$$w^{[l]} = (n^{[l]}, n^{[l-1]})$$

$$b^{[l]} = [n^{[l]}, 1$$

Deep Neural Networks Representation

Forward propagation:-

Input $a^{[l-1]}$

output $a^{[l]}$, cache ($z^{[l]}$ intermediate computation)

→ use to store ~~result~~ of one layer in back propagation

Back propagation

Input $da^{[l]}$

output $da^{[l-1]}$, $dw^{[l]}$, $db^{[l]}$

$$\Longrightarrow dz^{[l]} = da^{[l]} * g^{[l]}(z^{[l]})$$

$$\Longrightarrow dw^{[l]} = dz^{[l]} \cdot a^{[l-1]^T}$$

$$\Longrightarrow db^{[l]} = dz^{[l]}$$

$$\Longrightarrow da^{[l-1]} = w^{[l]^T} \cdot dz^{[l]}$$

$$\Longrightarrow dz^{[l]} = w^{[l+1]^T} dz^{[l+1]} * g^{[l]}(z^{[l]})$$

$$dz^{[l]} = dA^{[l]} * g^{[l]\prime}(z^{[l]})$$

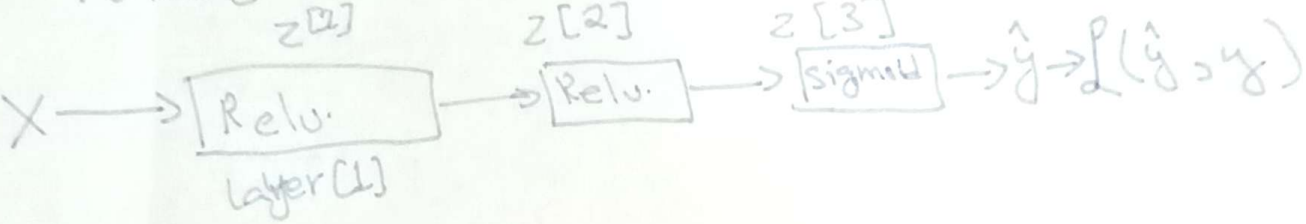
$$dw^{[l]} = \frac{1}{m} dz^{[l]} \cdot A^{[l-1]T}$$

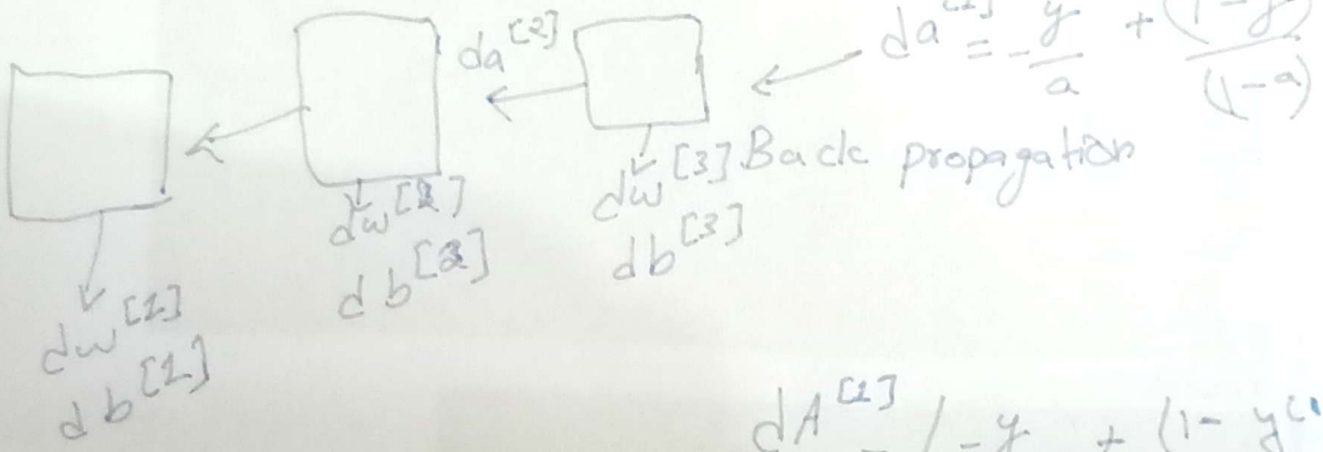
$$db^{[l]} = \frac{1}{m} np.sum(dz^{[l]}, axis=1, keepdims=True)$$

$$dA^{[l-1]} = W^{[l]T} \cdot dz^{[l]}$$

Summary:

Forward





$$dA^{[1]} = \left(\frac{-y}{a^{[1]}} + \frac{(1-y^{(1)})}{1-a^{[1]}} \right)$$

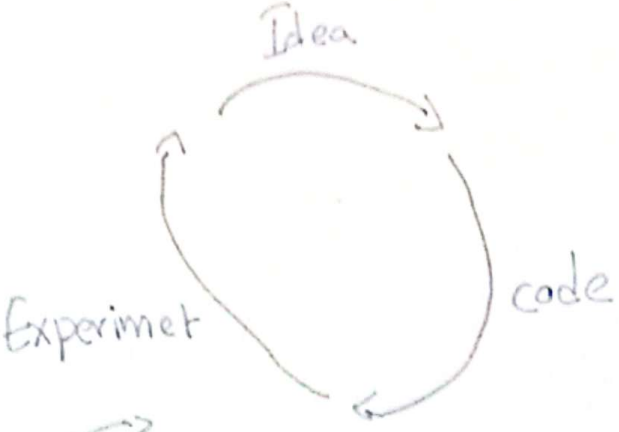
parameters.

$W^{[2]}, b^{[1]}, W^{[2]}, b^{[2]}, W^{[3]}, b^{[3]} \ldots\ldots$

Hyperparameters: Defines what u want to train nn. This includes.

- Learning rate α iterations.
- Iterations
- Hidden layer L
- Hidden units (n, n_2, n_3)
- Choice of activation functions (Relu, Tan, sigmoid)

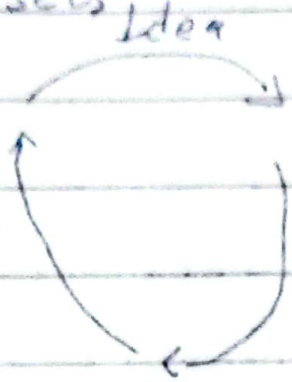
Many other : Momentum Term, mini batch, size, and other are parameters in



to get best model we have to change hyperparameters or try different parameter like LR α, Relu, tan, trying activation to get the best model

Deep learning by andrew Ng C2

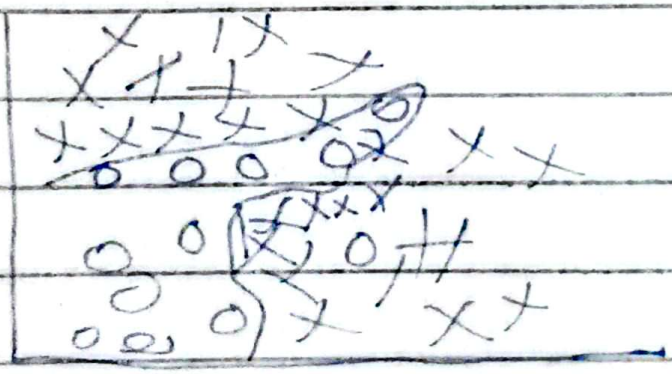
Train / Dev / Test sets



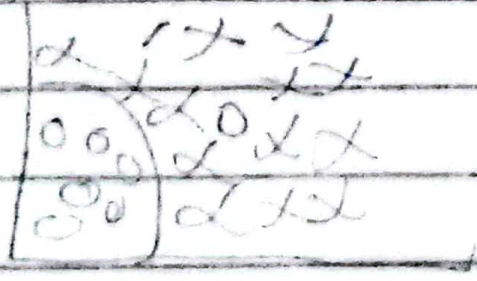
ML is iterative model.

