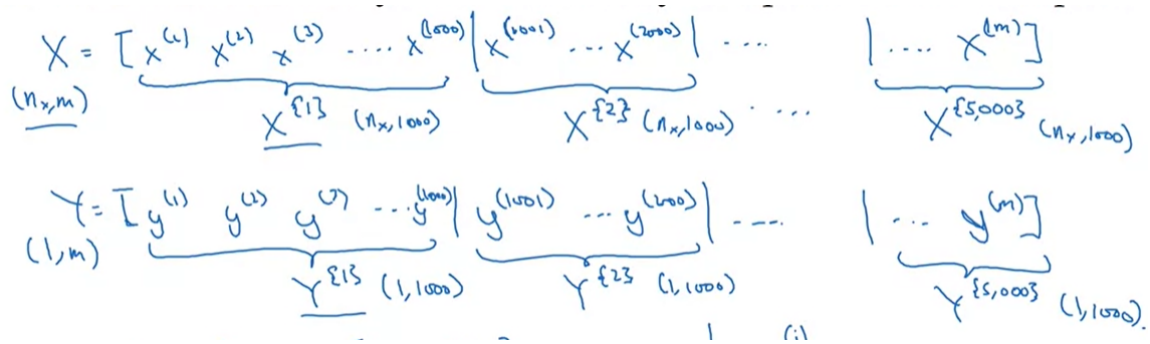
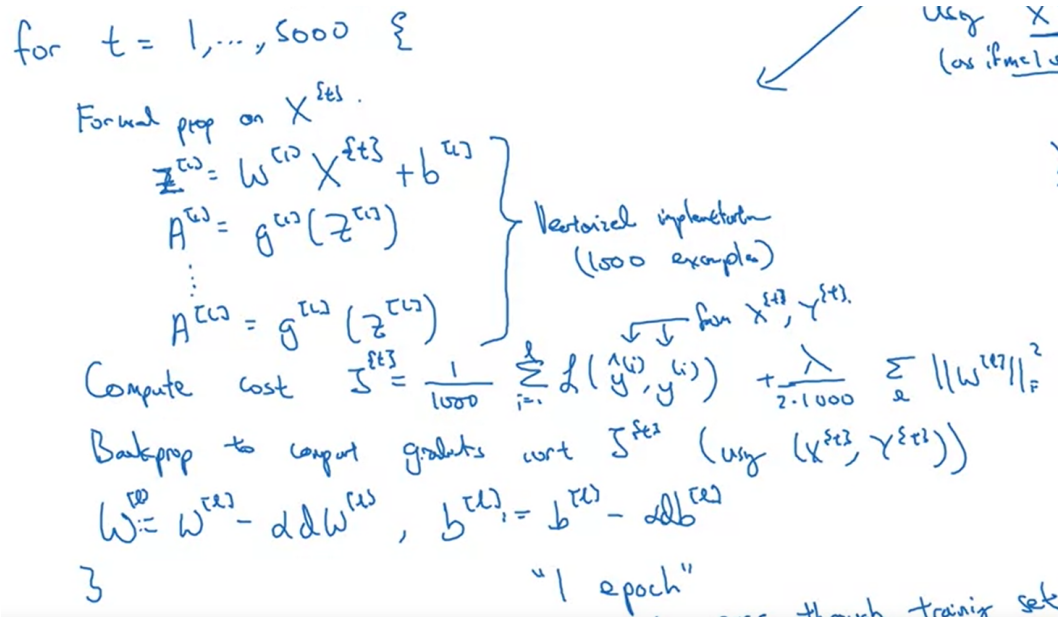
DI.Ai_Optimization algorithms

Optimization algorithms are what we use to speed up the process of learning though they may affect the accuracy of the model sometimes, they are not meant to increase the accuracy of the model usually

