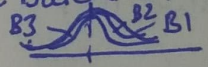
x → L1 → L2 → L3 → L4 → L5

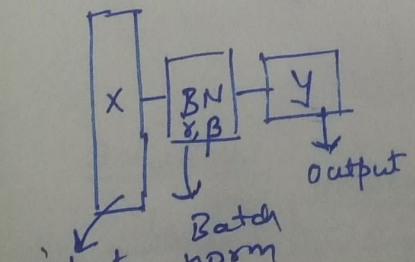
↓ add Batch normalizat^n layer

x → L1 → L2 → L3 → L4 → BN → L5 → ŷ

we apply BN layer it does normalisation for each min batch.

mini batch B1, B2, B3.





Input in BN x_i

Output :- $\tilde{x}_i = BN_{\gamma,\beta}(x_i)$ → It just mean input $z = wx + b$ not x.

$$\mu_{BN} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i$$

$$\sigma^2_{BN} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{BN})^2$$

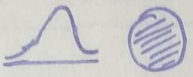
$$\hat{x}_i \leftarrow \frac{x_i - \mu_{BN}}{\sqrt{\sigma^2_{BN} + \varepsilon}}$$ ← small value so that dividing by 0 error does not come.

$$\tilde{x}_i \leftarrow \gamma x_i + \beta = BN_{\gamma,\beta}(x_i)$$

Learning of γ, β is done just like weights learning it can be done using backpropogation.

⊕ **Advantages of Batch Normalisation.**

⊕ faster convergence. L5



- larger learning rate.

η

can be used. as $\frac{\partial L}{\partial w_{ij}^{k}}$ will be small

⊕ act as weak regulizer

use BN + Dropout

and $\frac{\partial L}{\partial w_{ij}^{k}}$ will be small because all the values come from same distribution.

⊕ Avoid internal Covariate Shift → train deep NN easy

⑥ Optimizers :- [Hill descend]

L.r. Reg & Optimization → GD, SGD, mini batch-SGD

$w^* = \min_w \underbrace{L(w)}_{\text{Loss function}}$

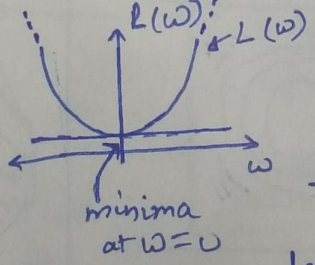
Deeplearning → simple SGD, GD → advances??

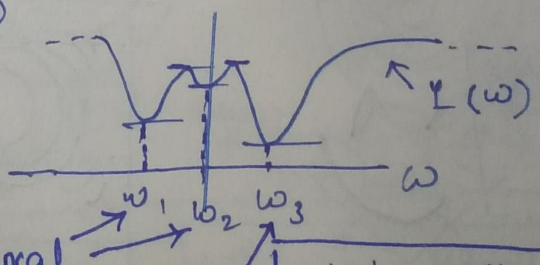
2D Visualisation

$\min_w L(w)$

a) w is scalar

↑ 1 dimension

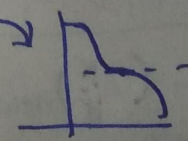




Local minima. global minima. minima, maxima, saddle pt. $\frac{\partial L}{\partial w} = 0$

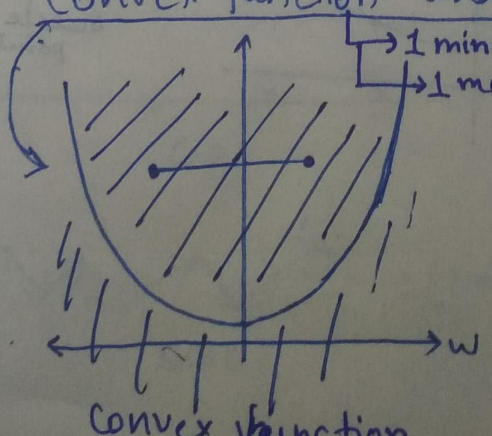
$w_{new} = w_{old} - \eta \left(\frac{dL}{dw}\right)$ ← Stop when $\frac{\partial L}{\partial w} = 0$

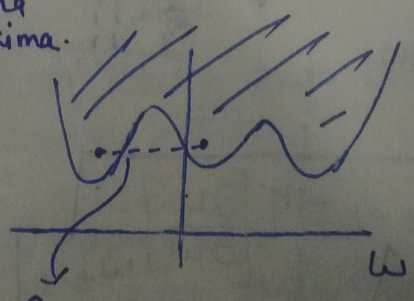
So it could stuck at saddle pt.



In saddle pt also $\frac{\partial L}{\partial w} = 0$

Convex function and Non-Convex function

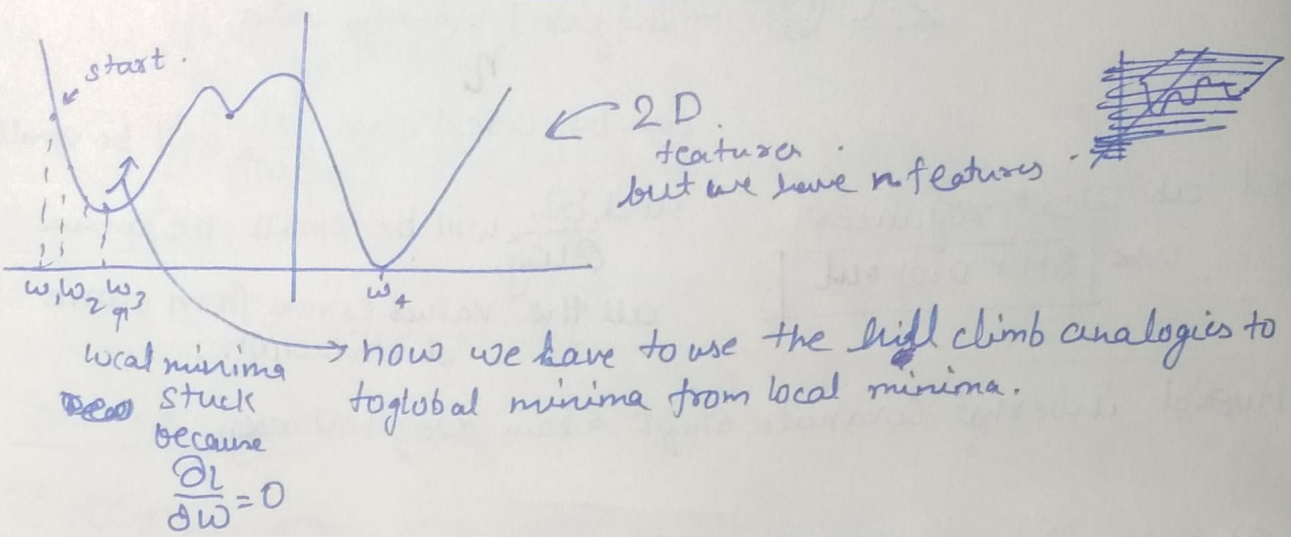




Some pts like in other region ie. it is not Convex function.