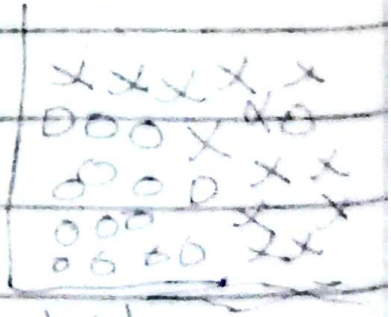
Bia and Variance trade ~~off~~

~~Sq~~ bial = $\frac{1}{m}\sum \hat{y} - y$, ~~Square~~ absolute-mean-er



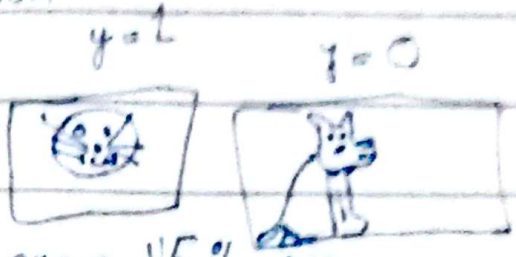
overfitting





Bias and Vat variance

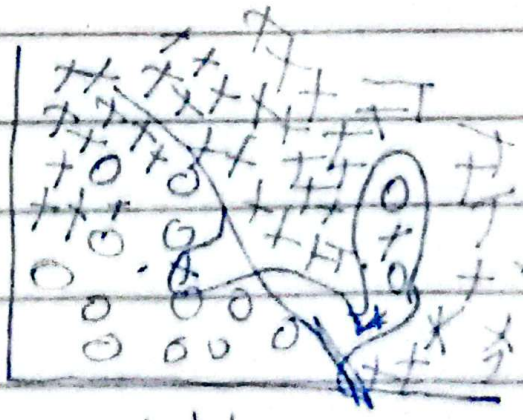
Cat classification



Train set error:	1% of error	15% error
Dev set error:	11% of error	16% error
	high variance error	underfitting error
		high bias problem

Train error:	15%	0.5%
Dev set:	30%	1%
	high bias & high variance	low bias problem low variance problem

0% optimal error:-



high bias, high variance

topic) Basic recipe for ml

High bias? → Bigger network (try)
(training data performance) try train longer
NN architecture search

if we have to check high variance or bias problem first we have to check High bias to reduce the error to get rid of High bias problem.

Now for High variance

High Variance → More data
(data set performance) Regularization
Sometimes (
NN architecture

By C. GPT

	Symptom	Cause
Bias: Error due to simple model (training set) that can't capture patterns well	High training and dev error	Underfitting
Variance: · Errors due to overly complex model that fits noise in training data	low training but high dev/valid error	Overfitting

Step 2:

Observation	Interpretation	Problem Type
High training error High test error	Can't fit training data well	High Bias (underfitting)
low training error → High test error	Model fits training but fails generaliza...	High variance (overfitting)
low training and low test error	Model is doing well	Good fit

If High bias (Underfitting)

Use complex model (add layers / neurons in NN
(deeper tree in decision tree)

+ Train longer / better optimization.

+ Add more relvant feature in traing.

- Decrease regularization (L1 and L2

High Variance

- Use simple model (reduce layers, limit tree depth
- Reduce number of x
- Use more training data
- Use cross-validation to tune parameters
- Data Cleaning.

Regularization.

When model learns very well from training data, including noise or irrelevant patterns it perform poorly on unseen data, this is called overfitting.

$$J(w,b) = \frac{1}{m}\sum_{i=1}^{m} L(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2m}\|w\|_2^2 + \lambda b/2m$$

Frobenius norm $\|W^{[l]}\|_F^2 = \sum_{j=1}^{n^{[l]}} \sum_{j=1}^{n^{[l-1]}} (w_{i,j}^{[l]})^2$

L_2 regularition is more often use

L_1 regularition

$$\frac{\lambda}{2m}\sum(w_j)\sum_{j=1}^{n_x}|w_j| = \frac{\lambda}{2m}\|w\|_1$$

where λ is a parameter.

Neural Network

$$J(w^{[1]}, b^{[1]}, \dots, w^{[L]}, b^{[L]}) = \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2m}\sum_{l=1}^{L} \| w^{[l]} \|^2$$

$$\| w^{[l]} \|_F^2 = \sum_{i=1}^{n^{(l-1)}} \sum_{j=1}^{n^{(l)}} (w_{ij}^{[l]})^2$$

L2 regularization

$$dw^{[l]} = (\text{from back prop}) + \frac{\lambda}{m} w^{[l]}$$

$$w^{[l]} := w^{[l]} - \alpha\, dw^{[l]}$$

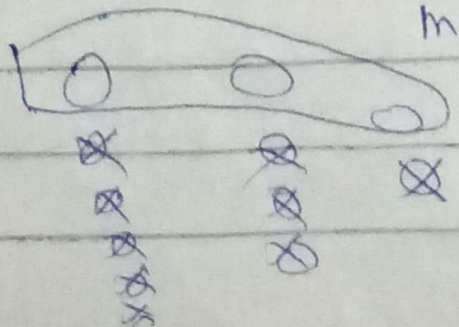
$$w^{[l]} := w^{[l]} - \alpha\, dw^{[l]}$$

Why regularization reduce overfitting

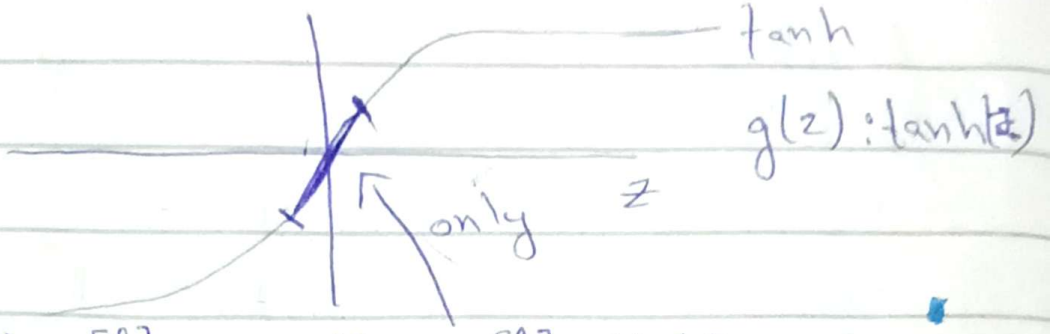
$$J(w^{[l]}, b^{[l]}) = \frac{1}{m}\sum_{i=1}^{n} \mathcal{L}(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2m}\sum_{l=1}^{L} \| w^{[l]} \|_F^2$$

high variance or bias

In ~~regularization~~ $w^{[l]} = 0$ cause 0 in neural networks



make nn simple cause high variance

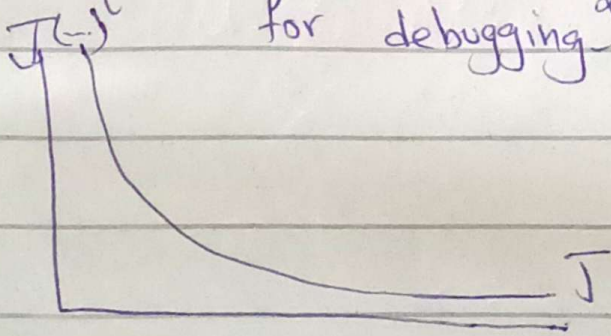


$\lambda \uparrow , \downarrow w^{[l]} \Rightarrow z^{[l]} = w^{[l]} a^{[l-1]} + b^{[l]}$

when $w^{[l]}$ decreases, $z^{[l]}$ also increases causing only linear performance which cause not a good performance

i.e Every layer $\approx$ linear.

$$J(\ ) = \sum_i \mathcal{L}(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2m} \sum_l ||w^{[l]}||_F^2$$



for debugging.

Dropout Regularization Technique

It causes in the drop of neural network cause/result in more simple and generalize NN causing in good performance as it learns patterns not memorize everything.

How regularization happens:-

In cause the complex neural network into simple or general which help network to learn general or simple pattern by

- droping some nn connections

Implement dropout.