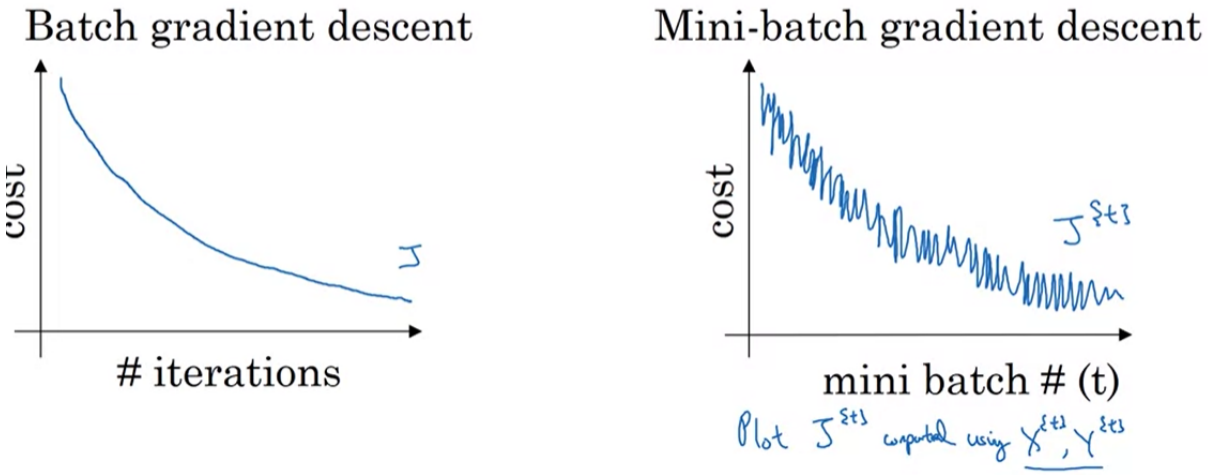
a)Mini Batch Gradient Descent



- In batch gradient descent all the instances are vectorized to form a vector x with dimension n x m where n is the no of features and m the no of instances
- each instance is represented as $x^{(i)}$ we will create a new vector $x^{\{i\}}$ compromising of $x^{(i)}$ to $x^{(1000)}$ which is called as our minibatch

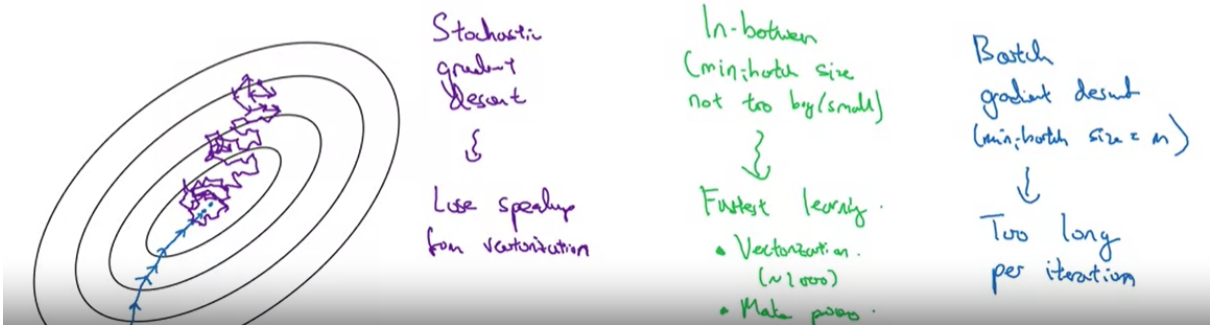


- first we must run backward propogation and forward propgation for each mini batch individually
- so we use for t in range (1000):
- we run our forward propogation steps just for the mini batch we have $z^{[1]}$ is t[2] x 5000
- We compute the loss function just for this mini batch, then use $J^{\{t\}}$ to update weights of the layer for the nueral network
- Note that $z^{[l]}$ and $a^{[l]}$ are seperate for each mini batch but the weights and biases of each layer is accesed by all mini batches.
- once all mini batches has been accesed atleast once a epoch is completed
- The weight and bias update we do for each minibatch is called a gradient step.Each epoch of mini batch gradient descent consists of 5000 gradient steps compared to1 step in batch gradient descent.
- Mini batch converges faster than a batch gradient descent.



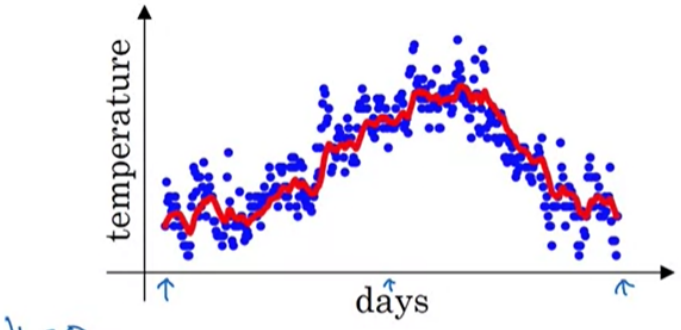
the cost vs mini batch is highly noisy because of obvious reasons

- Stochastic is not the fastest model because we loose the speedup we get by using vectorization.



b)Exponentially weighted averages

- this is just a statistical concept for now



$V_0 = 0$

$V_1 = 0.9\, V_0 + 0.1\, \theta_1$

$V_2 = 0.9\, V_1 + 0.1\, \theta_2$

$V_3 = 0.9\, V_2 + 0.1\, \theta_3$

$\vdots$

$V_t = 0.9\, V_{t-1} + 0.1\, \theta_t$

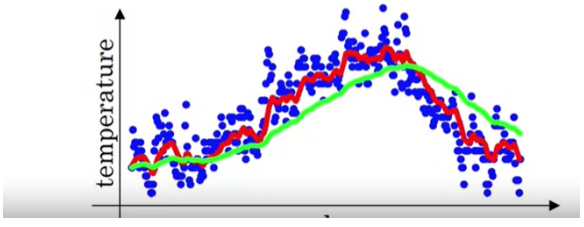
- we take a moving average of all existing instances but giving more weightage to recent values.
- even though technically all pre existing values are considered in reality only the last

$\frac{1}{1-\beta}$ will affect the average.hence its said to be a moving average

