48.1 Deep multi layer Perceptrons : 1980's to 2010's

1980's, 1990's, 2006
- → 2-3 layered n/w
- → Vanishing gradient → (1)
- → too little data → overfit → (2)
- → too little compute → too much time → (3)

Early 2010 :- → we had lots of data
- ↳ we had labelled data (because of internet companies)
- ↳ we have a new type of compute infrastructure
  - ↳ GPU
    - ↳ v.v. good & super suitable for deep learning
- ↳ new ideas & new algorithms

classical ML → SVM (90's)
→ RF (2000's)

{ Theory (1), Experiments (2).

modern ML

Experiment (1) & Theory (2)

48.2 Dropout layers & Regularization

Deep NN $\longrightarrow$ overfitting $\xleftarrow{L_1, L_2}$

↳ many layers ⇒ many weights

RF: we are doing randomization of features to create regularization.

RFR (Randomization for Regularization) ⟶ NN (MLP's)

dropout rate : $0 \leq p \leq 1$

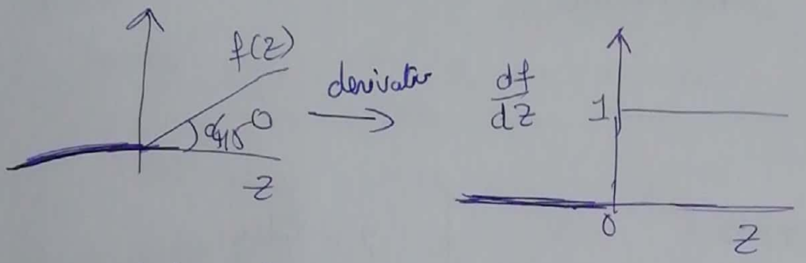
→ If I say $P = 0.2$, in every layers 0.2 neurons will be dropped out

→ dropout ≈ random subset of features

483 Rectified Linear Units (ReLU) → $\boxed{\text{Converge faster}}$

As of 2018 → Best activation
↳ default activation in MLP

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



← slop = 0

Relu: not differentiable at zero

$\frac{d f_{Relu}}{dz} \in \{0, 1\}$ → we don't have any value called Exploding gradient or Vanishing gradient

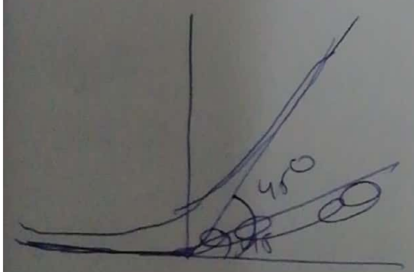
→ As we have zero we will have dead activations

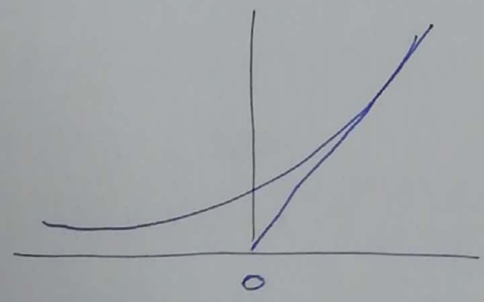
A smooth approximation of ReLU is the softplus function

$f(x) = \log(1 + \exp x)$.

The derivative of softplus is $f'(x) = \frac{\exp(x)}{1 + \exp x} = 1/(1 + \exp(-x))$

which is a logistic function





$$\frac{d f_{Relu}}{dz} = \begin{cases} 0 & \text{if } z < 0 \\ 1 & \text{if } z > 0 \end{cases}$$

↗ Soft plus

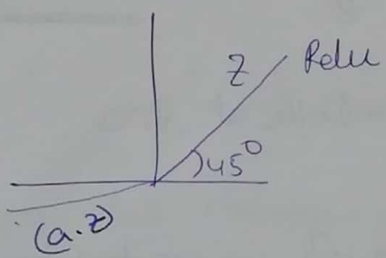
Noisy ReLU's

~~Leaky~~ $f(x) = \max(0, x + \Psi)$ with $\Psi \sim N(0, \sigma(x))$

adding a random value from a gaussian distribution with some variance

Leaky ReLu's

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



a = hyperparameter

↳ This can lead to vanishing gradient

18.4 # Weight Initialization : Deep MLP

→ Logistic Regression → inialize weight randomly

↳ $W_{ij}^{k} \sim N(0,\sigma)$

uniform random   Gaussian random

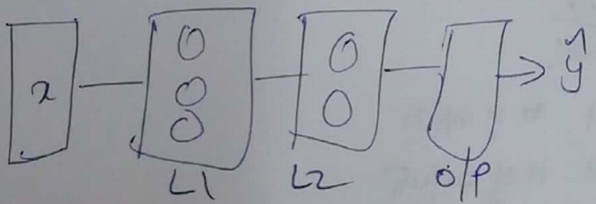
Idea 1

iniatialize $w_{ij}^{k} = 0 \; \forall \; i,j,k$ → v.v. bad.
(∀ forall)

→ f_{ij} : Relu
→ all neurons compute the same thing
→ Same gradient updates happen to all neurons } Symmetry

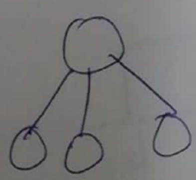
If we initialize either 0 & 1 & 2 same problem as below (Symmetry)

→ we want models to be asymetry



Ensemble : more different the base models are, the better will be the output of Ensembling