property → Convex function have only one minima or maxima.
↳ local minima = global minima

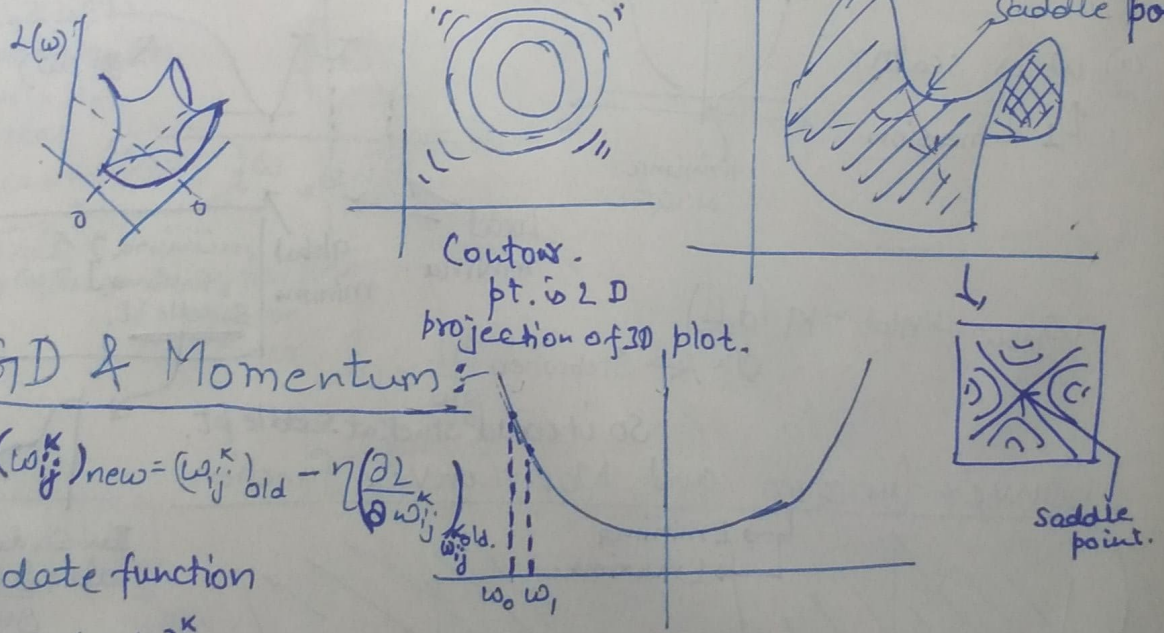
In real life we have non convex functions.



So based on init value of w we can land of diff local minima but we have seen what is global minima and all depends on what initial weights we have initialised.

(7) 3D and Contours (Hill descent)

$w = [w_1, w_2]$



(8) SGD & Momentum:

iter: $(w_{ij}^k)_{new} = (w_{ij}^k)_{old} - \eta\left(\frac{\partial L}{\partial w_{ij}^k}\right)_{old}$

↳ update function

$w \leftarrow w_{ij}^k$

notation for simplicity ←*

$$w_t = w_{t-1} - \eta\left[\frac{\partial L}{\partial w}\right]_{w_{t-1}}$$

t^{th} iteration $t-1^{th}$ iteration

$\eta \leftarrow 0.01$ generally.

but $\frac{\partial L}{\partial w}$?

- $D\{x_i, y_i\}_{i=1}^{n}$ ← size of data
- using all the n points in D → GD → consume so much time & memory.
- using only one one point x_i @ random → SGD
- using a random subset of k points in D → minibatch SGD.

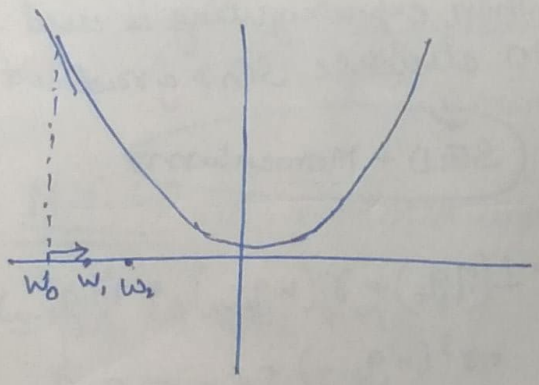
this is what we have discussed earlier for back propagation

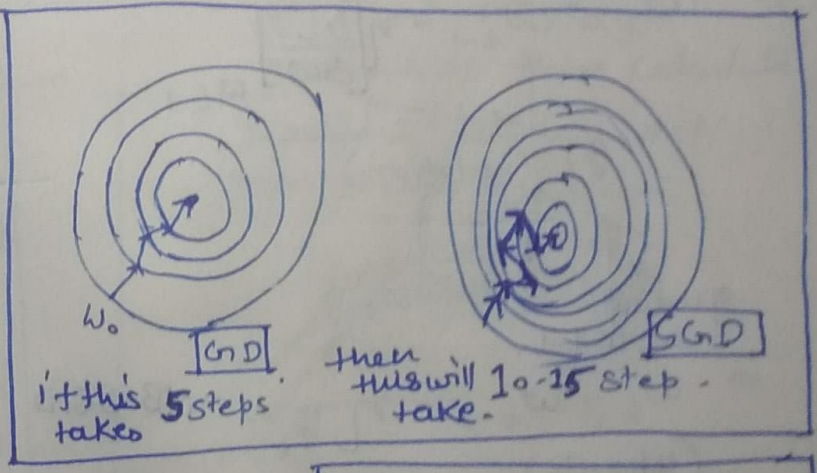
used multiple

While applying mini batch SGD

$$\left[\frac{\partial L}{\partial w}\right]_{\text{minibatch SGD}} \approx \left[\frac{\partial L}{\partial w}\right]_{GD}$$

These are noisy but not equal to $\left[\frac{\partial L}{\partial w}\right]_{GD}$





if this takes 5 steps

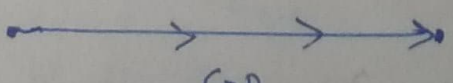
then this will take 10-15 step.

Basically what we are doing is estimating gradient /des using SGD.

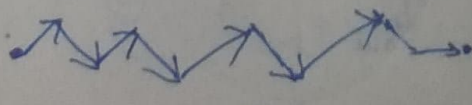
→ GD

w^*_{SGD} result will same as w^*_{GD} but take more iterations

→ each of updates are more in SGD than GD.



