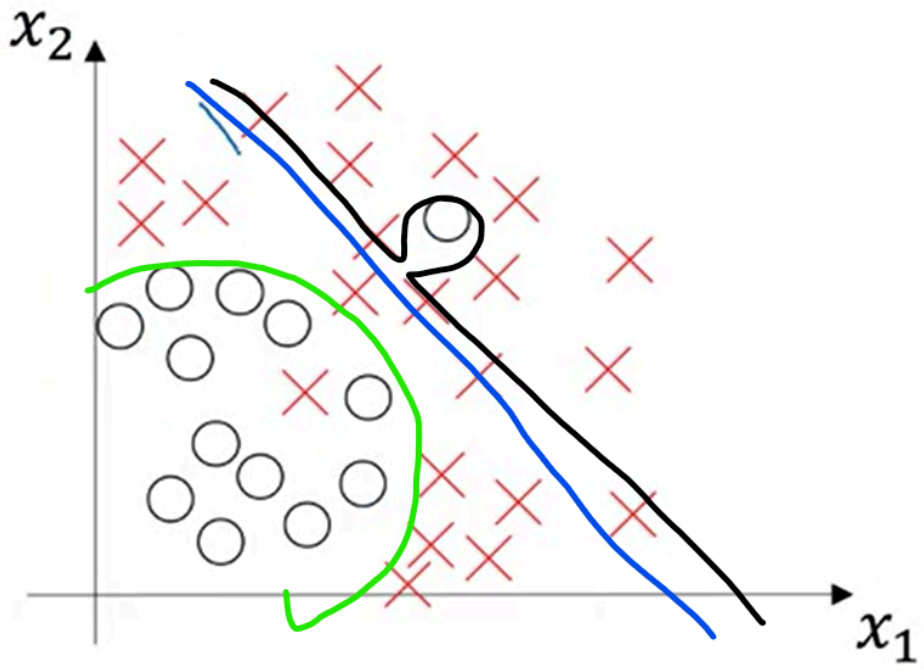
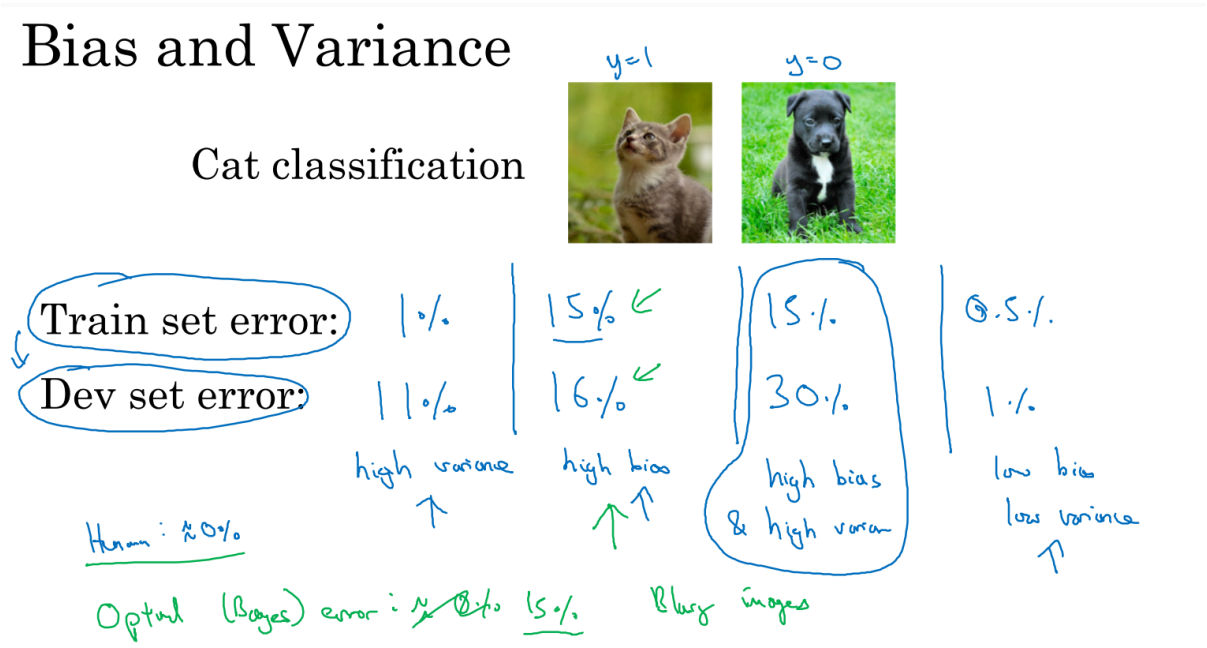
Week 1 c2