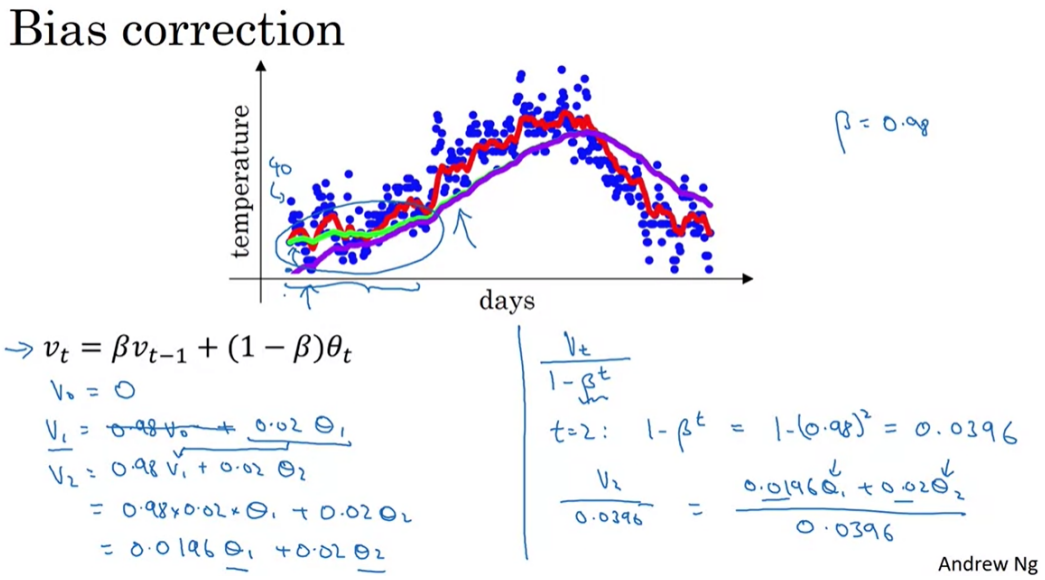
- Higher value of beta means that the average is taken over a larger window leading to more smoothening out

$\beta = 0.9$: $\approx$ 10 days' temperature

$\beta = 0.98$: $\approx$ 50 days

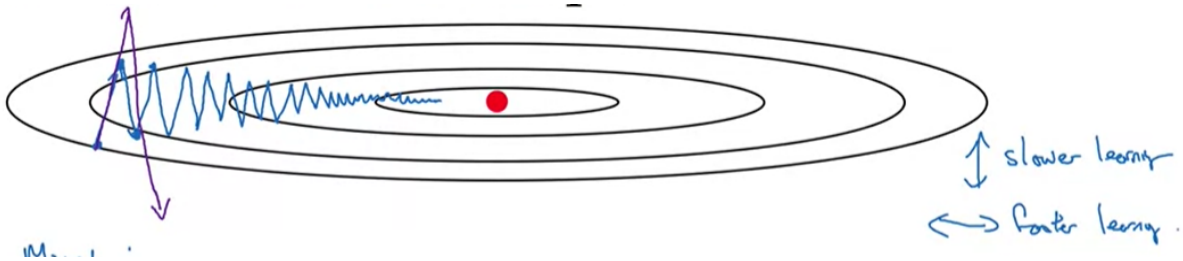


- It can be observed that for higher values such as b=0.98 the average takes place over an window of 50 instances the initial value of the average will be very low (first 50 insatnces) hence we use bias correction



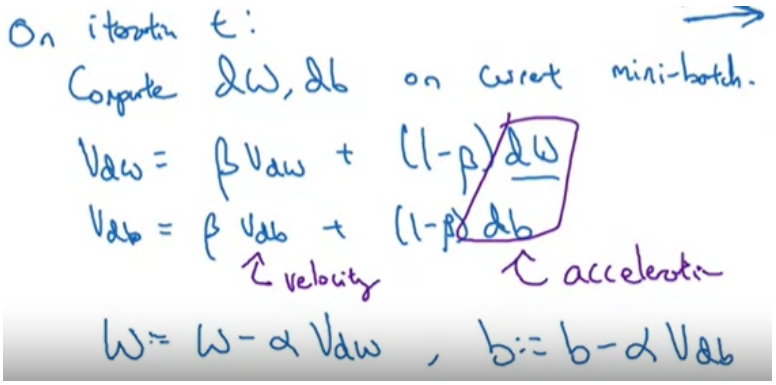
- we use $\frac{V^t}{1-\beta^T}$ instead of Vt it will converge to Vt for higher values of t