Stacking



Idea 2

Initalize w_{ij}^{k} = large -ve numbers

Relu $w^T x = z \equiv$ large -ve value $\approx F(z) = 0$ (relu)

normalized < mean centering, variance scaling

* data normalization is mandatory

Solution

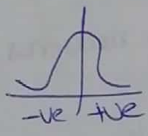
Idea 1

- → weights should be small (not too small)
- → not all zero
- → good-variance ← $Var(w^k_{ij})$
- → All weights come from a ~~stan~~ Gaussian distribut

$$w^k_{ij} \sim N(0, \sigma) \quad (\sigma : \text{is reasonably small})$$

① σ^2

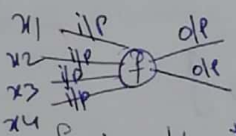
② $\underbrace{+ve, -ve}_{\text{centered at } 0}$



better init strategy

→ fan-in

→ fan-out



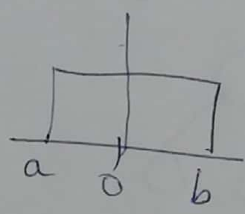
fanin = 4 # of inputs

fanout = 2 # of outputs

Uniform Initialization (work well for sigmoid)

$$w^k_{ij} \sim unif\left[\overset{a}{\frac{-1}{\sqrt{fanin}}}, \overset{b}{\frac{1}{\sqrt{fanout}}}\right]$$

$\frac{-1}{\sqrt{fanin}}$ ⇒ min value; $\frac{1}{\sqrt{fanout}}$ ⇒ max value



→ no concrete agreement amongst all researchers about best initializa

Xavier/Glorot init (works well for sigmoid)

normal

a) $w^k_{ij} \sim N(0, \sigma)$ $\quad \sigma_i = \sqrt{\frac{2}{fanin + fanout}}$

uniform

b) $w^k_{ij} \sim U\left[\frac{-\sqrt{6}}{\sqrt{fanin+fout}}, \frac{+\sqrt{6}}{\sqrt{fanin+fanout}}\right]$

for each neuron fan in and fan out will change

He-initialization (2015) (Good for ReLu)

a) Normal

$$w_{ij}^{k} = \sqrt{\frac{2}{fanin}}$$

b) uniform

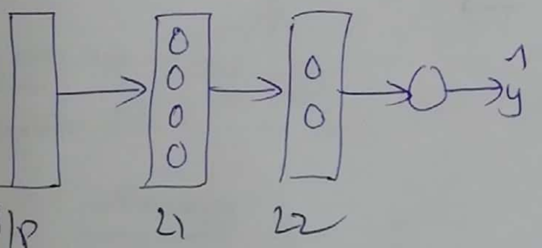
$$w_{ij}^{k} \approx U\left[\frac{-\sqrt{6}}{\sqrt{fanin}}, \frac{+\sqrt{6}}{\sqrt{fanin}}\right]$$

48.5 Batch normalization (2015)

$D = \{x_i, y_i\}$

Preprocessing : Data normalization (x_i)
↳ mean Centering
↳ Var. scaling

$\tilde{x}_i = \frac{x_i - \mu}{\sigma}$ $\mu = mean(x_i)$
$\sigma = stddev(x_i)$



→ fully Connected

→ fully Connected MLP

$\underbrace{x \rightarrow L1 - L2 - L3 \rightarrow L4 - L5}$ − output − $\hat{y}$

mini batch ↑↑ small change

↑ large changes

A small change in i/p will lead to a large change in the output especially if we have deep Neural N/W

→ batch normalization layer

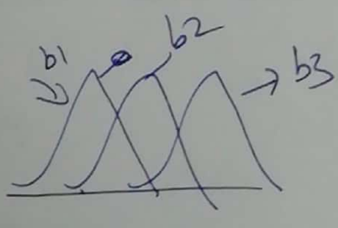
$x \rightarrow L1 - L2 - L3 - L4 - \boxed{BN} - L5 - O - \hat{y}$

b1 ↗
b2 ↗
b3 ↗

$b_1 = \{x_1, x_2 \cdots x_k\}$

⇓

Normalization



→ Batch normalization will work better when it is placed deep in the N/W.

$B_i \xrightarrow{x_i} \boxed{\begin{matrix} BN \\ \gamma, \beta \end{matrix}} \to \tilde{x}_i\ (\tilde{B})$

$\mu_B \leftarrow \frac{1}{m} \sum_{i=1}^{m} x_i$ // mini batch mean.

$\sigma_B^2 \leftarrow \frac{1}{m} \sum_{i=1}^{m} (x_i - \mu_B)^2$ // mini batch variance.

$\hat{x}_i \leftarrow \frac{x_i - \mu_B}{\sqrt{\sigma^2 + \epsilon}}$ → small value to avoid division by zero Error.

$\tilde{x}_i \leftarrow \gamma x_i + \beta \equiv BN_{\gamma,\beta}(x_i)$ // scale & shift

scale shift

Advantage

→ Helps to have faster Convergence

→ Weak regularizer

Bn + dropout

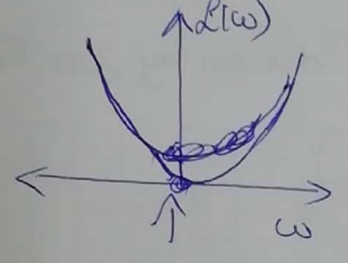
→ Internal covariate shift → we can train deep NN.