Eventually both will reach to the same point.



(Noisy) → but we can remove this noise by using the concept of momentum.

How to denoising gradient in SGD?

① SGD with momentum → change in gradient term
→ No change in learning rate term
→ just add a term.

denoise → take averages.

iter → t_1 t_2 t_3 t_4
val → a_1 a_2 a_3 a_4 a_5 a_6 ---------

CONCEPT

at t_1 $V_1 = a_1$
at t_2 $V_2 = \gamma V_1 + a_2$ ← recent point.

reccursive equation!

$\gamma = 1$
$0 \le \gamma < 1$

Z reacent point has more weight.

oldest point, middle point, newest point

$t_3 = \gamma^2 a_1 + \gamma a_2 + a_3$ also called exponential moving avercage.

Back to SGD: Mini-Batch

* $w_t = w_{t-1} - \eta \left[\frac{\partial L}{\partial w}\right]_{w_{t-1}}$

g_t (notation)

$\therefore g_t = \left[\frac{\partial L}{\partial w}\right]_{w_{t-1}}$

gradient at w_{t-1}.

$\Rightarrow w_t = w_{t-1} - \eta g_t$ → MB-SGD

(update)

No. what is this.

⇒ (exp-weighting)

γ is (0,1) generally 0.9

$v_t = \gamma V_{t-1} + \eta g_t$ ← new momentum in SGD which is added.

$w_t = w_{t-1} - v_t$ ← gradient as old gradient + momentum

Case 1: $\gamma = 0$

$V_1 = \eta g_1$

$w_1 = w_0 - \eta g_1$

Case 2: $\gamma = 0.9 V_{t-1}$

$w_t = w_{t-1} - (0.9 V_{t-1} + \eta g_t)$ ← old

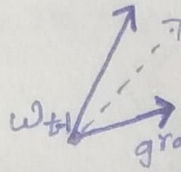
* when exp-weighting is used to denoise SGD gradient

SGD + Momentum

$V_t = 1(\eta g_t) + \gamma(\eta g_{t-1}) + \gamma^2(\eta g_{t-2}) + \gamma^3(\eta g_{t-3}) + \text{-----}$

Core-concept is → removing noise using exp-weight.

momentum ($-\gamma V_{t-1}$) ← it has all the information of past movements of $w_{t-2}, w_{t-3} \cdots w_1$.



$$w_t = w_{t-1} + (-\gamma V_{t-1} - \eta g_t)$$

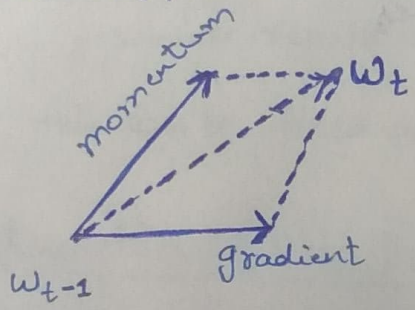
← this the main movement all this point but it has only current information about w_{t-1}.

* SGD + momentum → speedup convergence. and overcome the drawback of slow convergence in

(10) Nesterov Accelerated Gradient (NAG)

No change in learning rate
No change in gradient
just add another term

It is the similar technique as SGD + momentum but with suttle difference and better than SGD + momentum.



$$w_t = w_{t-1} + (-\gamma V_{t-1} - \eta g_t)$$

ηg_t = gradient term

γV_{t-1} = momentum

NAG (who people write)

$$\begin{cases} V_t = \gamma V_{t-1} + \eta g' \\ w_t = w_{t-1} - V_t \end{cases}$$

---

g_t is at w_{t-1}

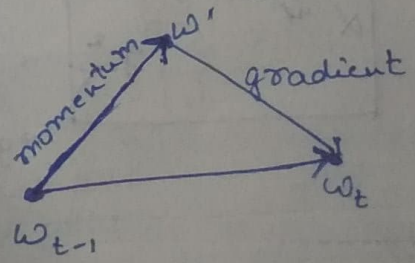
in NAG g'_t is at $w_{t-1} + \gamma V_{t-1}$.

$$g_t = \left(\frac{\partial L}{\partial w}\right)_{w_{t-1}}$$

$$g'_t = \left(\frac{\partial L}{\partial w}\right)_{w_{t-1} + \gamma V_{t-1}}$$

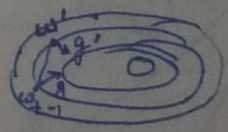
Nesterov AG

It says dont make resultant of momentum + gradient move in momentum then calculate gradient' which is diff. from earlier gradient.



$$w_t = w_{t-1} - (\gamma V_{t-1} + \eta g'_t)$$

$g'_t \neq g_t$



$$g' = \left(\frac{\partial L}{\partial w}\right)_{(w_{t-1} - \gamma V_{t-1})}$$

→ Because we have move in that direction already.

(11) Adagrad (Adaptive gradient) → No change in gradient
→ change in learning rate

→ In SGD & SGD with momentum.
: learning rate ($\eta = 0.01$) → same for each weight (parameter)

$w \rightarrow w_{ij}^{k}$

In Ada-grad → Adaptive gradient.
idea → each weight / parameter has different η.
why?
features → sparse → (BoW) most of them are zero
dense → (less zero values)

SGD:- $w_t = w_{t-1} - \eta g_t$

Adagrad:- $w_t = w_{t-1} - \eta'_t g_t$ ← Now this learning rate is adaptive.
↳ diff η' for each weight @ each iteration.

$\eta'_t = \frac{\eta}{\sqrt{\alpha_{t-1} + \epsilon}}$ ← constant like (0.01)
small +ve number (to avoid divide by zero error)

$\alpha_{t-1} = \sum_{i=1}^{t=1} g_i^2$ → $\left(\frac{\partial L}{\partial w}\right)_{(w_{i-1})}$ → gradient at w_{i-1}
+ve
+ve that's why squaring

$g_1\ g_2\ g_3\ g_4 \dots\dots\dots g_{t-1}\ g_t$
as t increases α_t increases because we are summing the square
(as t ↑) → (α_t ↑) → (η'_t) ↓
Hence learning rate will reduce constantly with iteration.

So, with the increasing iteration, learning rate is decreasing. Convergence in start is fast and slow to avoid overshooting problem.

Advantage & Disadvantage

(+) no need for manually tuning η

$\eta'_t \rightarrow$ weight, time/iter

(+) Sparse & dense feature → Adagrad

disadvantage

(−) α_{t-1} can become very large as t increase.

And $w_t = w_{t-1} - \eta g_t$.

very small because α_{t-1} very large.

which result in $w_t \approx w_{t-1}$.

amount of movementum is so small.

But this can be fixed using next algo.