Layer 3

d3 = np.rand.random(a3.shape[0], a3.shape[1])

> keep.prob = 0.8

0.8 probablity of droping a neuron

0.2 percent to eliminate NN

a3 = np.multiply(a3, d3)

if anone has value - zero remove it from network.

probability a3/ = keep-prob (if NN=50)

0.8 this cause 10 drop of NN from network

How regularization happens:-

In cause the complex neural network into simple or general which help network to learn general or simple pattern by

- droping some nn connections

Implement dropout.

Layer 3

d3 = np.rand.random(a3.shape[0], a3.shape[1]

> keep_prob = 0.8

0.8 probablity of droping a neuron

0.2 percent to elim

NN

a3 = np.multiply(a3, d3)

if anone has value - zero remove it from network.

probablity a3 /= keep_prob (if NN=5

0.8 this cause 10 drop of NN from network

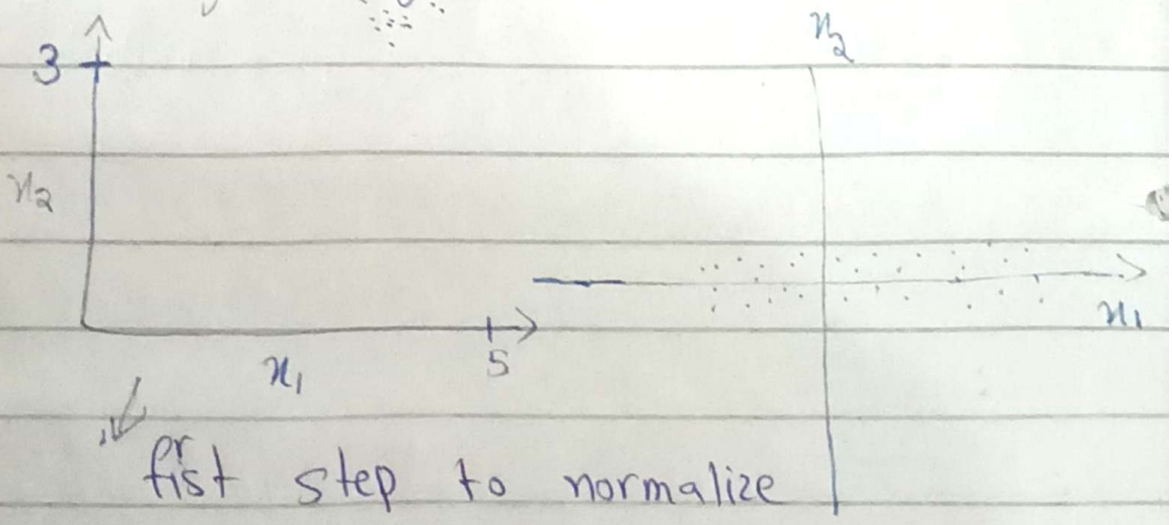
$w^{[3]} = 3 \times 7$

~~Regularizer~~ Understanding Dropout-

Data augmentation

Suppose we have one Photo of cat we can use it by tilting, decrease increase brightness, by zooming, and we can train these photo to train our models

Normalizing training sets



first step to normalize

Subtract mean

$$\mu = \frac{1}{m} \sum_{i=1}^{m} x^{(i)}$$

$$X_1 := X - \mu$$

2nd step

Normal variance

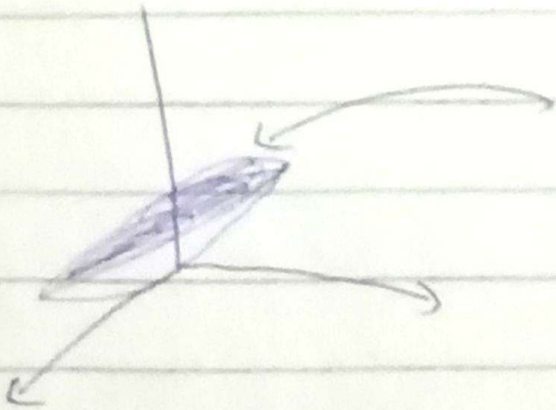
$$\sigma^2 = \frac{1}{m} \sum_{i=1}^{m} x^{(i)} \text{ Z}$$

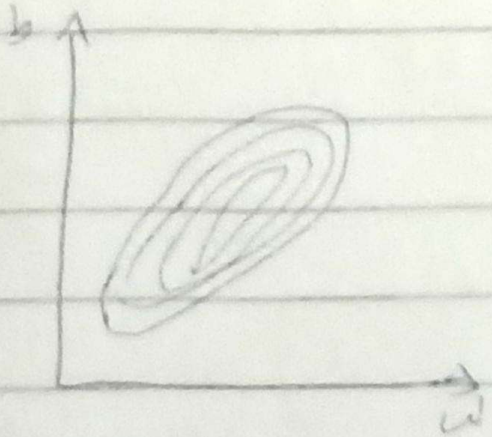
$$\eta / = \sigma$$

Cost function (loss function) - It tells how wrong system is

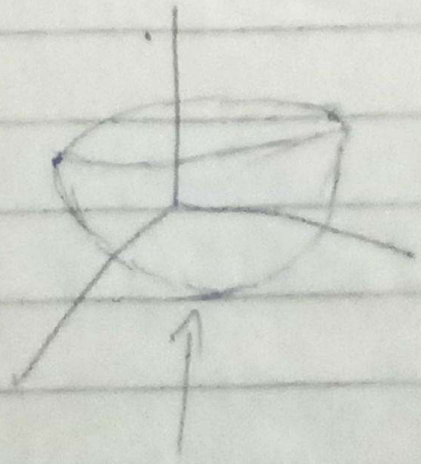
Why we use normalize inputs

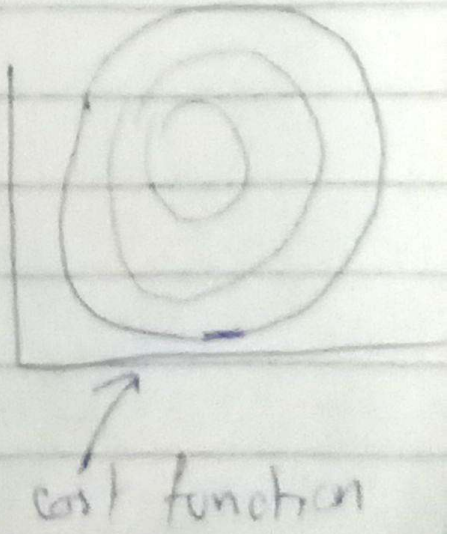
unnormalized:





normalize data





Vanishing - Exploding Gradients -

When neural network increase the activation function decrease like

$$W = \begin{bmatrix} 1.5 & 0 \\ 0 & 1.5 \end{bmatrix}$$

$$\hat{y} = W^{[L]} W^{[L-1]} W^{[L-2]} \cdots$$

Weight intialize for Deep learning -

$$z = w_1 x_1 + x_2 x_2 \cdots w_n x_n + b$$

larger n is we want smaller w_i will be - One way is to

$$Var(w_i) = \frac{1}{n}$$

$$w^{[l]} = np.random.rand(shape) * np.sqrt\left(\frac{2}{n^{[l-1]}}\right)$$

Relu $g^{[l]}(z) = Relu(z)$

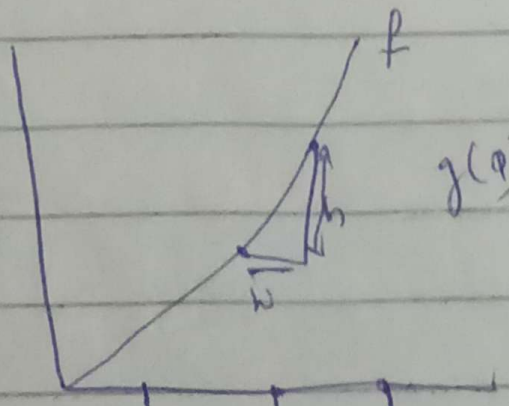
other varaince

1) tan h - $\sqrt{\frac{1}{n^{[l-1]}}}$

2) Xavier intiation

$$\sqrt{\frac{2}{n^{[l-1]} + n^{[l]}}}$$

They are all used to decrease wi cost fonction

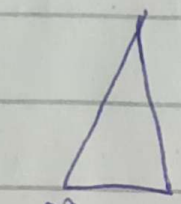


$$g(\theta) = \frac{f(\theta+\varepsilon)^2 - f(\theta-\varepsilon)^2}{2\varepsilon}$$

$\varepsilon = 0.01$

$$\frac{f(\theta+\varepsilon) - f(\theta-\varepsilon)}{2\varepsilon}$$

it is more accurate formula for derivative.