c)Momentum



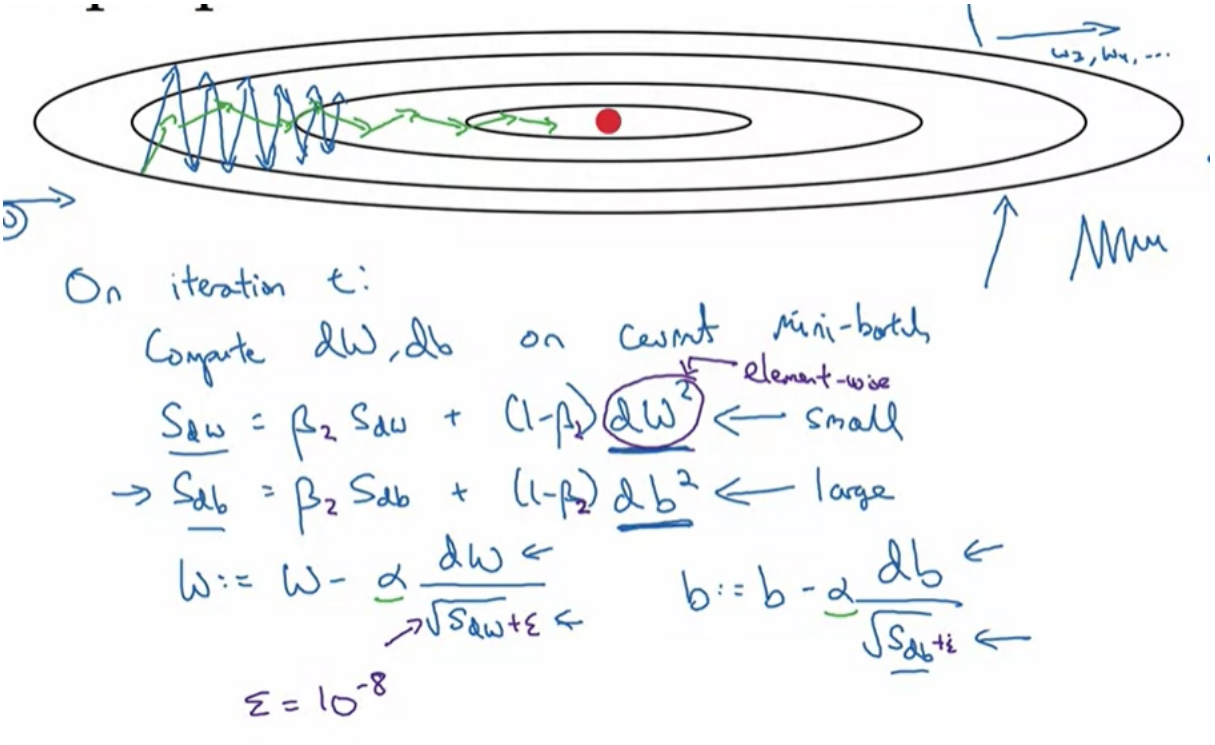
In this case we need to have faster learning in the horizontal direction and slower learning in the horizontal direction.We use Momentum which is a application of exponentially weighted average.

- If we can take the average of the individual steps we could neglect up and down oscillations.
- Hence we use exponentially weighted averages instead of the dw and db (i.e the gradients of loss function)

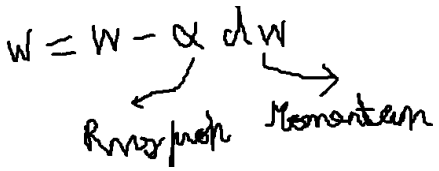


- Note we average out dw from different minibatches in this cases but we can also do it from different iterations.
- we can consider that the vdw represents the velocity of a body beta the friction that applies on it dw as the new acceleration it gets.
- β is a hyperparameter like α

d)RMSprop



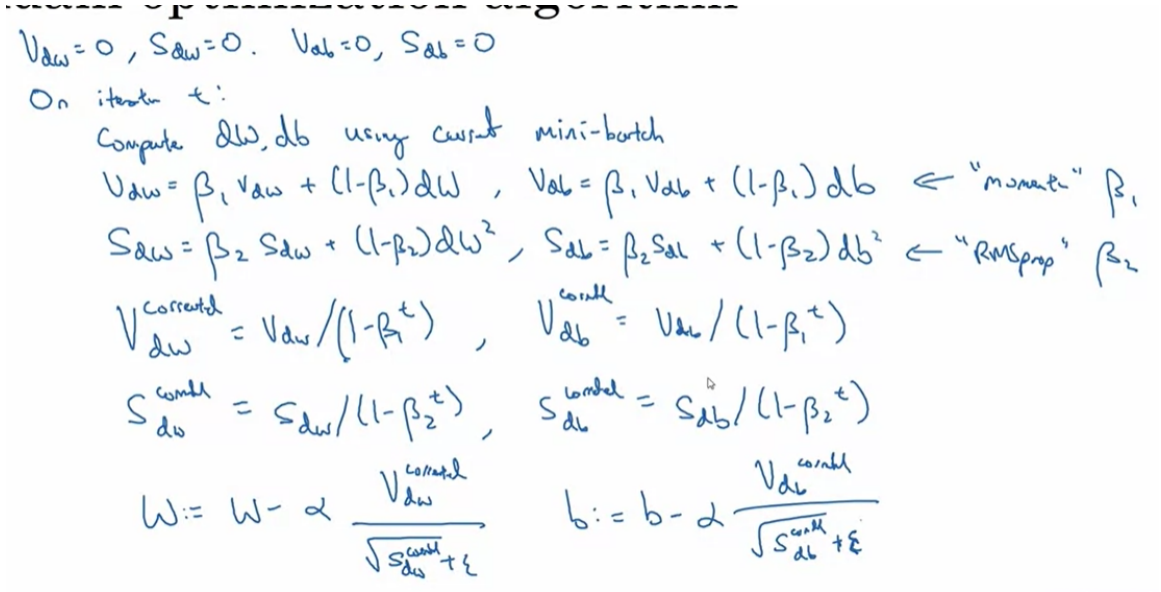
- RMS prop or root mean square prop is different from momentum in the sense that it solves the issue of oscillation by adapting the learning rate according to the parameter, instead of changing the differential.