Blue and black line is for **under fitting a**nd **over fitting** respectively
The green line perfect fitting



High Variance is caused because of high error in training dataset
High bias Is caused cus of high error percentage of test data and greater difference from train error
If human/ **Bayes error** is greater, then train error can be under Correct bias if its high too

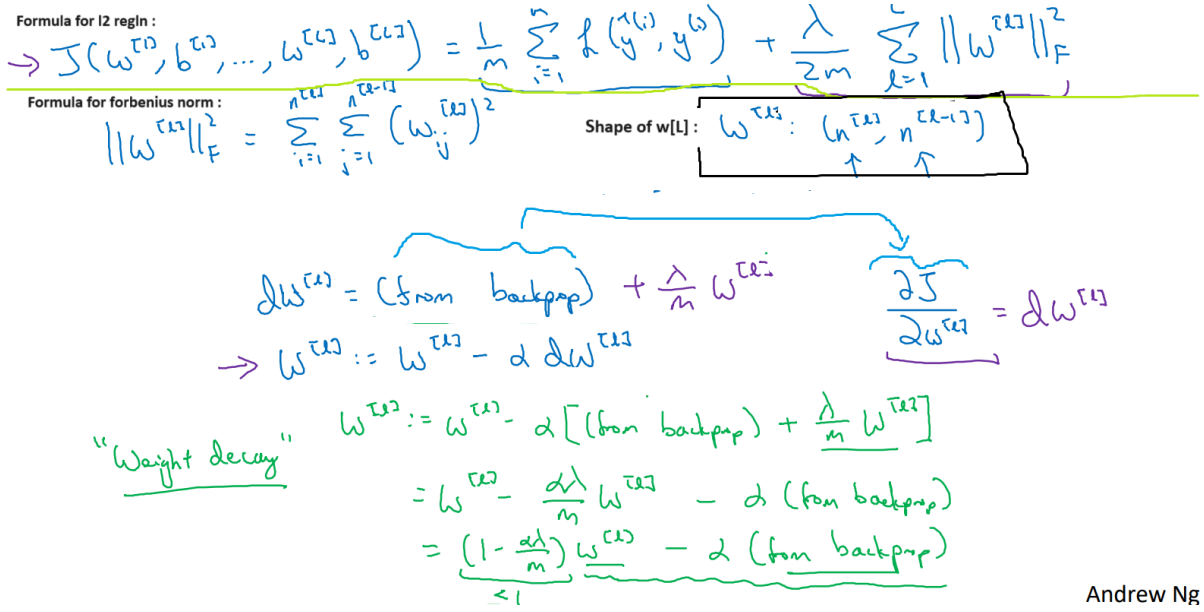
Solutions for,

High Bias (Underfitting): 1. Increase model complexity 2. Add more features 3. Reduce regularization

High Variance (Overfitting): 1. Increase training data 2. Reduce model complexity 3. Regularization 4. Dropout

For logistic regression:

As per **L2 regularisation -**



lambda is the regularization parameter.'Lambd' is used in code for lambda

||w||^2 represents the squared norm of the parameter vector w.

L2 regularization is the most common type, cus it helps prevent overfitting and reduces variance in the network

The Frobenius norm itself is a method of calculating the norm of a matrix.