(12) **Adadelta & RMS prop** → Only change in learning rate
→ NO change in gradient

Adagrad α_{t-1} V. large → Slow Convergence

$$\eta'_t = \frac{\eta (=0.01)}{\sqrt{\alpha_{t-1} + \epsilon}} \; ; \quad \alpha_{t-1} = \sum_{i=1}^{t-1} g_i^2$$

η'_t is becoming small because α_{t-1} is becoming large

α_{t-1} is becoming large because it is summation of g_i^2.

Idea here is take exponentially decaying rate. (eda)

Adadelta: $w_t = w_{t-1} - \eta'_t g_t$.

$$\eta'_t = \frac{\eta}{\sqrt{eda_{t-1} + \epsilon}}$$

$$eda_{t-1} = \gamma \, eda_{t-2} + (1-\gamma) g_{t-1}^2$$

exponential decay average

⇒ γ is hyperparameter typically 0.95.

so $eda_{t-1} = 0.95 \, eda_{t-2} + 0.05 \, g_{t-1}^2$

~~(12) Adam: Adaptive Moment estimation~~

$$eda_{t-1} = (0.05)\, g_{t-1}^2 + 0.95\, eda_{t-2} \quad -(1)$$

$$eda_{t-2} = (0.05)\, g_{t-1}^2 + 0.95\, eda_{t-3} \quad -(2)$$

① & ②

$$Cda_{t-1} = (0.05)\, g_{t-1}^2 + \left[0.95 * \{0.05\, g_{t-2}^2 + 0.95\, eda_{t-3}\}\right]$$

Now eda_{t-1} do not grow v.v. large and v.v. slow η_t'.

$$= \underbrace{0.05 g_{t-1}^2}_{\text{5\% of present weight}} + \underbrace{(0.95*0.05)\, g_{t-2}^2}_{\text{5\% times to prev. top rev. weight}} + \underbrace{(0.95)^2 eda_{t-3}}_{\text{95\% times of present weight}}$$

But doing this we can control the growth of α_{t-1} or named here eda_{t-1}. (denominator)
(like in adagrad) (adadelta)

RMS prop is same as Adadelta but some thing are diff. but idea is same.

(13) Adam: Adaptive Moment estimation. (2018) best

↳ Both learning rate & gradient changed + made something else.

idea: Adadelta → storing exp weight avg of g_t^2 → learning rate (η_t)
(eda)

→ store eda of g_t

In statistic mean → 1st order moment
var → 2nd order moment

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t \quad -(1) \quad 0 \le \beta_1 \le 1 \quad \text{typically } \beta_1 = 0.9$$

$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2 \quad -(2) \quad 0 \le \beta_2 \le 1 \quad \text{typically } \beta_2 = 0.99$$

$eda\, g_t$

$eda\, g_t^2$

$$\hat{m}_t = \frac{m_t}{1-(\beta_1)^t} \quad ; \quad \hat{V}_t = \frac{V_t}{1-(\beta_2)^t}$$

$$w_t = w_{t-1} - \frac{\alpha \hat{m}_t}{\sqrt{\hat{V}_t + \epsilon}}$$

α = 1 generally

→ Adam is best that comes by lot of experiment.
→ No need to remember formula.

If → $\beta_1 = 0$, $m_t = g_t$ } Adadelta

If → $\beta_1 = \beta_2 = 0$

ADAM has all the advantage
- converge fast
- Works for deep Network
- no slow learning rate problem
- No need to manually tune values
- No concept of learning rate directly it automatically adjusts.

(14) Which optimizer to use.

* Mini Bach SGD → Momentum & NAG → Adagrad → Adadelta / RMS prop → Adam

⇒ Small or shallow NN with no saddle point.
if saddlepoint Come in function it will stuck there.

work well in most cases but slow. (slower)

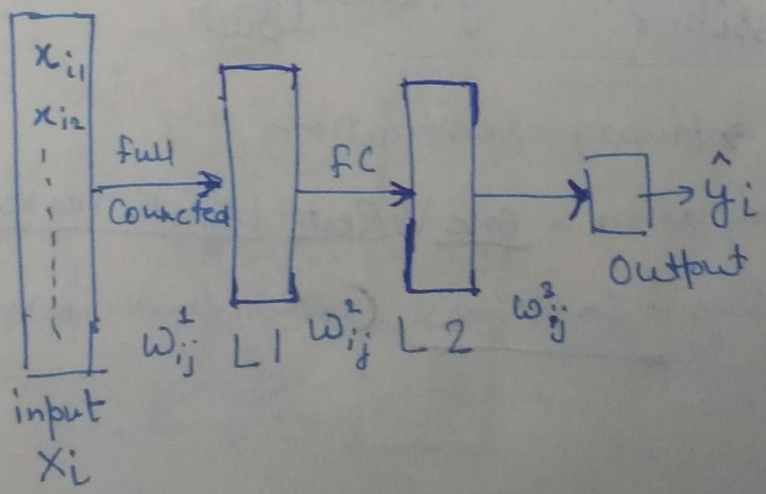
Sparse feature relative fast

fastest algorithm and highly used.
* no need to specify η.

(15) Gradient Monitoring & clipping.

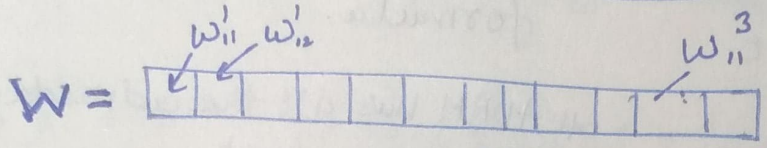
used to update weights.

→ good habit to monitor gradient and update
↓
to detect prob of vanishing gradient

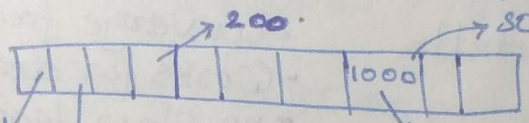


if gradient is large then prob. come will exploding gradient. Then we can visualize it can be used as change the optimization by Relu.

Another solution is (clipping) w_{ij}^{k}



G_1 = Gradient:



$\frac{\partial L}{\partial w_{11}^{1}}$ $\frac{\partial L}{\partial w_{12}^{1}}$ $\frac{\partial L}{\partial w_{11}^{3}}$

then we can do clipping otherwise it will create exploding gradient.

L2 norm clipping

$G_{new} = \frac{G_1}{\|G\|_2} * T$

clipped gradient

Threshold

$\|G\|_2 = \sqrt{G_1^2 + G_2^2 + G_3^2 \cdots}$

each $\frac{\partial L}{\partial w}$ is less than 1 now. if $T=1$ or less than T if T is anything else.