48.6 Optimizers: Hill-descent analogy in 2D

Lr. Reg & optimisation → GD, SGD, mini batch-SGD

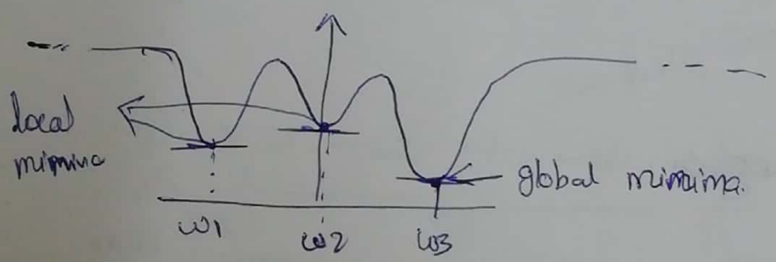
$$\min_{w} \mathcal{L}(w)$$

(1) If w is scalar



- There is no maxima

→ There is only one minima



At both minima/maxima } $df/dw = 0$

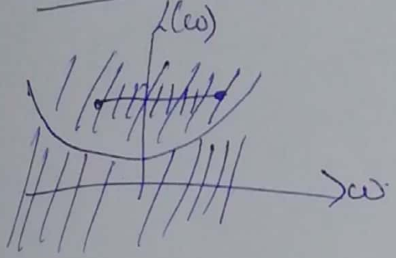
If $\frac{dL}{dw} = 0$ we can be either at minima/maxima/saddle point

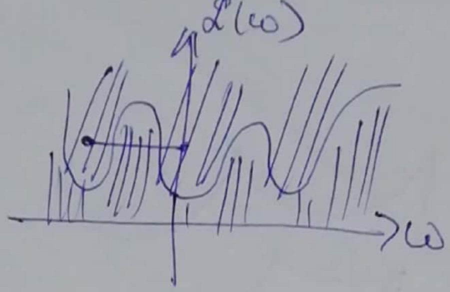
$$w_{new} = w_{old} - \eta \frac{dL}{dw}$$

we will keep updating w until $\frac{dL}{dw}$ become zero, and this can be zero at minima/maxima/saddle point

→ SGD/mini batch SGD can get stuck at a saddle point

Convex functions & non Convex functions





Convex functions has only one minima & one maxima.

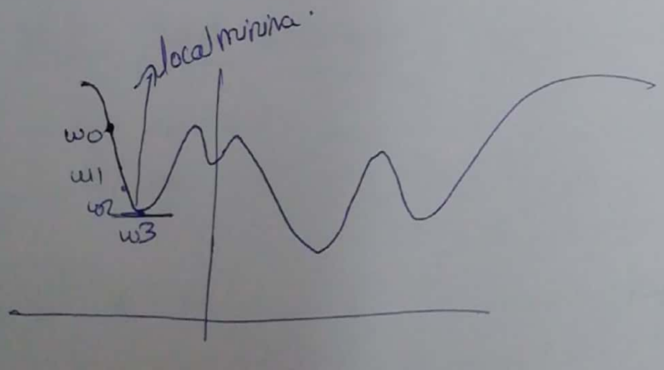
↳ local minima = Global minima

V.V.Imp

Lr. reg, lr. reg, SVM all of them the loss function can be shown as convex function. → local minima = global minima.

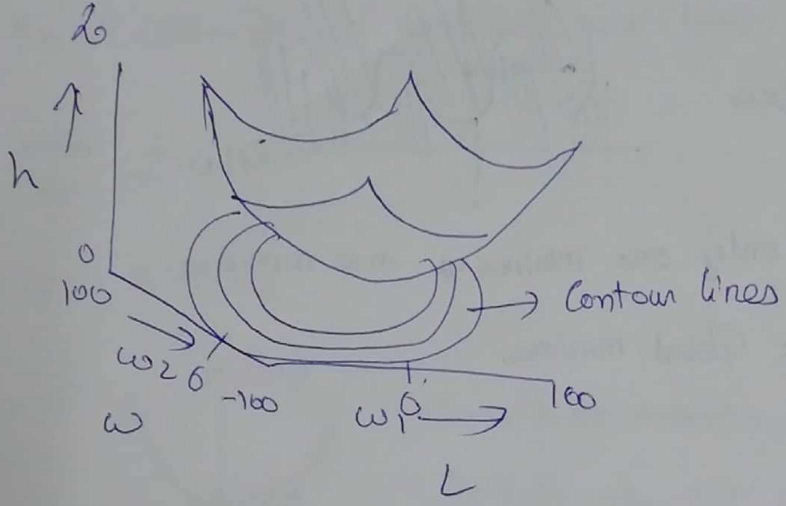
DL & MLP

↳ non Convex function → multiple local minima & can stuck at saddle point



→ Base on initial weights we can land up at different minima.

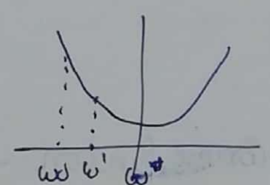
48.7 optimizers: Hill descent in 3D & contours