- we use dw^2 and db^2 since db is very high and dw is very low (change in b is more than change in w) Sdw is smaller hence w is learned more compared to b.
- we add the term E to ensure that we dont divide by zero

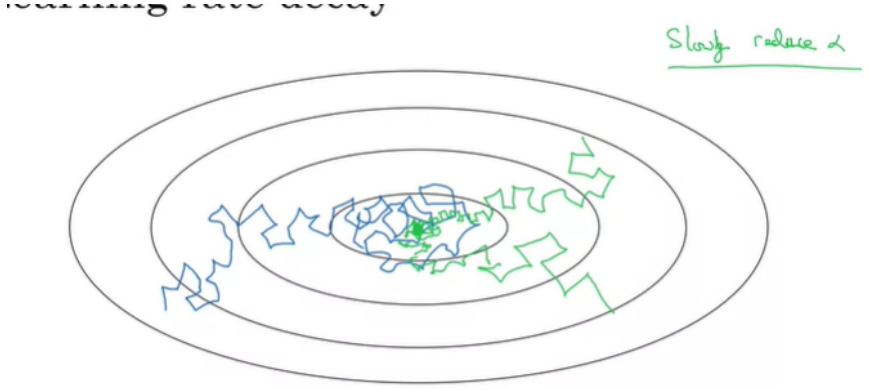
c+d)Adam Optimizer

- When we use both RMS prop and Momentum together we get adam optimizer which stands for adaptive moment optimizer.

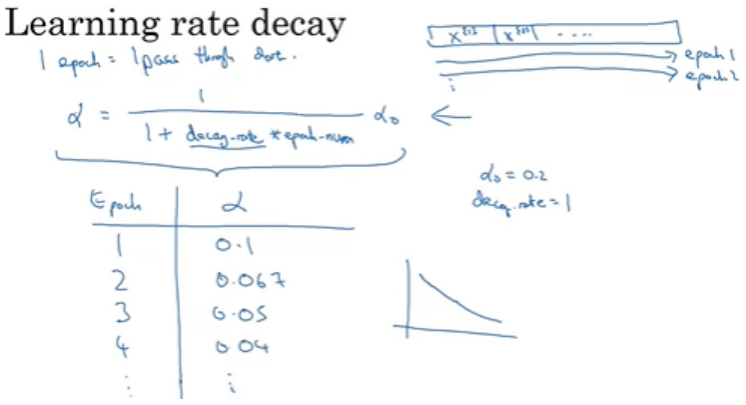


- Hyperparameters forb this model are α,β1,β2,E

e)Learning Rate Decay



- we can decrease the learning rate over epochs to reach convegence faster

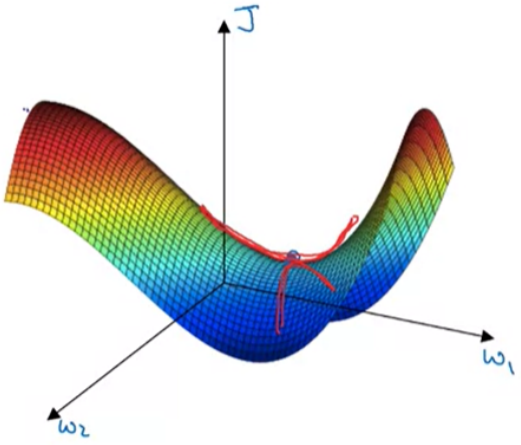


- decay is also an hyperparameter

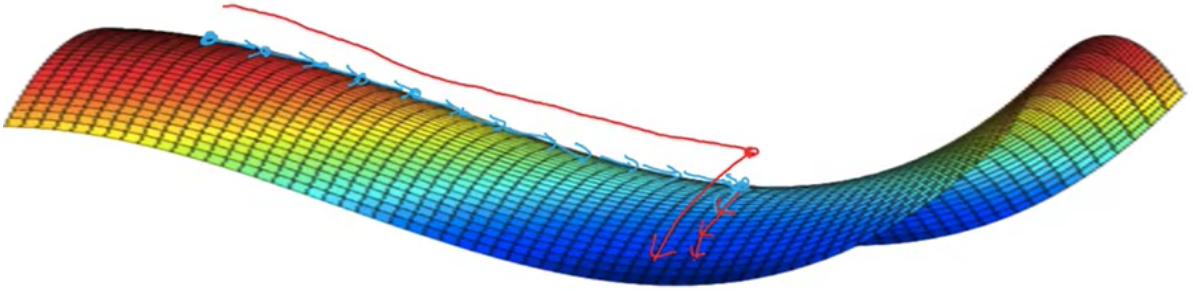
f)Local optima

- unlike ml models are have low dimensional, dl models deal with high dimensional data
- Lets say our model deals with 20000 parameters a local minima can occur in our cost vs parameters hyper plot only if a minima occurs in the sectional plots of each dimension
- The chance of all these sectional plot having a minima is $\frac{1}{2}^{20000}$ which is extremely low.
- One global minima will exist but the chances of our model running into a local minima is negligible.