G_1 G_2 G_3 G_4

G_1 = | 2 | 5 | 10 | 10000 | 0.5 | 50 |

$\frac{G_1}{\|G\|_2}$ = | <1 | <1 | <1 | <1 | | |

< 1

$* T$ ~~Set of~~

T is say 2

gradient $\leq T$

this is called

L2 norm clipping of Gradient

and it is done to avoid v.v. large gradient $\left(\frac{\partial L}{\partial w}\right)$.

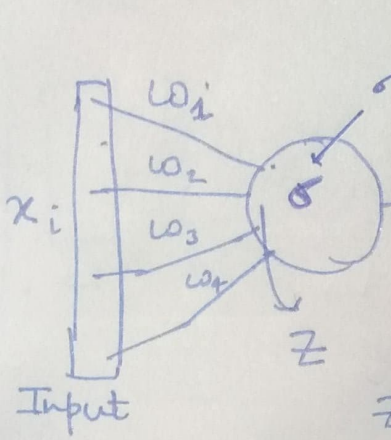
(16) Softmax - classifier:

logistic regression → binary classification

for multiclass classification → one Vs Rest Logistic regression

log. reg + Multi-class = Softmax (rather than one Vs Rest)

Logistic regression recap.



$\hat{y}_i = P(y_i = 1 \mid x_i)$ ← probability of y_i being 1 on given x_i

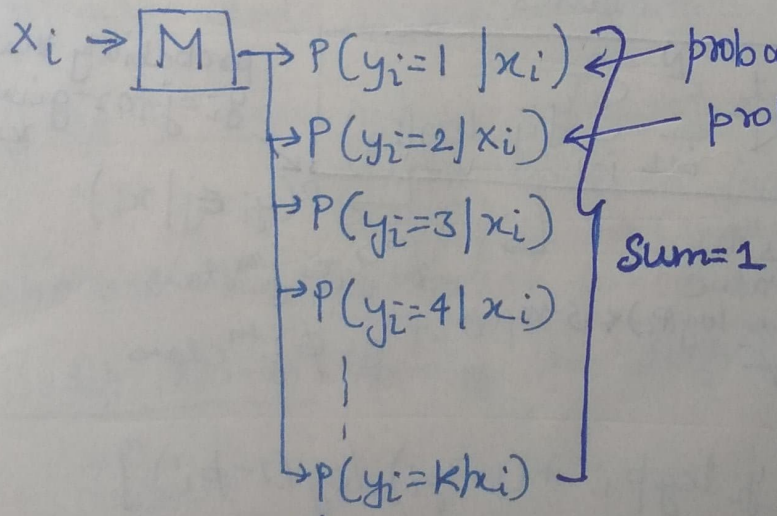
$z = x_i^T w$

$$P(y_i = 1 \mid x_i) = \hat{y}_i = \sigma(z) = \frac{1}{1 - e^{-z}} = \frac{e^z}{e^z + 1}$$

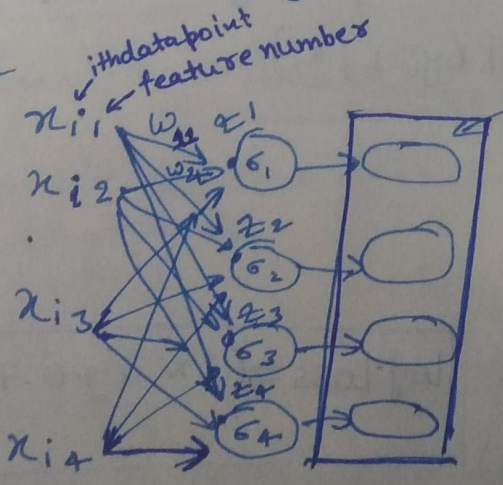
Softmax classifier

$\mathcal{D} = \{x_i, y_i\}$; $y_i \in \{0, 1\}$ in Log. reg.

$y_i \in \{1, 2, 3 \dots K\}$ in Softmax classifier.



Let $K=4$

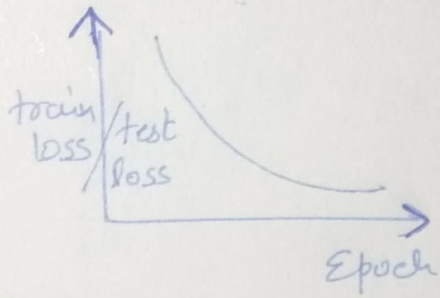


$$z_1 = \sum_{j=1}^{d} x_{ij} w_{j1}$$

$$z_2 = \sum_{j=1}^{d} x_{ij} w_{j2}$$

(8) monitor gradients → apply clipping

(9) plots



this plot is very imp to visualise the train/test loss with number of epochs.

(10) Avoid overfitting } → Deep MLP are easy to overfit in multilayer perceptron.

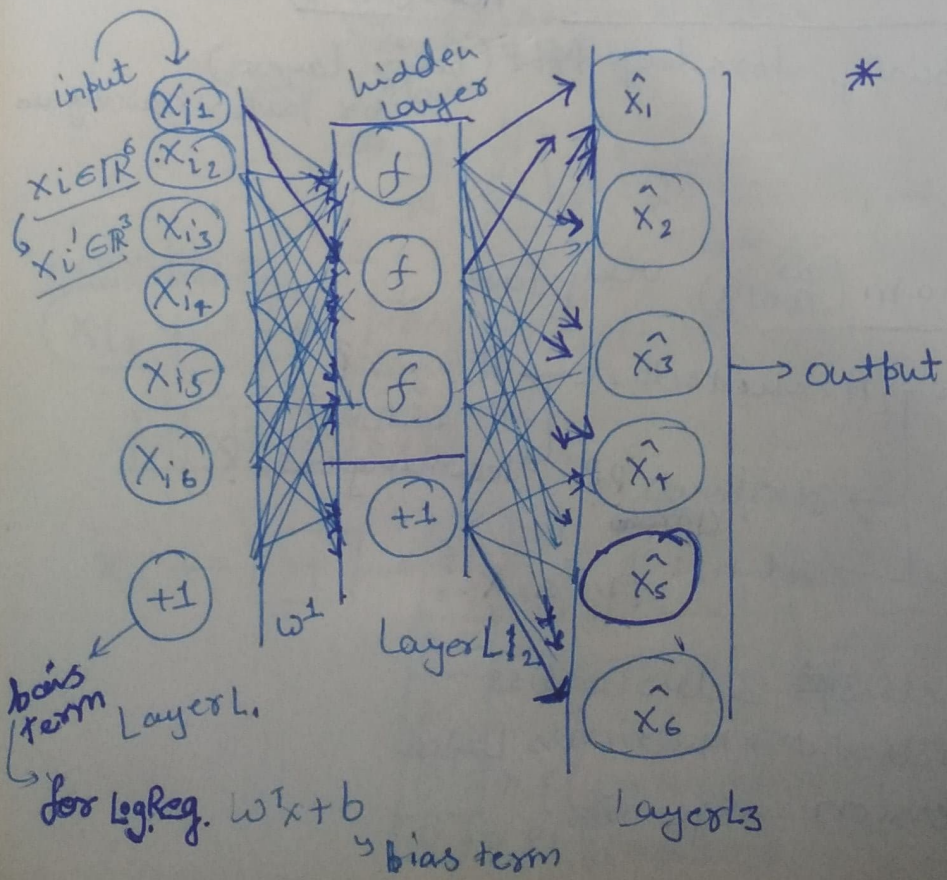
(18) Auto-encoder (AE) → revisit dimension-reduction.