48.8 SGD Recap

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

update function



$w \longrightarrow w^k_{ij}$

$\eta \to 0.01$

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

$D = \{x_i, y_i\}_{i=1}^{n}$

$\frac{\partial L}{\partial w}$ → using all the n pts in D → GD

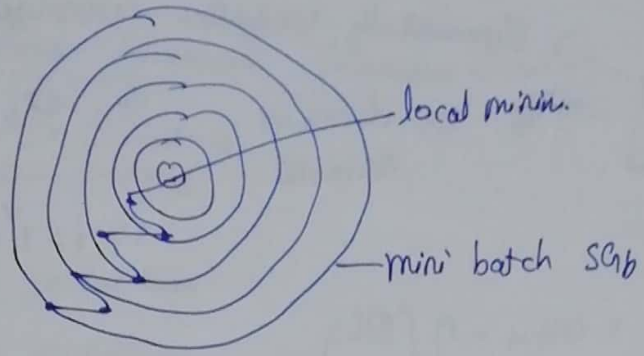
↳ using only one pt x_i @ random → SGD

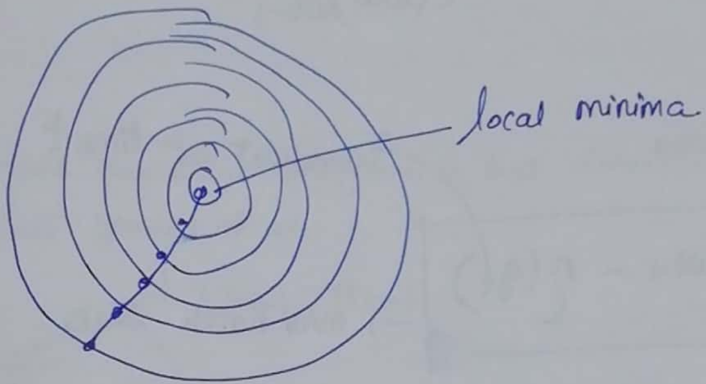
↳ using a random subset of k pts in D

↳ mini-batch SGD

mini batch SGD

noisy → $\left(\frac{\partial L}{\partial w}\right)_{min\_SGD} \overset{approx}{\approx} \left(\frac{\partial L}{\partial w}\right)_{GD}$





→ Each of the updata are more noisy in SGD & MBSGD than GD

→ denoising gradient from SGD to Converge faster

48.9 Batch SGD with Momentum

$t=1 \quad t=2 \quad t=3 \quad t=4$ } value at specific time.

$a_1 \quad a_2 \quad a_3 \quad a_4 \quad a_5 \quad a_6 \ldots$ } number

t=1 $\quad v_1 = a_1$

$t=2 \quad v_2 = \gamma v_1 + a_2 \quad (\gamma = 1)$

$(0 \leq \gamma \leq 1)$

$v_2 = v_1 + a_2$

$= a_1 + a_2$

if $\gamma = 0$

$v_2 = a_2$

$\gamma = 0.5$

$v_2 = 0.5 v_1 + a_2$

$= 0.5 a_1 + 1 a_2$

giving more weight to the new points

less weight to old points

$V_1 = a_1$

$V_2 = \gamma v_1 + a_2$

$V_3 = \gamma V_2 + a_3$

$= \gamma(\gamma v_1 + a_2) + a_3$

$= \gamma^2 v_1 + \gamma a_2 + a_3$

$\boxed{\text{If } \gamma = 0.5} \Rightarrow 0.25 a_1 + 0.5 a_2 + a_3$

$t=1 \quad t=2 \quad t=3$

$V_4 = 0.125 a_1 + 0.25 a_2 + 0.5 a_3 + a_4$

$$\left.\begin{aligned} v_1 &= a_1 \\ v_t &= \gamma v_{t-1} + a_t \end{aligned}\right\} \begin{aligned} &\rightarrow \text{Exponentially weighted averages.} \\ &\rightarrow v_t \approx \text{denoised Estimate} \end{aligned}$$

↑ changes over time.

$$v_t = 1 a_t + \gamma a_{t-1} + \gamma^2 a_{t-2} \dots$$

$$1 \geq \gamma > \gamma^2 \geq \gamma^3 \geq \gamma^4 > \gamma^5 \dots$$

MB-SGD → $w_t = w_{t-1} - \eta \left(\frac{\partial L}{\partial w}\right)_{w_{t-1}}$

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

$$\boxed{w_t = w_{t-1} - \eta(g_t)}$$ → mini batch SGD

$v_t = \gamma v_{t-1} + \eta g_t$ → using exponential weight average.

$w_t = w_{t-1} - v_t$

Case 1 $\gamma = 0$; $v_t = \eta g_t \Rightarrow w_t = w_{t-1} - \eta g_t$

People typically use $\gamma = 0.9$

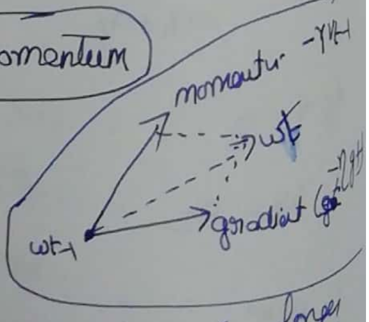
Case 2: $\gamma = 0.9 \Rightarrow v_t = 0.9 v_{t-1} + \eta g_t$; $w_t = w_{t-1} - (0.9(v_{t-1}) + \eta g_t)$

when we use Exponential weight to denoise your SGD gradient

↓

SGD + Momentum

$$w_t = w_{t-1} - [\underbrace{\gamma v_{t-1}}_{\text{Momentum}} + \underbrace{\eta g_t}_{\text{gradient}}]$$

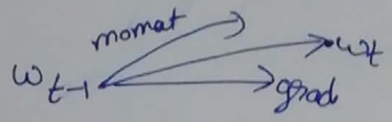


→ momentum will make longer steps to reach minima

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

SGD + momentum → Speed up Convergence

48.10 Nesterov Accelanate Gradiant (NAG)



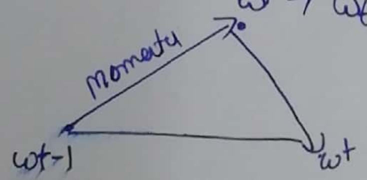
ηg_t -- gradiat term

γv_{t-1} = monentum

NAG

1) First Compute momentum and move in that direction, & Compute w' and Compute gradiat at w'

$w' \rightarrow w_{t-1} - \gamma v_{t-1}$



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

$$g' = \left(\frac{\partial L}{\partial w}\right)_{w'}$$

$$w' = w_{t-1} - \gamma v_{t-1}$$

$$\boxed{v_t = \gamma v_{t-1} + \eta g'}$$

$$\boxed{w_t = w_{t-1} - v_t}$$

48.11 Optimizers : Ada Grad

In SGD & SGD + Momentum → learning rate ($\eta = 0.01$) → same for each weight

Key idea of Ada Grad

Each weight/parameter has a different η

↳ why?