We can use this value $\varepsilon = 0.01$ for own but it will be small

always built in library/creator. ③

$$\frac{(1.01)^3 - (0.99)^3}{2(0.01)}$$

$$= 3.001$$

$$g(\theta) = 3\theta^2 = 3$$

Gradient checking (apply)

for each i:

$$d\theta[i] = \frac{J(\theta_1, \theta_2, \theta_3 \dots \theta_i + \varepsilon \dots) - J(\theta_1, \theta_2 \dots \theta_i - \varepsilon)}{2\varepsilon}$$

this is used to check errors in back propagation

$$d\theta[i] = \frac{\partial J}{\partial \theta}$$

Check the loop if

answer 10^{-7} u did a good job.

if 10^{-5} u' have to look into model but ok

10^{-3} something is in error

- Don't use in traing only to debug.
- If algorithm fails grad, chect at component
- Remember regularization.

---

* We cannot use drop out with Grad check we us in this sequence.

drop out ← to make NN simple → Grad check (how accurate model)

↓

backpropagation

Note: Grad check is only use we we have to check back propagation accuracy at once.

Week 2 (Use case: back propagation + optimization algorithem)

Optimization algorithems

if we have 500000 exampel to train

Batch

=> Usee all training gradient data

=> Slower, takes high memory.

=> ☆ More accurat gradient.

Mini Batch

for t=1...5000

forward prop on $X^{[t]}$

$$Z^{[1]} = W^{[1]} X^{[1]} + b^{[1]}$$

$$A = g^{[1]}[Z^{[1]}]$$

compute cost $J = \frac{1}{1000} \sum_{i=1}^{n} l(\hat{y}^{(i)}, y^{(i)})$

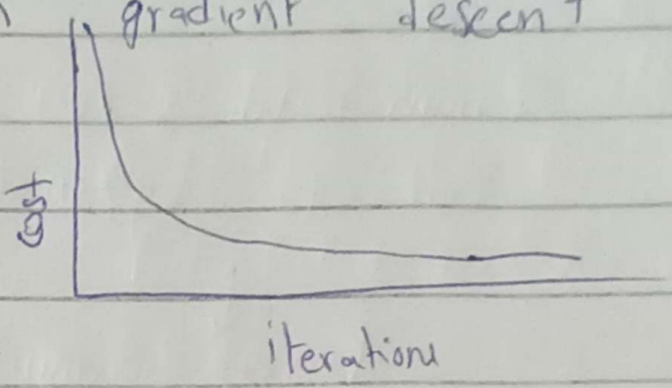
$$\frac{\lambda}{2.1000} \sum_{t} \| w^{[t]} \|_F^2$$

convert these large data set into small patches result in eficiency, faster working or training

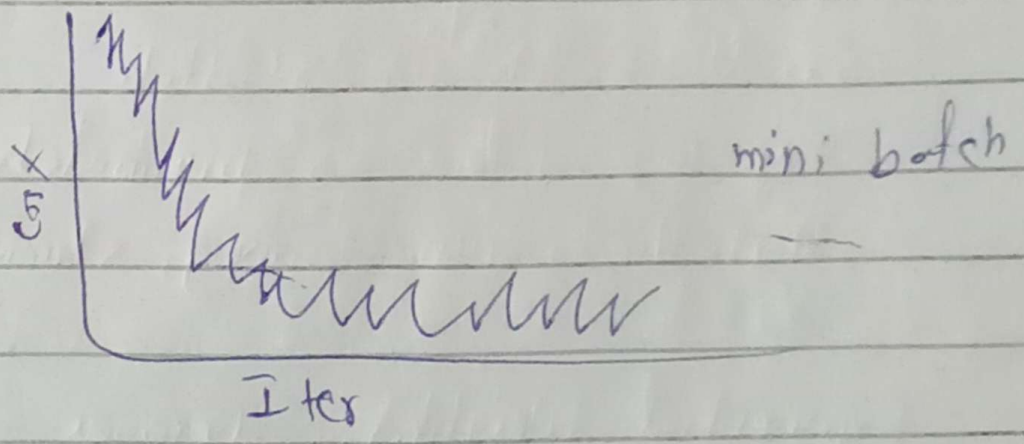
Mini_batch and batch model

Create small batches(32, 64, 128 samples)

Batch gradient descent

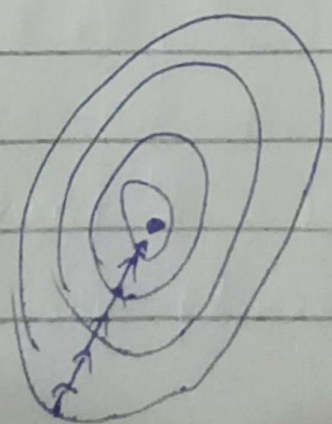


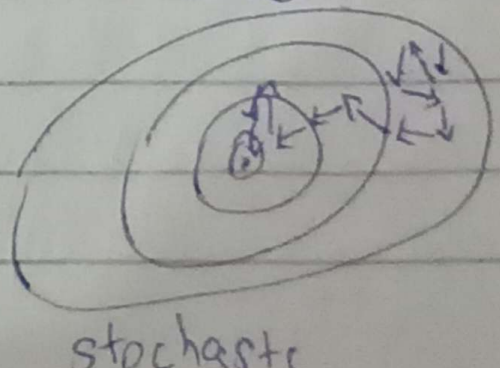
Iteration ↑ cost Increase



If minibatch size = 1 =
stochasting gradient deven

If minibatch size = m = Batch gradient deseent.





Stochastic gradient descent (Use for small, faster streaming, data)

↓

loose speed up from vectorization

(mini-batch size
not to by (small)

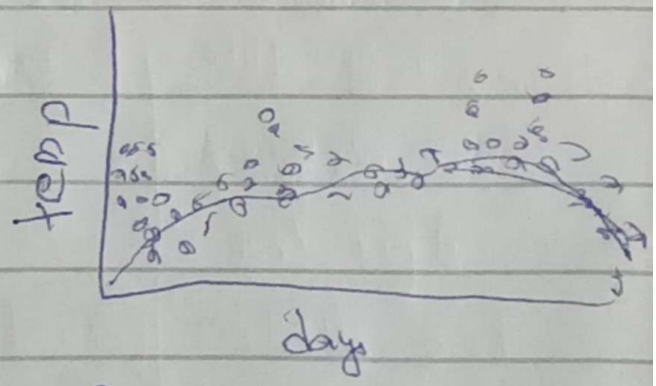
↓

forfeit learning
. Vectoring (1000)

small ~~batch~~ data size ⇒ batch
very large data ⇒ minibatch
Stochastic ⇒ online learning, streaming, IOTs

. Exponential Weighted averages

$\theta_1 = 40°F$ $4°C$
$\theta_2 = 49°F$ $9°C$
$\theta_{180} = 60°F$



They create lines of moving example of daily data (any interval) like stockmarket

Key components EWA

$v_t = B v_{t-1} + (1-B)\theta_t \quad ; \theta_t = \theta - \alpha * v - t$

$v_{100} = 0.9 v_{99} + 0.10_{100}$

$v_{100} = 0.1\theta_{100} + 0.1\theta_{99} + 0.9$

$B = 0.9 \; , \; v_0 = 0$

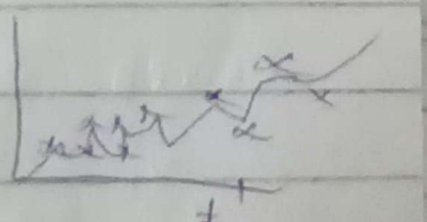
raw gradient $\nabla\theta = [4, 3, 2, 1]$