* Neural Network which perform dim-red. like [PCA, tSNE.]
  - PCA → Variance
  - tSNE → Neighbourhood.

* Simple Auto Encoder

$$\mathcal{D} = \{x_i\}_{i=1}^{n} \quad x_i \in \mathbb{R}^d$$

$$\mathcal{D}' = \{x_i'\}_{i=1}^{n} \quad x_i' \in \mathbb{R}^{d'} \quad d' < d$$



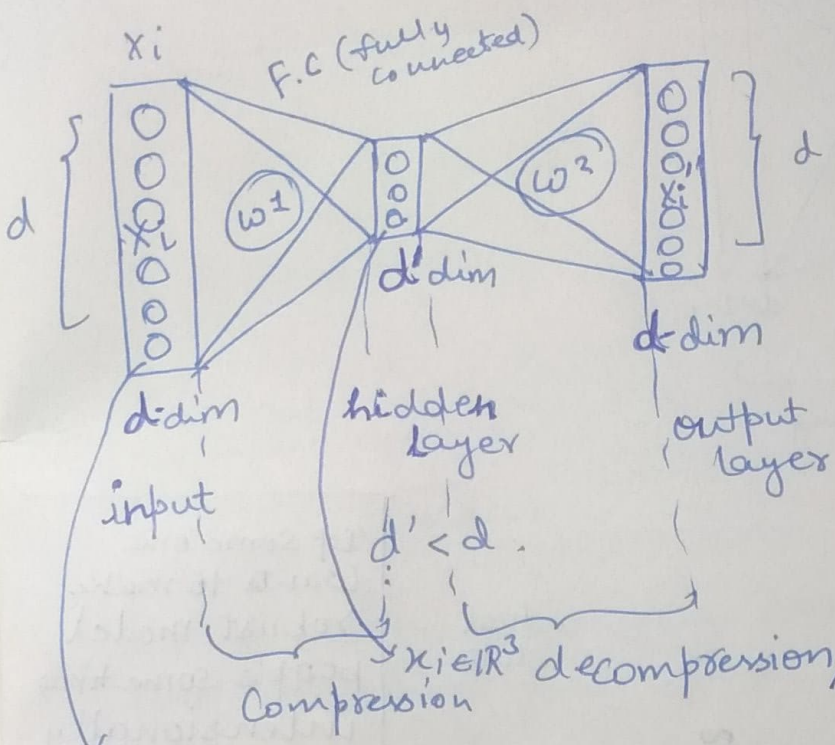
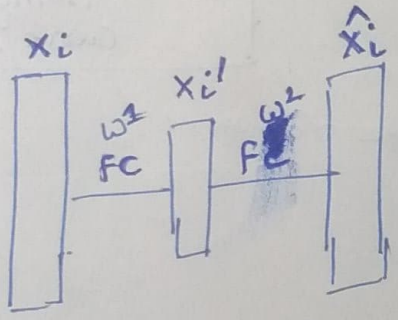


fig - Auto Encoder with 1 hidden layer

$x_i \in \mathbb{R}^6$ *Here 6 dim are compressed to 3 dim & then all these 6 dim can be obtained from 3 dims.

Actual NN of Auto Encoder



Q. so what we want from autoencoder??

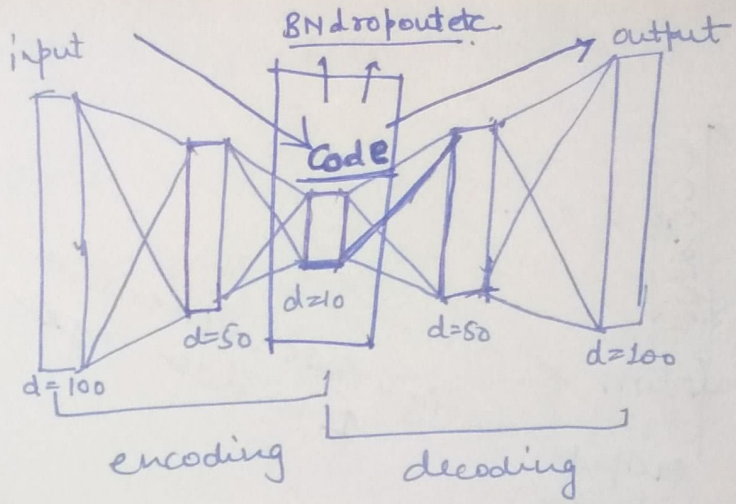
→ want $\hat{x}_i \approx x_i$ → decomparision after Comprasion

$$L(x_i, \hat{x}_i) = \underbrace{\|x_i - \hat{x}_i\|^2}_{\text{distance loss}}$$ ← lets say squared loss / distance loss

① differential, so we can minimise it or do optimisation.

fun part is $x_i' \overset{d'}{<} x_i \overset{d}{}$ → so we need not create v.v. deep model.

Layer wise pretraining very popular but now Adam is used.



Denoising Auto Encoding

$D = \{x_1, x_2, x_3 \dots x_n\}$ ← actual data

we get → $\tilde{D} = \{\tilde{x}_1, \tilde{x}_2, \tilde{x}_3, \dots \tilde{x}_m\}$ ← noisy & corrupt

$\tilde{x}_1 \rightarrow [AE] \rightarrow y_i \in \mathbb{R}^{d'}$

$d' < d$

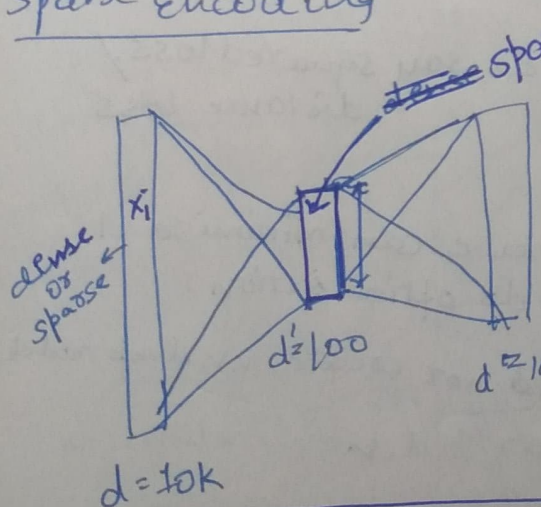
automatically noise will be removed.