feature → Sparse → (BOW) → few times we can see non zero

↳ dense → (W2V)

Simple SGD

$w_t = w_{t-1} - \eta g_t$ → Same for all weights

Adagrad

$w_t = w_{t-1} - \eta'_t \, g_t$

different η for each weight @ each iteration t it changes

$$\eta'_t = \frac{\eta^{\to 0.01}}{\sqrt{\alpha_{t-1} + \epsilon}}$$

small +ve number to avoid division by '0'

$$\alpha_{t-1} = \sum_{i=1}^{t-1} g_i^2 \rightarrow \left(\frac{\partial L}{\partial w}\right)_{(w_{t-1}^i)}$$

↳ always +ve as it is a square

Computing $g_1, g_2, g_3 \cdots g_{t-1}$

$\alpha_t \geq \alpha_{t-1}$

as $t \uparrow \rightarrow \alpha_{t-1} \uparrow \rightarrow \eta'_t \downarrow$

As iteration increases, learning rate for that weight is decreasing

It uses all the information in the previous gradients

(+) no need to tune η for each iteration

(+) Sparse & dense features

(-) α_{t-1} can become very large as $t \uparrow$

4.8.12 Optimizers : Ada delta & RMS Prop

Adagrad : α_{t-1} V. large $\rightarrow$ slow convergence

$$\eta'_t = \frac{\eta(=0.01)}{\sqrt{(\alpha_{t-1})+\epsilon}} \; ; \; \alpha_{t-1} = \sum_{i=1}^{t} g_i^2 \rightarrow \left(\frac{\partial L}{\partial w}\right)_{(w_{t-1})}$$

Gradient squares

Ada delta

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

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

Controls the growth of (It should grow but it should not become large)

Exponential decaying average.

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

$$\boxed{\text{Typically } \gamma = 0.95}$$

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

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

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

Ada delta : Exp. weighted avg of g_i^2 instead of sum. of g_i^2 (Adagrad) so as to avoid large denominator in η'_t

$\downarrow$

So we avoid slow convergence.

48.13 : <u>Adam</u> (Adaptive Moment Estimation) (Best algorithm)

<u>idea</u> : Adadelta → storing Exp. weight. avg of g_t^2 → learning rate (η'_t)

↳ what if we store Eda of g_t

<u>In statistics</u>

mean → 1st order momentum

Var → 2nd order momentum

$$m_t = \underset{\to 0.9}{\beta_1} m_{t-1} + (1-\beta_1) g_t \quad \to (1) \qquad 0 \leq \beta_1 \leq 1$$

mean at time 't'

$$v_t = \underset{0.99}{\beta_2} m_{t-2} + (1-\beta_2) g_t^2 \quad \to (2) \qquad 0 \leq \beta_2 \leq 1$$

$$m'_t = \frac{m_t}{1-(\beta_1)^t} \; ; \; v'_t = \frac{v_t}{1-(\beta_2)^t} \quad \Big\} \text{ Bias correction.}$$

$$\boxed{w_t = w_{t-1} - \frac{\alpha \, m'_t}{\sqrt{v'_t + \epsilon}}}$$

(1)

If $\beta_1 = 0 \Rightarrow$ Ada delta

If $\beta_1 = \beta_2 = 0 \Rightarrow g_t^2$ will remain.

48.14 Which algorithm to choose when ?

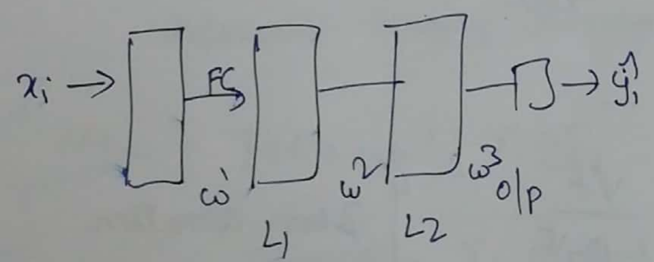
Momentum

MB-SGD → NAG → Adagrad → Adadelta → Adam.

- MB-SGD: Small/shallow NN
- NAG: works well in most cases (slower)
- Adagrad, Adadelta: Sparse
- Adam: most favorite, fastest

48.15 Gradient monitoring & clipping

→ update weights



→ monitor gradients & updates

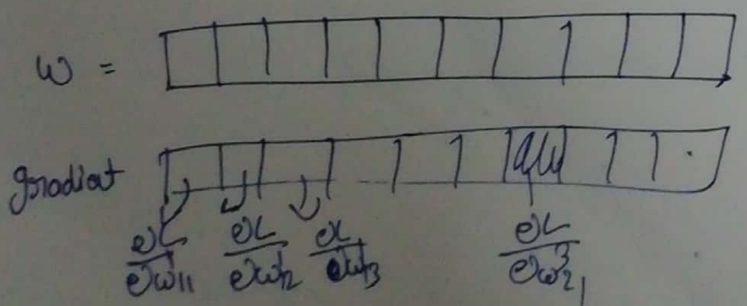
→ Each epoch

→ layer

→ If we found Vanishing Gradient problem, immediately we should change the activation function to ReLU

→ Exploding gradient

Sol: clipping



L2 norm clipping

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

τ threshold

	G_1	G_2	G_3	G_4	G_5
$G =$	2	5	10	100	0.5

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

$$\frac{G}{\|G\|} = \underbrace{[<1 \mid <1 \mid <1 \mid <1 \mid <1]}_{<1} * \tau \quad (\tau = 2)$$