$= 0.9 \times 0 + (1-0.9)\, 4$

$v_{t_1} = (0.1)\, 4 = 0.4$

$v_{t_2} = 0.9 \times 0.4 + 0.1 \times 3$

$= 0.66$



Bias correction in this case:-

$t = 2 \; , \; 1 - B^t = 1 - (0.98)^2 = 0.0396$

$\frac{v_t}{0.0396} =$

Bias correction

$$v_t = \beta v_{t-1} + (1-\beta) g_t$$

$x = [10, 10, 10, 10]$

$v_0 = 0$

$\beta = 0.9$

with out bias correction we get

1) $0.9 \cdot 0 + 0.1 \cdot 10 = 1$

2) $0.9 \cdot 1 + 1.0 = 1.9$

$= 2.9 \Rightarrow 4.1$

this cause increase only

Now with Bias correction

$$v_t = \frac{v_t}{1-\beta^t}$$

v_t	$1-\beta^t$	$v_t = v_t / (1-\beta^t)$
$v_t = 1$	$1-0.9^1 = 0.1$	$= 1/0.1 = 10.0$

Concep	Without Bias	With Bias correction
v_t	1.0	10.0
Update scaling	Too large	Just righ
Real effect	Unstable learning	stable and accurate

it correct the initial value of V_t as

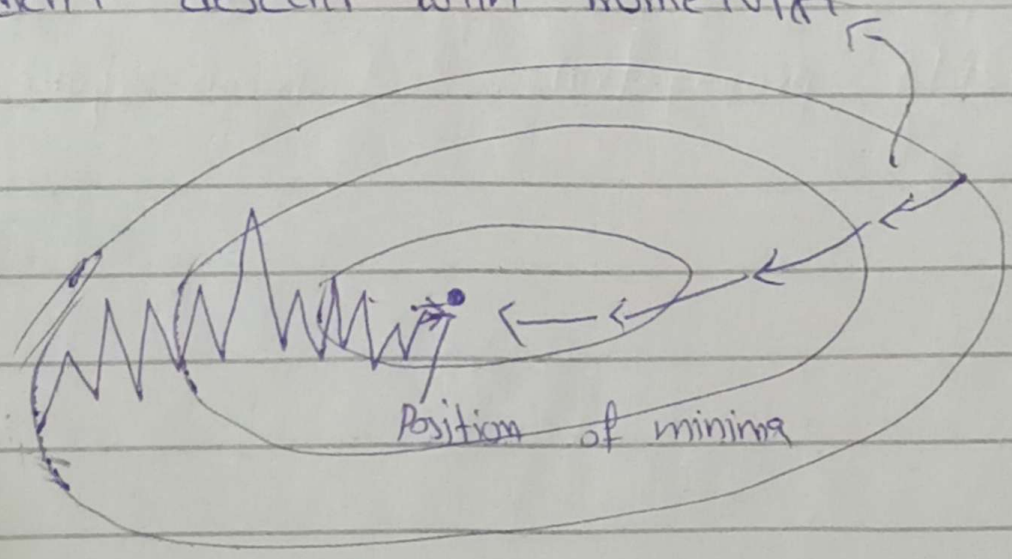
V_t initial may not be zero.

Reasons-

we cannot beleive every data start with zero-

* Bias don't change initial value but it corrects ~~the~~ or impacts the early steps.

Gradient descent with momentum



$$V dw = \beta V dw + (1-\beta) dw$$

$$V_b = \beta V_0 + (1-\beta) Q_t$$

this cause in increasing speed

GD Momentum

- Used for speeding convergence
- Reduce oscillation

GD EWA

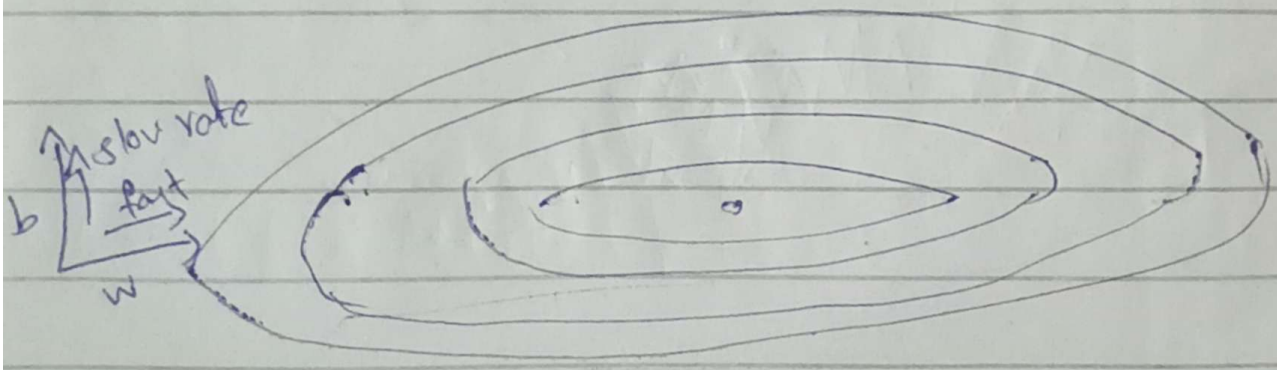
Smoothing noisy data

Detecting trends

e.g.

SGD, Adams.

RMS propagation: Root mean square propagation



Compute dw, db on eaunt mini-batch

$S_{dw} = \beta S_{dw} + (1-\beta) dw^2$ — small

$S_{db} = \beta S_{db} + (1-\beta) db^2$ → large

$$w := w - \alpha \frac{dw}{\sqrt{S_{dw}}} \qquad b := b - \alpha \frac{db}{\sqrt{S_{db}}}$$

In next step, we will use RMS with Momentum

Adam Optimization

$$S_{db} = \beta_2 S_{db} + (1-\beta_2) db^2$$

This algorithm take GD Momentum and put it with RMS

$$V_{dw} = \beta_1 V_{dw} + (1-\beta) dw, \quad V_{db} = \beta_1 V_{db} + (1-\beta) db$$

$$S_{dw} = \beta_2 S_{dw} + (1-\beta_2) dw^2, \quad S_{db} = \beta_2 S_{db} + (1-\beta_2) db^2$$

Bias correcting –

$$V_{dw} = V_{dw} / (1-\beta^t), \quad V_{db} = V_{dw} / (1-\beta^t)$$

$$S = S_{dw} / (1-\beta_2^t), \quad S_{db} = S_{db} (1-\beta_2^t)$$

Hypermater choice

$\beta_1 = 0.9$

$\beta_2 = 0.999$ (dw) - default adam

$\varepsilon = 10^{-8}$

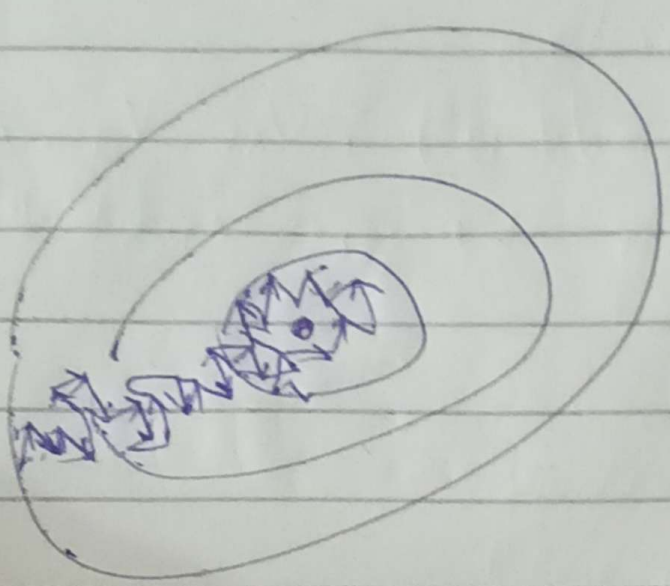
Adam: Adaptive moment distination

Clarification about learning rate decay.

$$\alpha = \frac{1}{1 + \text{decayRate} \times \text{epoch num}} \alpha_0$$

epoch = one cycle/or 1 pass training cycle.