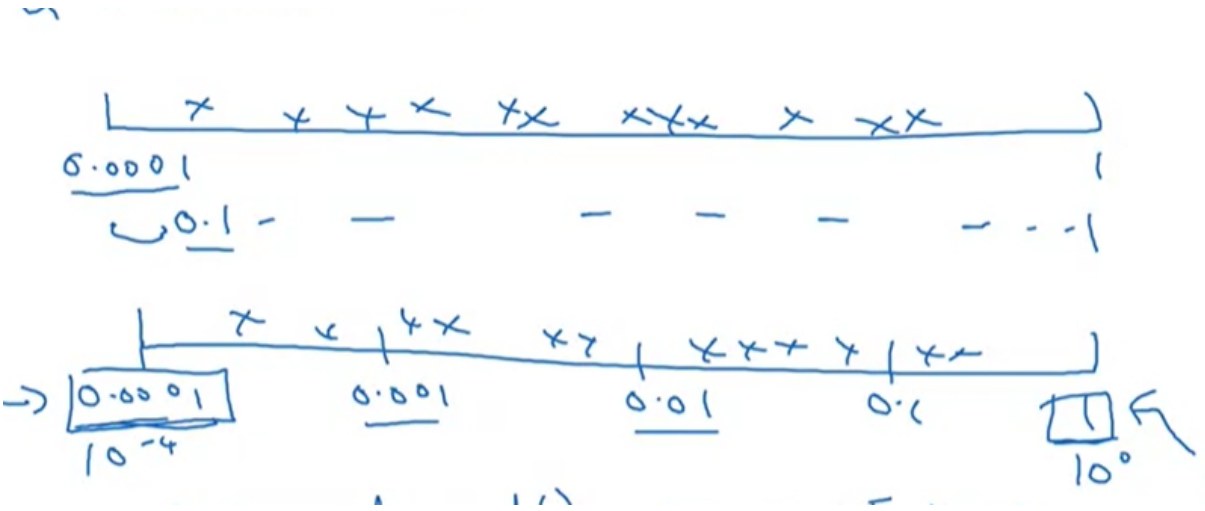
- Instead our model can run into multiple saddle points where some of the parameters reach a minima but all of the do not.