48.16 : Softmax and Cross Entropy for multi class classification

Softmax - classifier

$\hookrightarrow$ logistic reg $\rightarrow$ binary classification

$\hookrightarrow$ multiclass $\rightarrow$ one vs Rest

{Log. reg + multiclass} = Softmax.

x_i — w_1, w_2, w_3, w_4 — $(\sigma) \rightarrow z$ — $\rightarrow \hat{y}_i = P(y_i = 1 | x_i)$

$$z = w^T x_i$$

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

$$= \frac{e^z}{e^z + 1}$$

Soft max

$D = \{x_i, y_i\}$

$y_i \in \{1, 2, 3, 4, 5, 6, 7, 8, 9 \ldots k\}$

$x_i \rightarrow \boxed{m} \rightarrow$

$P(y_i = 1|x_i)$

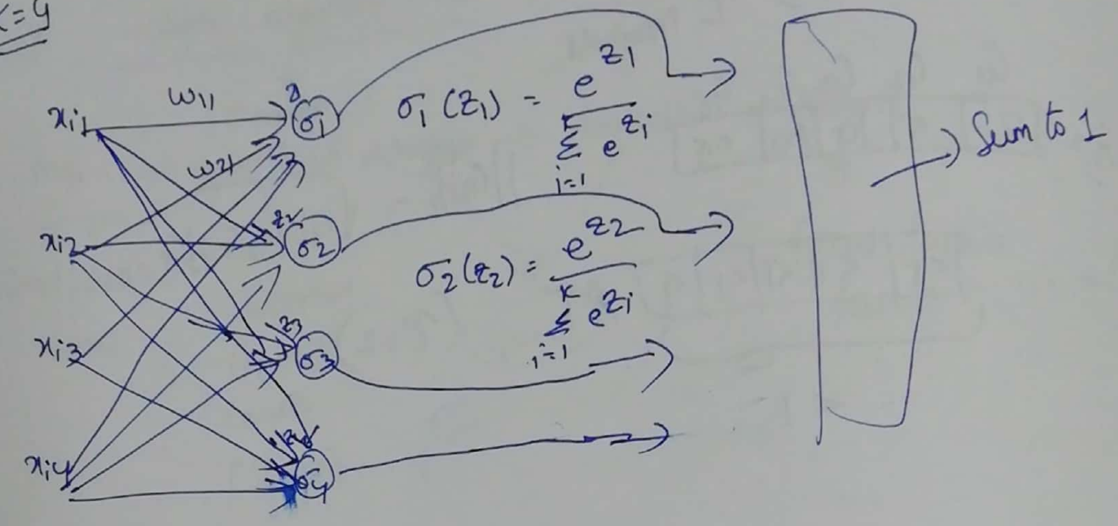
$P(y_i = 2|x_i)$

$\vdots$

$P(y_i = k|x_i)$

sum to 1

$\underline{K=4}$



$$z_1 = \sum_{j=1}^{d} x_{ij} w_{j1} + b, \quad z_2 = \sum_{j=1}^{d} x_{ij} w_{j2} + b$$

Softmax

↳ generalization to LR to "multi class" setting.

→ minimize binary class log loss.

$$LR: \sigma(z) = \frac{1}{1+e^{-z}} = \frac{e^z}{e^z+1}$$

$$\text{Softmax}: \sigma_1(z_1) = \frac{e^{z_1}}{\sum_{i=1}^{k} e^{z_i}} \; ; (\sigma_1, \sigma_2 \ldots \sigma_k)$$

minimize $\underbrace{\text{multiclass log loss}}$

↳ cross Entropy

Ex

n pts

k class

$$-\frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{k} y_{ij} \log p_{ij} \qquad \begin{cases} 1 & \text{if } y_i \in j \\ 0 & \text{otherwise.} \end{cases}$$

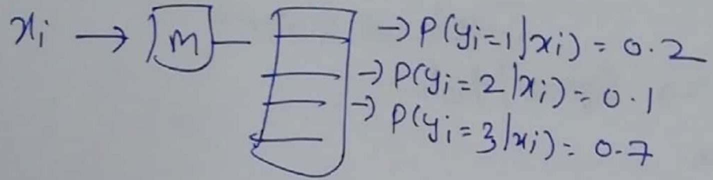
↳ $p(y_i \in j | x_i)$

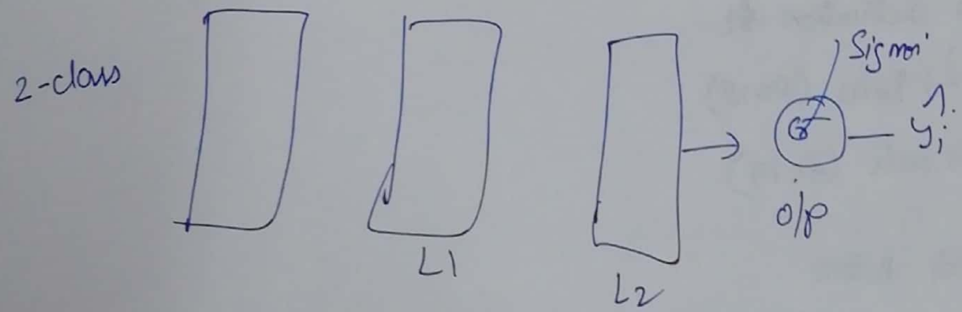
2 class

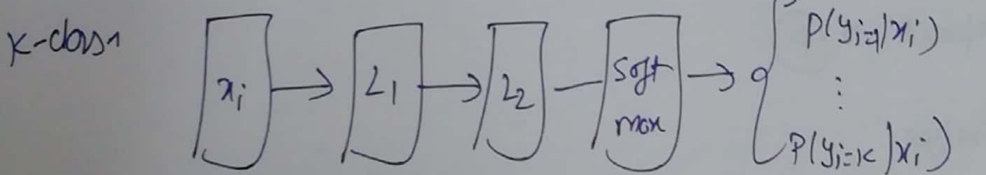
$$-\frac{1}{N} \sum_{i=1}^{N} y_i \log p_i + (1-y_i) \log (1-p_i)]$$