Learning rate decay:-

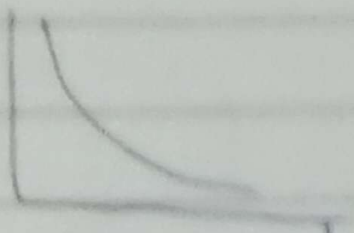


Suppose we are using mini-batch, due to small value(α) it will not reach minima, instead it will rotate around minima.

$$\alpha = \frac{1}{1 + \text{decayRate} \times \text{epoch}}$$

Epoch	α
1	0.1
:	

$\alpha = 0.02$
decay = 1



learning rate decrease

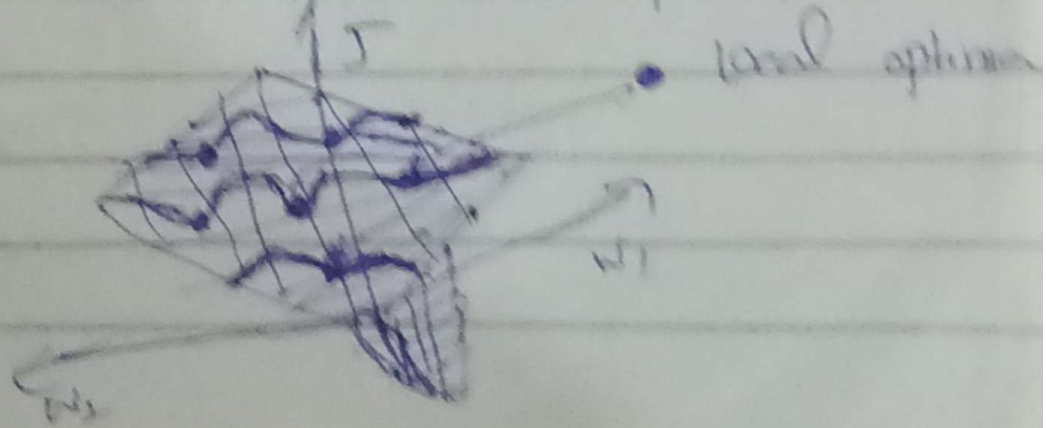
other learning rate decay.

$\alpha = 0.95^{\text{epoch\_num}}$

$\alpha = \frac{1}{\sqrt{\text{epoch\_num}}} \cdot \alpha_0$ or $\frac{k}{\sqrt{t}} \cdot \alpha_0$

Manual decay.

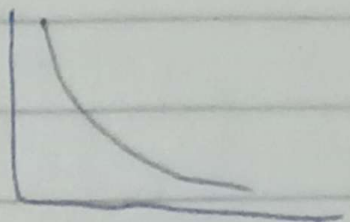
Problem with local optima



Epoch	α
1	0.1
.	

$\alpha = 0.02$

decay rate = 1



Learning rate decrease

---

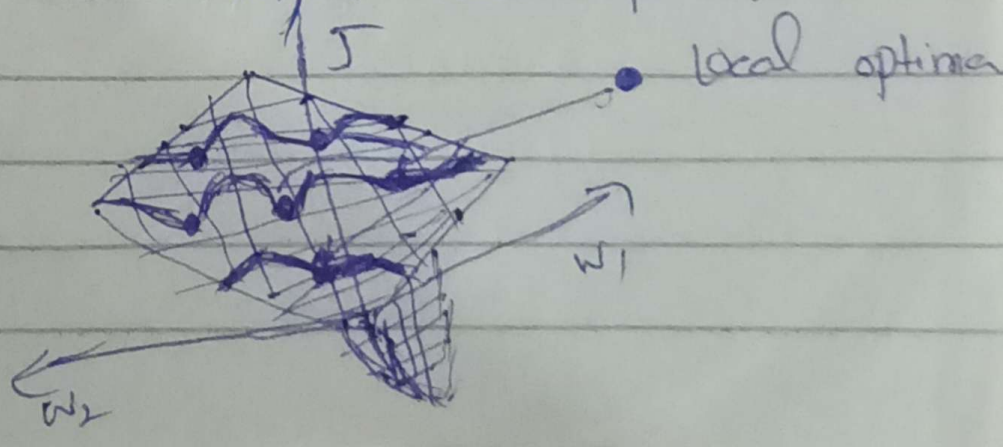
other learning rate decay.

$$\alpha = 0.95^{\text{epoch\_num}}$$

$$\alpha = \frac{1}{\sqrt{\text{epoch num}}} \cdot \alpha_0 \quad \text{or} \quad \frac{k}{\sqrt{t}} \cdot \alpha_0$$

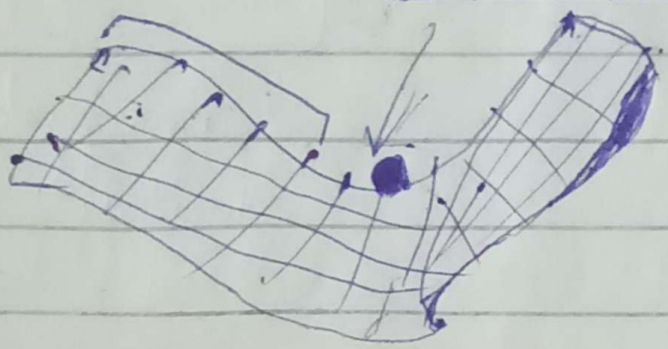
Manual decay.

Problem with local optima.



but according to graph it is not always true it is

plateaus — local optima



this point loss is low called saddle point

saddle = زین

⇒ plateaus ~~case~~ cause in slow learning.

WEEK THREE : 3

Hyperparameter tuning.

most important hyperparameter

$\beta = 0.9$ α, β_1, β_2, ε.

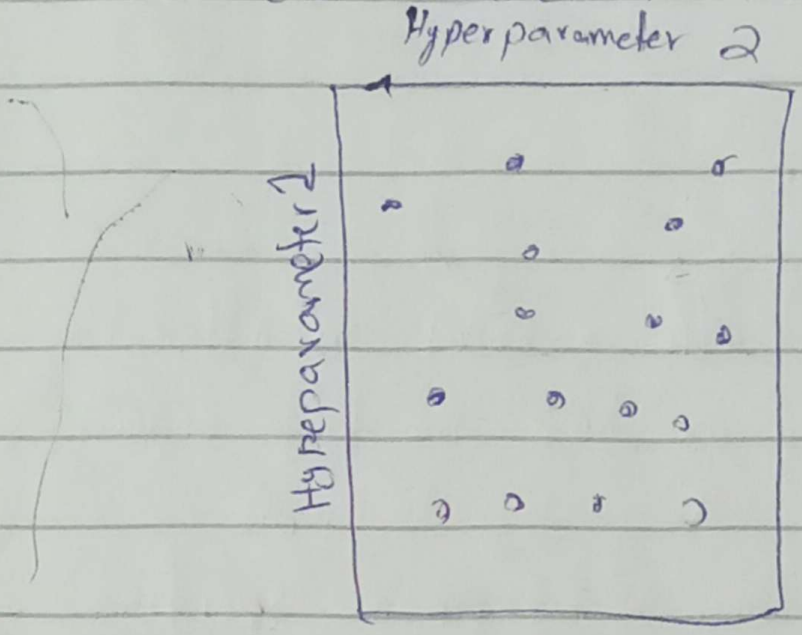
layers

hidden units and unit Import

learning rate decay — 3rd important

mini batch size

How can make it in work



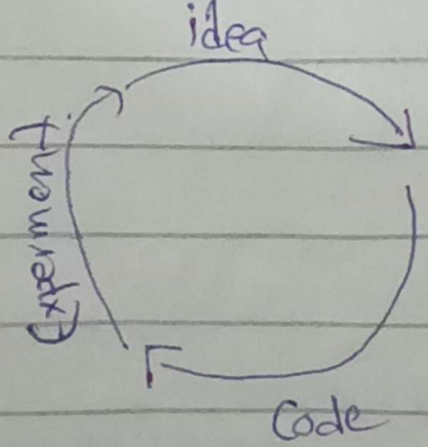
Search Bat hyper Daramter.