If some one wants to make robust model people sometimes intensionally add noise

$\tilde{x}_i = x_i + \tilde{N}(0, \sigma)$

↑ random value from gaussian curve.

Sparse Encoding



→ use L_1 regularizer on x_i during loss function.

★ if Activation function used is sigmoid output is much similar to PCA with 1 hidden layer.

19 Word 2 Vec → Amazon fine food reviews.

{W2V(w) → vectors}

not a deep learning algo. but see the NN explaination.

def: C_5 C_4 **f** C_1 C_2 C_3

The cat **sat** on the wall.

Context words — focus word — Context words

Core Id is context words are very useful in understanding focus word.

W2V

2-algorithm (popular) — C-BOW (continous bag of words) / skip gram

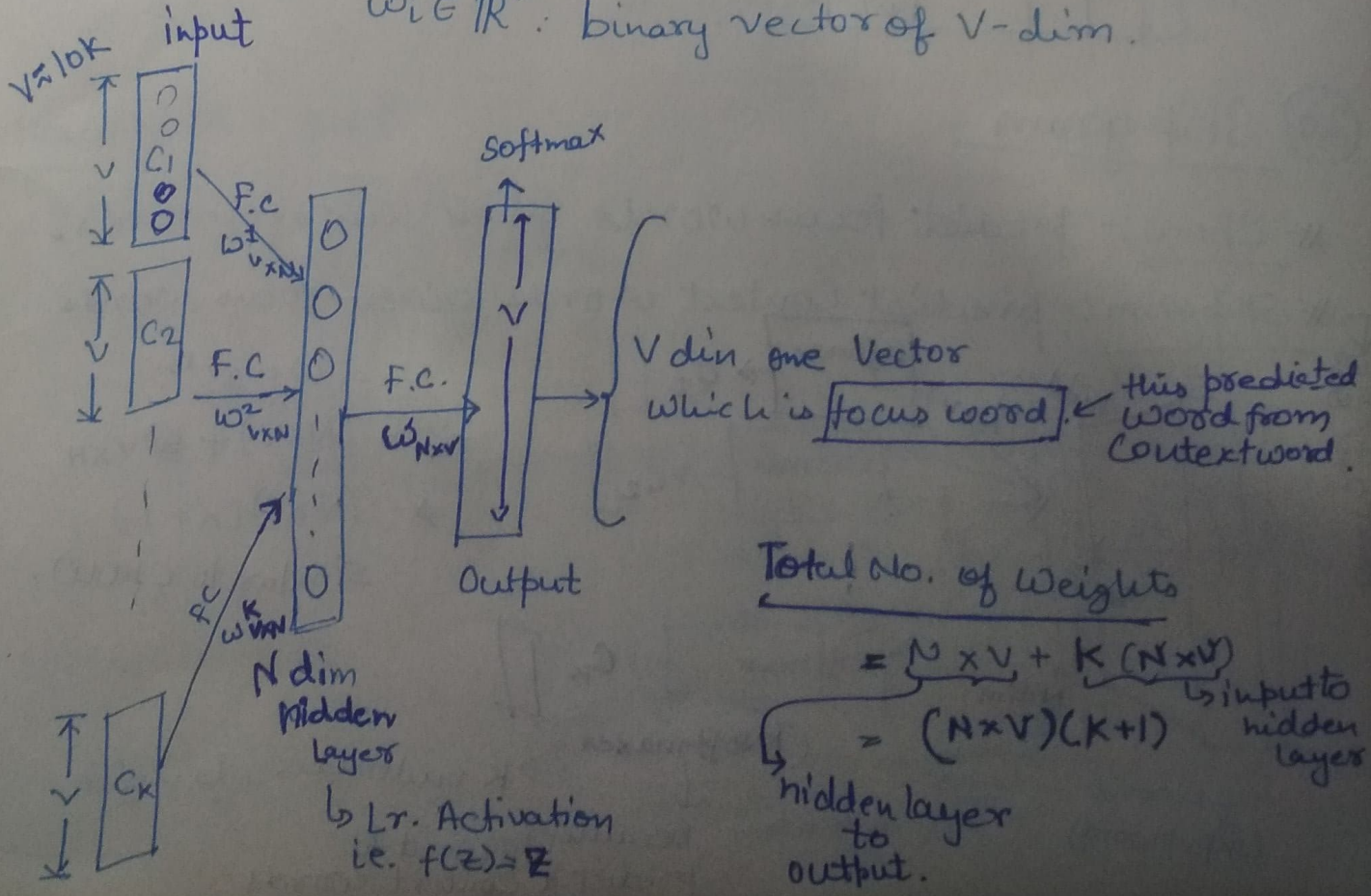
CBOW → given context word predict focus word.

① dictionary/Vocablary of words = V

V = length/size of Vocals

② One-hot encoding

$w_i \in \mathbb{R}^V$: binary vector of V-dim.



↳ Lr. Activation i.e. $f(z) = z$

Total No. of Weights

$= N \times V + K(N \times V)$ ↳ input to hidden layer

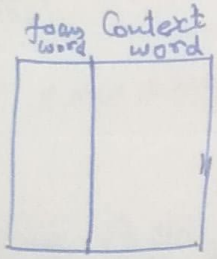
$= (N \times V)(K+1)$

↳ hidden layer to output.

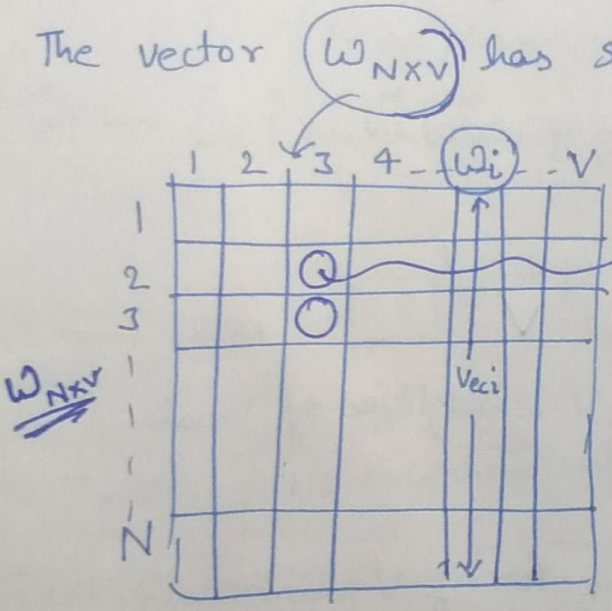
Train? CBOW

text

* Make combination of focus words & context words. that becomes training data.