$x_i, y_i = 3$

1	2	3	4	...	K
0	0	1	0	..	0
y_{i1}	y_{i2}	y_{i3}	y_{i4}	...	y_{iK}

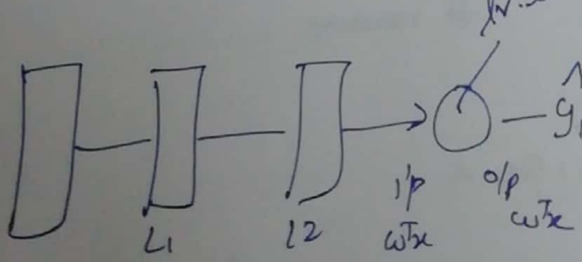






Vector of size K

<u>Regression</u>



48.17 How to train a deep MLP?

1) Preprocess data
   ↳ Data normalization.

2) weight init
   ↳ Xavier / Glorot → Sigmoid / tanh
   ↳ He → ReLu
   ↳ Gaussian (small σ)

3) Choose activation fn
   ↳ ReLu (2018)
   ↳ Selu (2017)

4) Batch Norm
   ↳ Esp for deep MLP (later layers)
   dropout → P

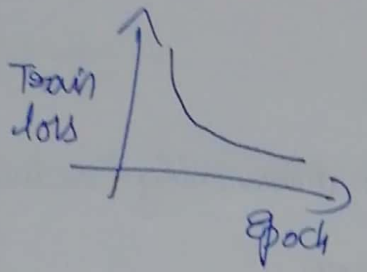
5) Optimizer
   ↳ Adam (2018) (Adaptive fast)

6) hyper parameters
   ↳ Architecture ⇒ # layers
   # neurons
   ↳ drop out
   ↳ Adam : β_1, β_2, α.

7) Loss function : 2-class classificatn → log loss
   k-class classifical → multiclass LL
   regression → square loss

8) Monitor your Gradients
   ↳ Gradient clipping (if needed)

9) Plots



10 avoid overfitting
{train & test losses}

48.18 Auto Encoders

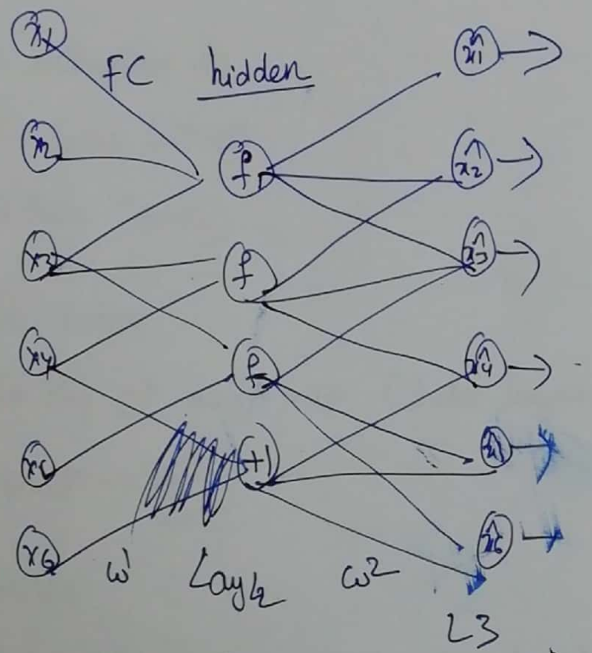
→ It is a NN, which performs dimensionality reduction [PCA, TSNE]
↳ variance
→ neighborhood

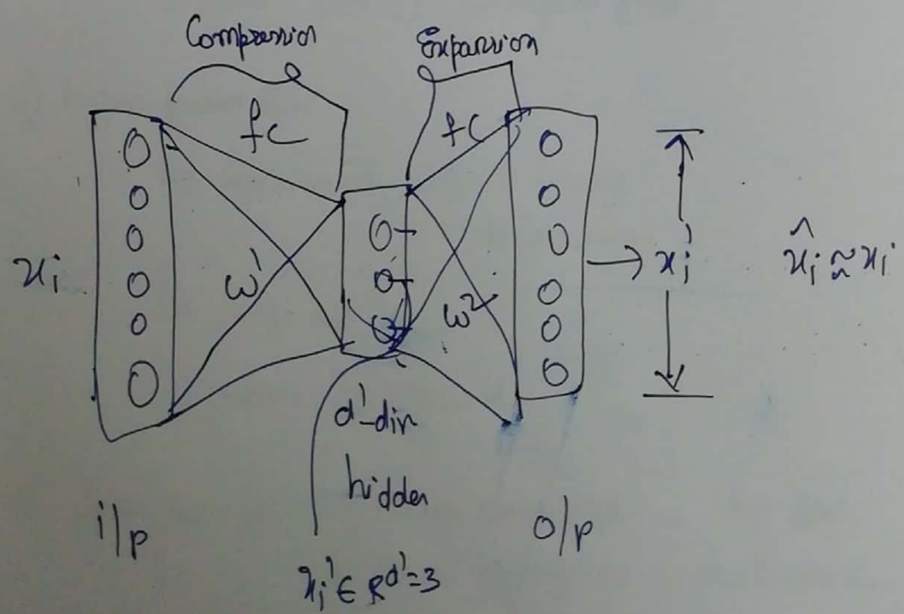
→ Sometimes better than PCA, TSNE

→ $D = \{x_i\}_{i=1}^{n}$ $x_i \in R^d$

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

Ex





$\mathcal{L}(x_i, \hat{x}_i) = \|x_i - \hat{x}_i\|^2$

$\boxed{\mathcal{L} \approx 0}$ → distance loss

Denoising Auto Encoder :-

$\mathcal{D} = \{x_1, x_2, \cdots x_n\} \leftarrow$ actual data

we get $\rightarrow \tilde{\mathcal{D}} = \{\tilde{x}_1, \tilde{x}_2, \tilde{x}_3 \cdots \tilde{x}_n\} \rightarrow$ noisy & corrupted