Grid Box

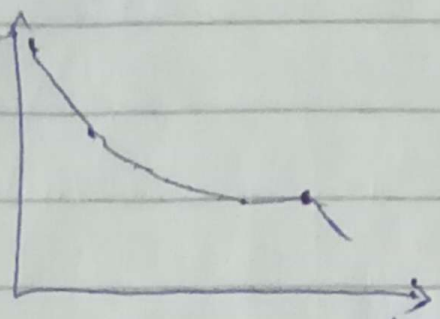
Using appropriate scale to pick hyperparameters

Layers (2-4)

~~typet~~ Tips to choose hyperparmeters.



Babysitting on model

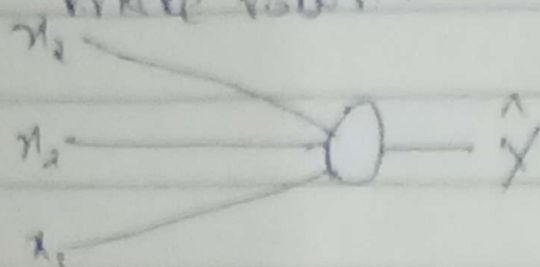


we can train ~~diffe~~ a model on different hyperparatare if we have no large capacity we can do it day by day, on each we tel ~~one~~ ~~hyperparam~~ specific hyperparameter known as panda approach one beby.

Caviar approach (many babies at a time)
We can try many hyperparameter if we have efficient resources.

Batch Normalization. ~~X = X/6~~

=> parameters work well and parameters make robust.



$$\mu = \frac{1}{m} \sum x^{(i)}$$

$$X' = X - \mu$$

$$\text{variance} = \sigma^2 = \frac{1}{m} \sum x^{(i)2}$$

$$X = X / \sigma^2$$

Can we normalize $a^{[2]}$ so that we train $w^{[3]}, b^{[2]}$ faster. Yes this is the way

Standardization: Model perform better

Normalization: If you don't normalize algorithm KNN, SVM, NN will give more importance with features with large range

Implementing Batch Norm

Given some intermediate values in NN

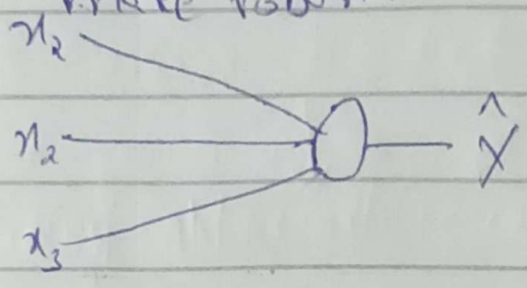
$$\mu = \frac{1}{m} \sum z^{(i)}$$

$$\sigma = \frac{1}{mi} \sum (z^{(i)} - \mu)^2$$

Batch Normalization

$X = X/6$

⇒ parameters work well and parameters make robust.



$$\mu = \frac{1}{m} \sum x^{(i)}$$

$$X' = X - \mu$$

$$\text{variance} = \sigma^2 = \frac{1}{m} \sum x^{(i)2}$$

$$X = X/\sigma^2$$

Can we normalize $a^{[2]}$ so that we train $w^{[3]}, b^{[2]}$ faster, Yes this is the way

Standardization: Model perform better

Normalization: If you don't normalize algorithem KNN, SVM, NN will give more importance with features with large range

Implementing Batch Norm

Given som intermediate velues in NN

$$\mu = \frac{1}{m} \sum z^{(i)}$$

$$\sigma = \frac{1}{mi} \sum (z^{(i)} - \mu)^2$$

μ = mean

$$z^{(i)}_{norm} = \frac{z - \mu}{\sqrt{\sigma^2 + \varepsilon}}$$ for stabily ε = small value

$$\tilde{z}^{(i)} = \gamma z^{(i)} + \beta$$

γ: gamma; β: learnable parameters of model

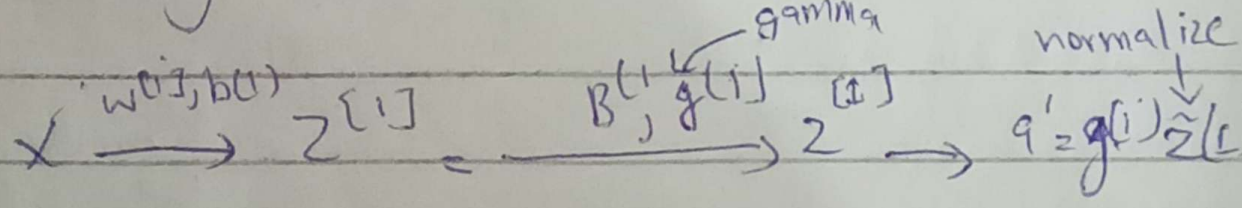
If $\gamma = \sqrt{\sigma^2 + \varepsilon}$

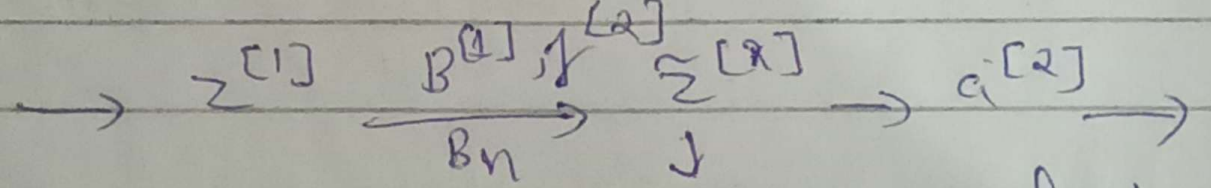
$\beta = \mu$

then $\tilde{z}^{(i)} = z^{(i)}$

Use $\tilde{z}^{[l](i)}$ instead of $z^{[l](i)}$

Fitting Batch Norm to a network





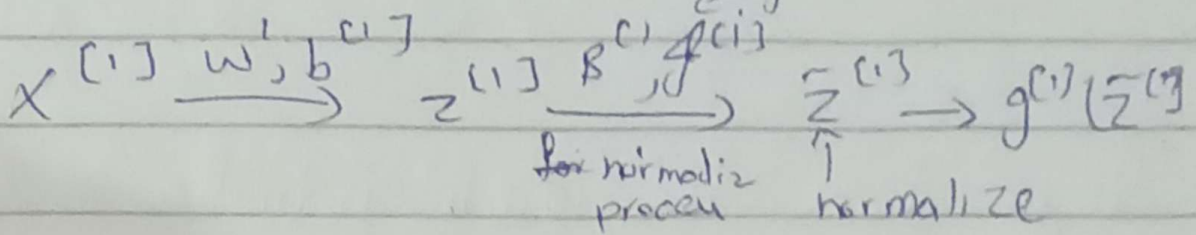
Parameters we get

$w^{[1]}, b^{[1]}, w^{[2]}, b^{[2]} \ldots$

$\Rightarrow \beta^{[1]}, \gamma^{[1]}, \beta^{[2]}, \gamma^{[2]} \ldots$

gama

Working with minibatch



Parameter

$\gamma^{[l]}, b^{[l]}, \beta^{[l]}, \gamma^{[l]}$

Implementation gradient descent

=> for t=1 ---- num Mini Batches

Compute forword propagation, Use BN to repa $z^{[l]}$ with $\tilde{z}^{[l]}$.

Use back prop & compute.

Update parameters.

Work w/ GD momentum, RMS, Adam.

Batch Norm:

Summary: It is a process in which parameters w, b are normalised into β, γ (gamma), such that it increase the working of model by preventing bw importance to features with large ranges.