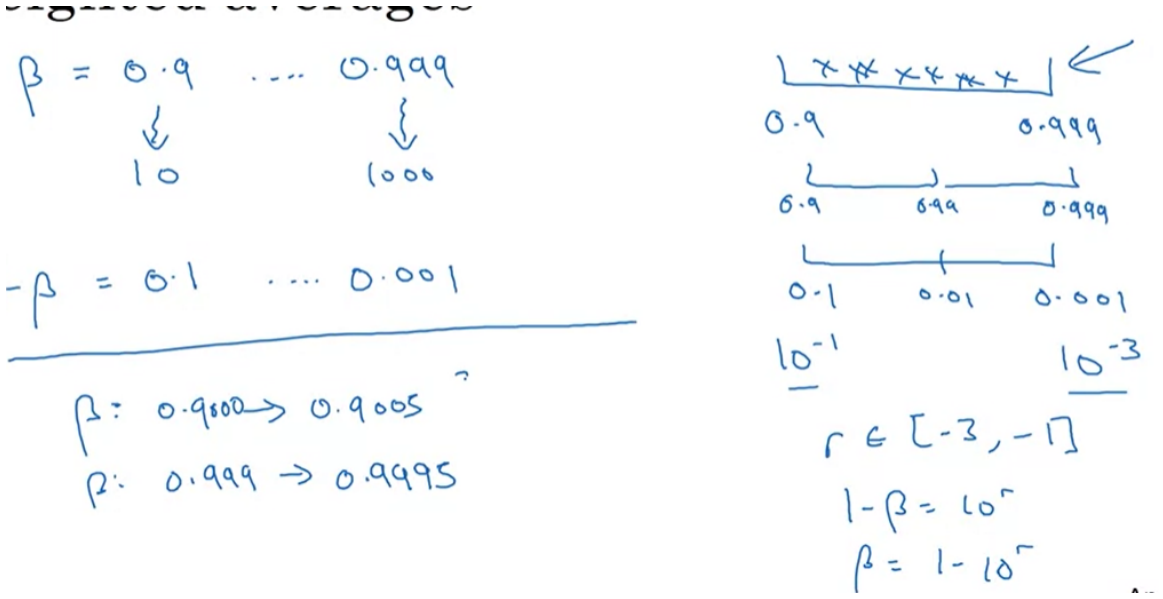


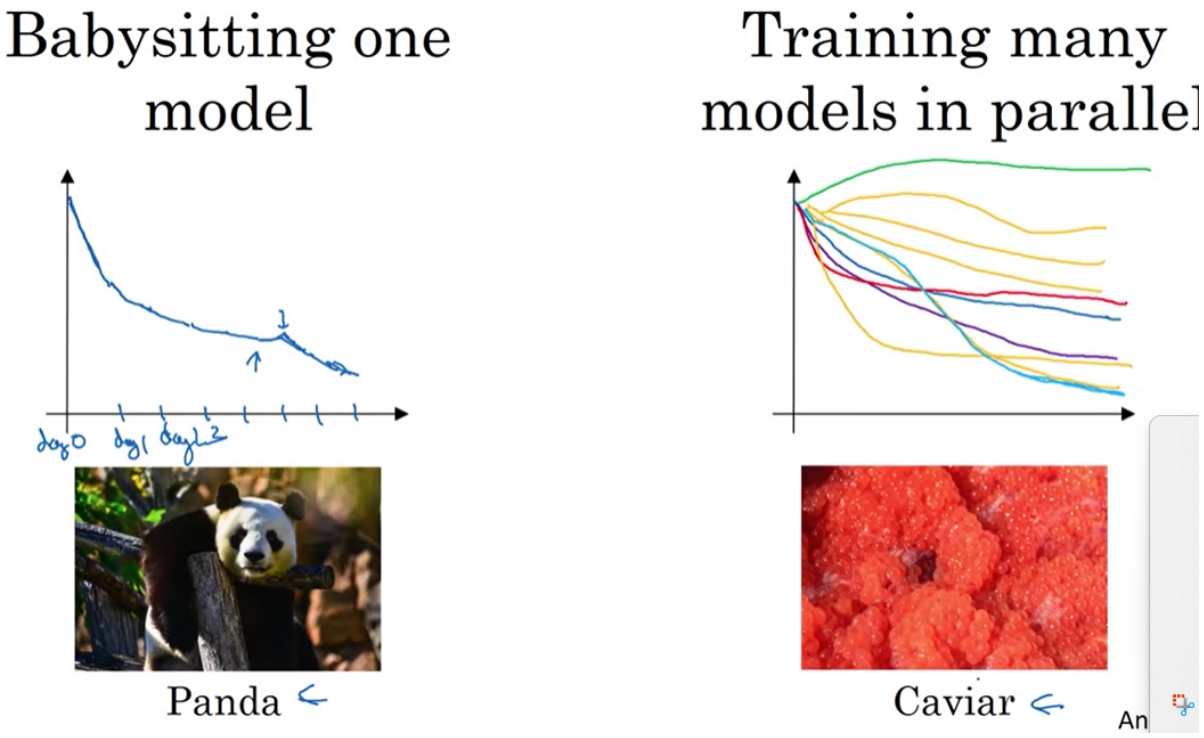
- plateaus instead slowdown our models



g)Hyperparameter Tuning

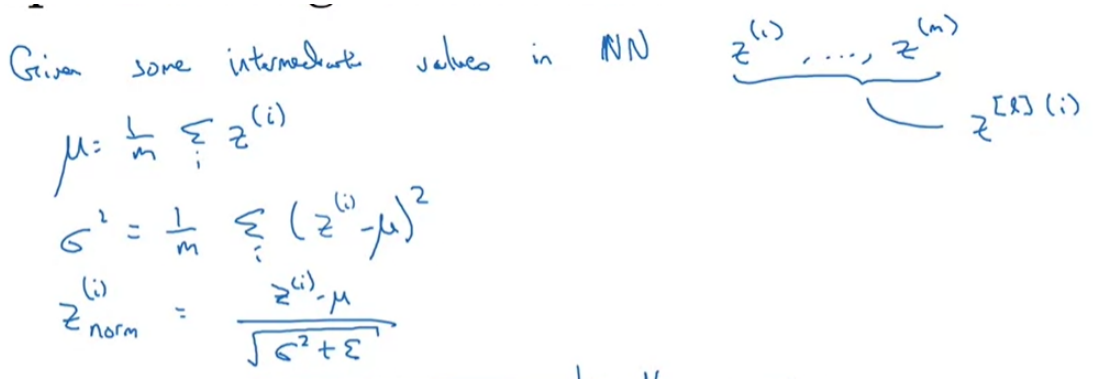




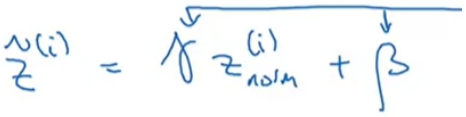


f)Batch Normalization

- In batch normalization we normalize the values of z^i which come out of a layer.

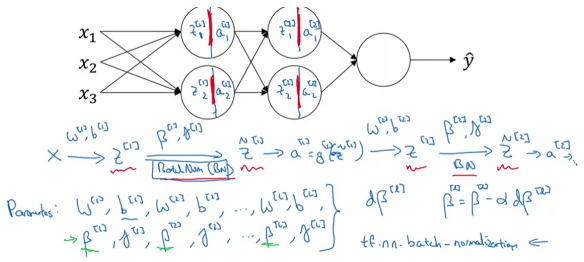


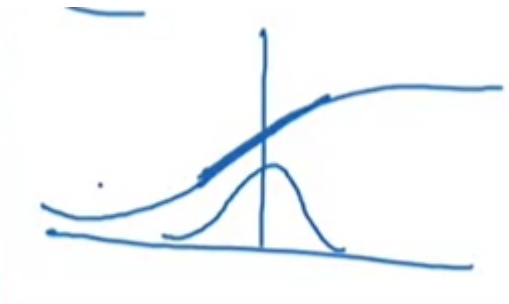
- But we dont use z norm directly instead we use