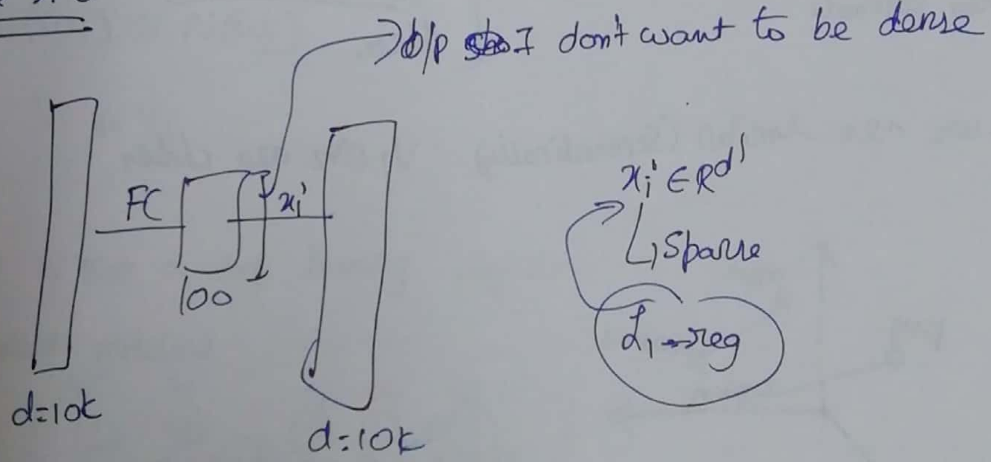
$\hookrightarrow$ AE

$\tilde{x}_i \in \mathbb{R}^d \rightarrow \boxed{AE} \rightarrow x_i' \in \mathbb{R}^{d'}$

$\hookrightarrow$ Robust to noise

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

Sparse AE



$x_i' \in \mathbb{R}^{d'}$

$\hookrightarrow$ Sparse

$L_1 \rightarrow$ reg

$\rightarrow$ If all linear activation are used & only a single sigmoid hidden layer, then the optimal solution to an autoencoder is strongly related to principal component analysis (PCA)

48.19 Word2Vec: CBOW

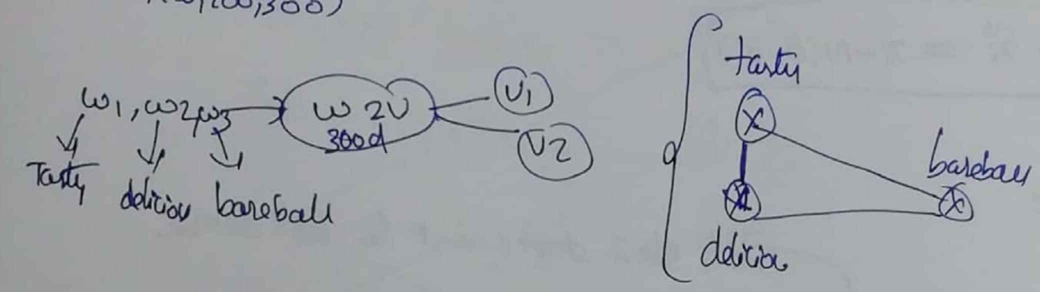
→ give it a word it returns a vector

→ Semantic meaning of word's into consideration.

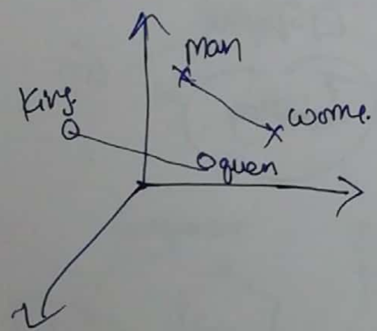
word → vector

word → [ ] → d-dime vect
→ not a sparce vect

d typically
(50, 100, 200, 300)



w_1 & w_2 are similar (semantically) V_1 & V_2 are closer



$(V_{man} - V_{woman}) \parallel (V_{King} - V_{queen})$

W2V → learning relationships automatically from raw-text

large text corpus → W2V → word:vec

300
200
100
50

larger dimension → more info rich the vectors

data corpus ↑ → d ↑

Google-News → W2V → 300-dim

review
text → W2V → word:vec
sour

core ÷ W2V

will look at the sequence of words.

$w_1, w_2, w_3, w_4, w_5, w_2$

$\underbrace{w_1, w_2, w_3, w_4, w_5, w_2}$

neighborhood of w_3

$W2V(w_3) =$

$N(w_i) \approx N(w_j)$

$v_i \approx v_j$

→ W2V is not a deep learning algorithm.

→ ~~Mikolav~~ mikolav

C5 C4 C1 C2 C3

1 2 3 4 5

Sentence ÷ The cat sat on the wall

Content word   focus word   Content word

→ Content words are very useful in understanding the focus word. and vice versa

2-algo
- CBoW (Continuous Bag of words)
- Skipgram