Why does Batch normalize work:-

2nd reason:

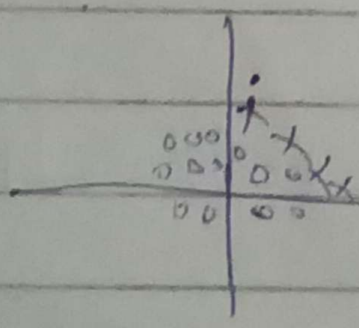
Let train cat detection model, if we compare colored or black cats. without Batch Norm.

our model or cost func don't work well.

What batch Norm do →

mean 0
variance = 1

remain same



- it limits the amount to which updating the parmaters in early layers which can effect distribution of layers in later layers and batch layer effect reduce the changes

* allow each layer layer independently.
* It allows to add some noise to valves $z^{[l]}$, within that minibatch.
similar of hidden layer's activation

* This has a slight regularization effect.

---

* Adrew ng don't use Batch norm as regularizer but it has extra effect on learning algorithem.

* don't use it as a regulizer

* Use it only to regularize hidden layers parameters only.

Use Batch Norm at test time

Batch Norm process data 1 mini batch at a time but at test time right we want process all at a time.

$$\mu = \frac{1}{m}\sum_i z^{(i)}$$

$$\sigma^2 = \frac{1}{m}\sum_i (z^{(i)} - \mu)^2$$

~~to~~ : μ, σ^2 = estimate this using weighted average (across mini-batch)

$x^{[1]}, x^{[2]}, x^{[3]}$

mini batches

$\mu^{[1]}, \mu^{[2]}, \mu^{[3]}$

$$z^{(i)}_{norm} = \frac{z^{(i)} - \mu}{\sqrt{\sigma^2 + \varepsilon}}$$

$$\tilde{z}^{(i)} = \gamma z^{(i)}_{norm} + \beta$$

in this same way we get same

$\sigma^{2[1]}, \sigma^{2[2]}, \sigma^{2[3]}$

after getting this we find this

Softmax Regression

$$a^{[L]} = \frac{e^{z[L]}}{\sum_{i=1}^{4} t_i}$$

Recognizing cats, dogs and baby chicks

$$z^{[L]} = w^{[L]} a^{[L-1]} + b^{[L]}$$

Activation function:-

$$t = e^{z[L]}$$

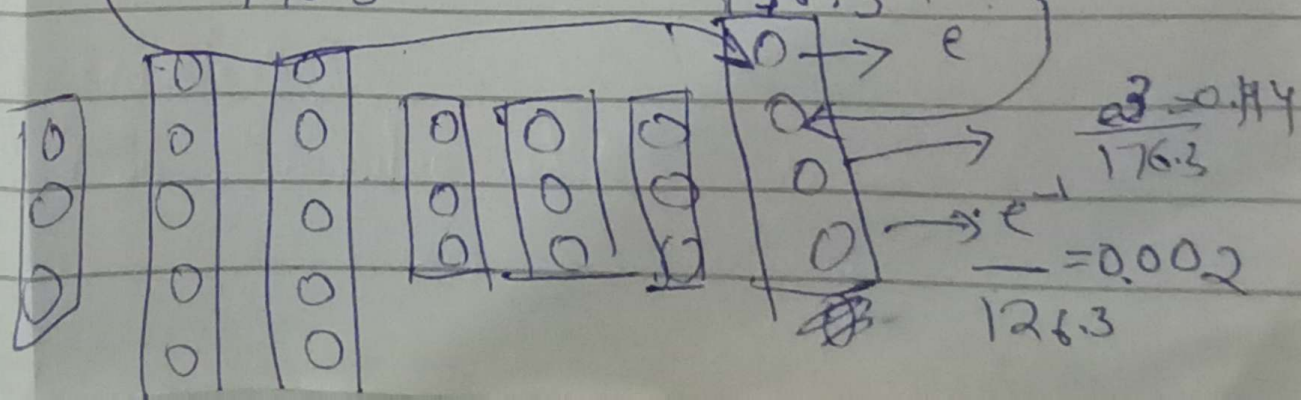
$$a^{[L]} = \frac{e^{z[L]}}{\sum_{j=1}^{n} t_i}, \quad a^{[n]} = \frac{t_i}{\sum t_i} [4,1]$$

example:

$$t = \begin{bmatrix} e^5 \\ e^2 \\ e^{-1} \\ e^3 \end{bmatrix} = \begin{bmatrix} 148.4 \\ 7.4 \\ 0.4 \\ 20.1 \end{bmatrix} = \sum_{j=1}^{4} t_i = 176.3$$

$$a^{[2]} = \frac{1}{176.3} =$$

for $\frac{e^5}{176.3} = 0.842$, $\frac{e^2}{176.3} = 0.042$



foreach NN

Softmax Regression ⟹ extension of logistic Regression.

Example :-

Classifying image of handwritten digits (0-9)
prediciting fruits {apple, banana, orange ...}

Training a Softmax classifier

$$z^{[L]} = \begin{bmatrix} 5 \\ 2 \\ -1 \\ 3 \end{bmatrix} = t = \begin{bmatrix} e^5 \\ e^2 \\ e^{-1} \\ e^3 \end{bmatrix}$$

$$g^{(L)}(z^{(L)}) = \begin{bmatrix} e^5/(e^5+e^2+e^{-1}+e^3) \\ e^2/(e^5+e^2+e^{-1}+e^3) \\ e^{-1}/(e^5+e^2+e^{-1}+e^3) \\ e^3/(e^5+e^2+e^{-1}+e^3) \end{bmatrix} = \begin{bmatrix} 0.842 \\ 0.042 \\ 0.002 \\ 0.114 \end{bmatrix}$$

↑ Probablity

0.84 it means 84% confidence of being familiar or detecting apple.

loss function

$\mathcal{L}(\hat{y}, y) = -\sum$
$j=$
$= -y$

Cos

$J(w, b$ --

Introductio

Framework

Caffe/Caf
CNTK
DL4J
KERAS
lasag
mxnet
Paddl Padd
Tensorflo
Torch,

foreach NN

Softmax Regression ⟹ extension of logistic Regression.

Example :-

Classifying images of handwritten digits (0-9)

predicting fruits {apple, banana, orange...}

Training a Softmax classifier

$$z^{[L]} = \begin{bmatrix} 5 \\ 2 \\ -1 \\ 3 \end{bmatrix} \quad t = \begin{bmatrix} e^5 \\ e^2 \\ e^{-1} \\ e^3 \end{bmatrix}$$

$$g^{[L]}(z^{[L]}) = \begin{bmatrix} e^5/(e^5+e^2+e^{-1}+e^3) \\ e^2/(e^5+e^2+e^{-1}+e^3) \\ e^{-1}/(e^5+e^2+e^{-1}+e^3) \\ e^3/(e^5+e^2+e^{-1}+e^3) \end{bmatrix} = \begin{bmatrix} 0.842 \\ 0.042 \\ 0.002 \\ 0.114 \end{bmatrix}$$

↑

Probablity.

0.84 it mean 84% confidence of being familiar or detecting apple.

loss function $y = \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix}$ $\hat{y} = \begin{bmatrix} 0.3 \\ 0.2 \\ 0.1 \\ 0.4 \end{bmatrix}$

$$L(\hat{y}, y) = -\sum_{j=1}^{4} y_j \log \hat{y}_j$$

$$= -y_2 \log \hat{y}_2 = -\log \hat{y}_2$$

Cost

$$J(w, b, \ldots) = \frac{1}{m} \sum_{i=1}^{m} L(\hat{y}^{(i)}, y^{(i)})$$

Introduction to Programing

Frameworks:

Caffe/Caffe2

CNTK

DL4J

KERAS

~~t~~ lasagne

mxnet

PaddlePaddle

Tensorflow Theano

Torch,

Tensor flow

If I want to learn tensor flow I get through the link of Adrew Ng course on Improving NN on coursera. At last there are link of further course on tensor flow.

Course 3:-

Structuring Machine learning strategy-

When training a model we have ideal, we will use it in a structure way, finding and creating strategies