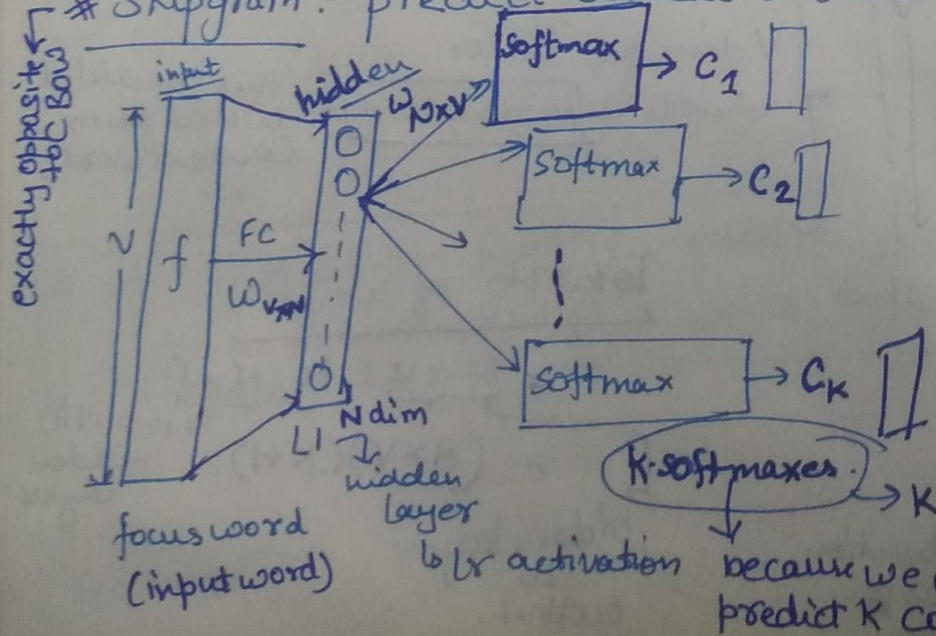
The vector W_{NXV} has summary of $W^1_{VXN} + W^2_{VXN} \cdots + W^K_{VXN}$



(2.0) Skip gram

* CBOW:- predict focus words given Context words.

* Skipgram:- predict Context words given focus words.



No. of weights:-

$K(N \times V) + V \times N$

$\Rightarrow (N \times V)(K+1)$

Similar to CBOW.

In both wo. weight are same but

In CBOW → 1-Softmax
In Skipgram → K-softmax } → Skipgram is computationally more expensive (takes more time)

CBOW ⊕ faster to train than skipgram
⊕ better for frequently occusing word (more variety of data for same focus word so better training)

Skipgram → ⊕ can work with less amount of data
⊕ better with less freq words also.

For both skipgram & CBOW →
lets say 'k' → # Context words
K↑ ⇒ more context → N-dim for each pt is better.

How hard is this?

weights = (K+1)(N×V)
≈6 ↓ ↳ ≈10k
×200

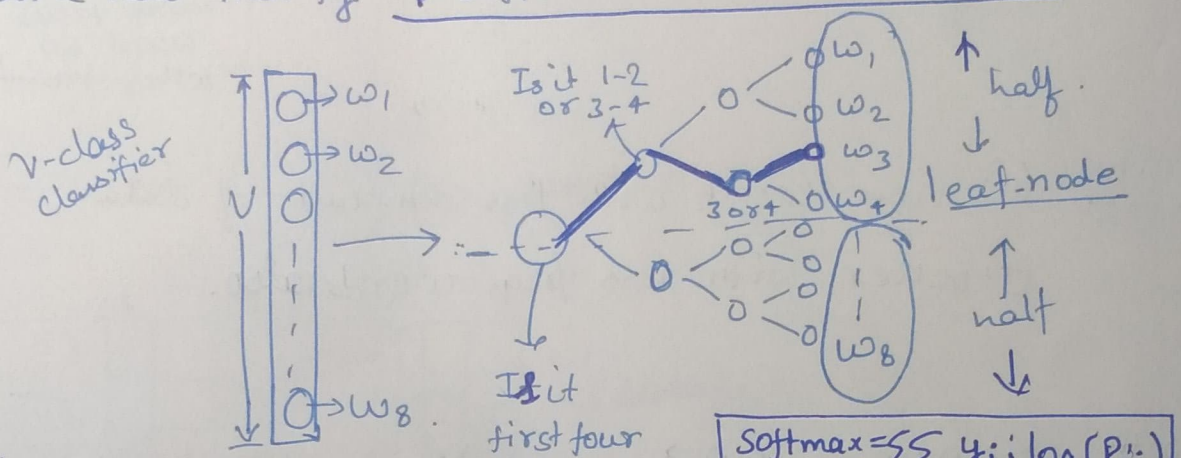
= 1.2K × 10K
= 12 Million (impossible) for normal Computers.
So we need to optimise this??
We see How they use algorithms to optimise this?

(21) Algorithm Optimization.

Algorithms.
- CBow & skipgram → Million of weights.
  - (1) hierarchical softmax (algo)
  - (2) Negative sampling (stat)

(1) Heirarchical softmax.

Some how modify V-softmax to make it optimal

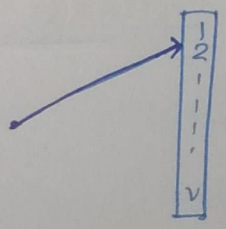


$$\text{softmax} = -\sum y_{ij} \log(P_{ij})$$

What we want is $P(y_i = 3 | x_i)$ rather than checking for all the 8 units we use some tree kind of structure which goes to path 1-4 → 3-4 → 4 i.e. (3) computions only.

So instead of 8 activation we need 3 activation ie $\log(V)$ ← complexity now.

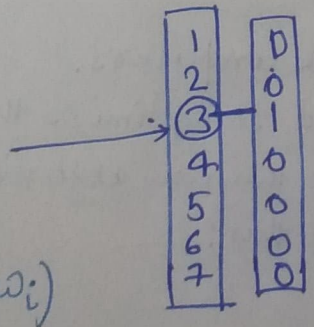
(2) Negative Sampling (stats)



{update only a sample of words per iteration}

(1) always keep the target word.

(2) Amonst non-target word update sample of them



$$P(w_i) = 1 - \sqrt{\frac{t}{freq(w_i)}} \quad t = 10^{-5}$$

if word is very freq. $\sqrt{\frac{t}{freq}}$ is small $P(w_i)$ is large

for more study→
https://blog.acolyer.org/2016/04/21/the-amazing-power-of-word-vectors/