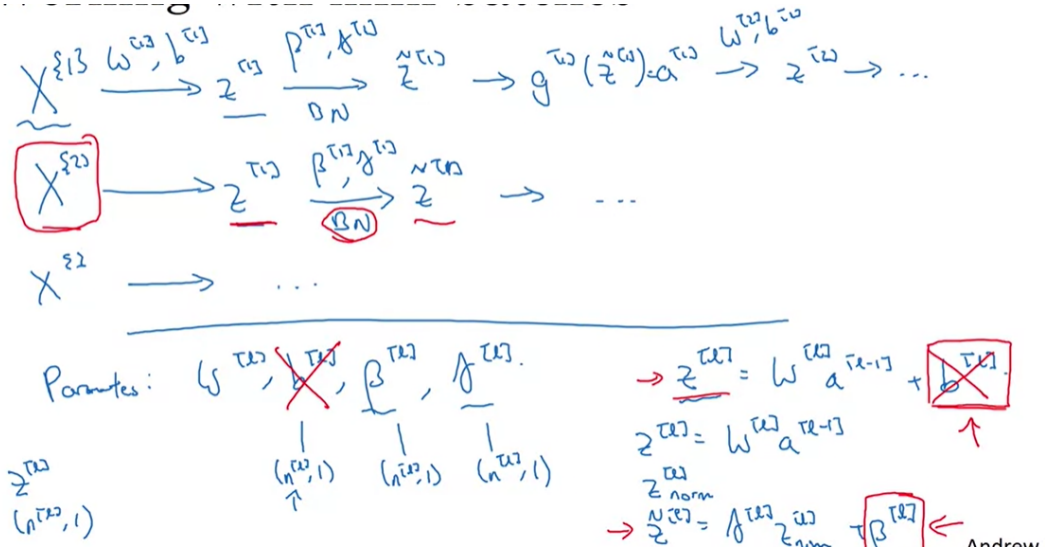
where gamma and beta are **Scale and Shift-**learnable parameters of the model

- we cant directly use , z norm is centered at z=0 with variance 1, hence only the lienar part of the activation function will be used which makes the model linear.

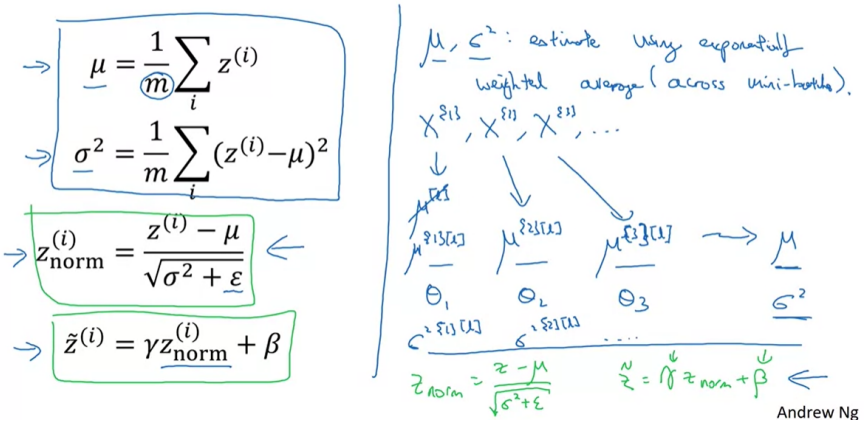




- adding batch normalisation to a batch gradient descent
- when we use batch normalization bias becomes reductive as z norm won't change when bias is changed(if you add or subtract a constant mean remains same).
- Beta has same dimension as b,z,z norm which is [n(l),m] j must have [n(l),n[(l)] but its some how different in coursera



- Covariance shift-occurs when the task is same but the model is not able to generalize to a different distribution of data.eg)black cat vs dogs and white cat vs dogs
- if we learn a mapping X→Y if distribution of X changes then we have to retrain y.
- occurs because the inputs received by the hidden layers in the middle vary a lot leading to them being highly dependent on the initial layers
- Batch normalizing ensures that the inputs that each layer receives always have a fixed mean and variance i.e more stable making the later layers more robust.
- Batch Noramalization does different changes in different mini batches leading to slight regularization effect.
- When we Normalize the values in input layer it is done outside of the model as a transformation on the input data. Hence it can be done both in test and train data
- However batch normalization requires the data of all instances in a batch and if we don’t normalize the values of z during test time the outputs may change a lot.


$\mu = \frac{1}{m}\sum_i z^{(i)}$
$\sigma^2 = \frac{1}{m}\sum_i (z^{(i)}-\mu)^2$
$z^{(i)}_{\text{norm}} = \frac{z^{(i)}-\mu}{\sqrt{\sigma^2+\varepsilon}}$
$\tilde{z}^{(i)} = \gamma z^{(i)}_{\text{norm}} + \beta$
μ, σ^2: estimate using exponentially weighted average (across mini-batches).
θ_1
θ_2
θ_3
$z_{norm} = \frac{z-\mu}{\sqrt{\sigma^2+\varepsilon}}$
$\tilde{z} = \gamma z_{norm} + \beta$

Andrew Ng