When used in the context of regularization, it is part of the L2 regularization technique called the weight decay - the last term in the l2 formula given above

How does regularisation reduce over fitting

Using the weight decay in L2 , We can increase lambd That in turn reduces w

Inverse proportionality and ,

If w=0, Most of the hidden units create lessor impact cus the weight have reduced this results in a simple network which looks like Logistic regression

Another reason that regularisation helps is cus Increase in Lambda decreases w which leads to smaller Z that means

the activation function tanh Decreases and if nonlinear activation function is less,thus the layers become linear leading to simple network

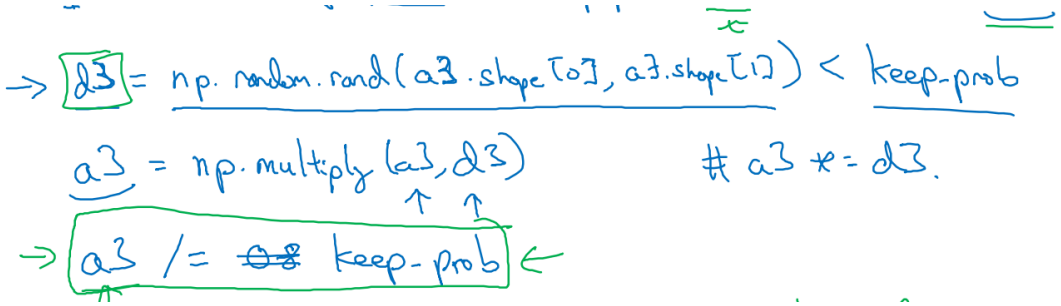
This brings over fitting closer to under fitting

dropout

Dropout is a regularization technique implemented during training time

It drops out/ zeroes out Some hidden units every time the training set is passed

Inverted dropout is a common technique



dropout mask (d), here, d3 Decides What to zero out in both forward and backward prop

Chances of D3 to be one is 0.8 if keep_prob = .8, So there are 20% chances for any neuron to drop out and 80 for keeping it.

the activations from the previous layer (a3) are multiplied element-wise with a d3. d3 is a binary matrix of the same shape as a3, where each element has a probability of being set to 1 (kept) or 0 (dropped out). This element-wise multiplication effectively zeros out (drops out) a certain percentage of the activations in a3, based on the d3

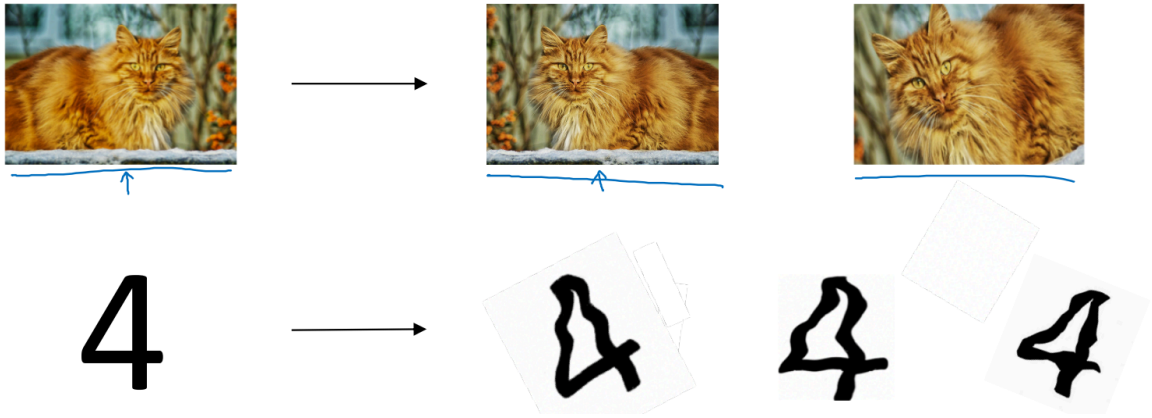
The activations are still maintained due to scaling technique,i.e., dividing a3 by keep prob as per Inverted drop out

this leads to easier test time

Why does drop out work?

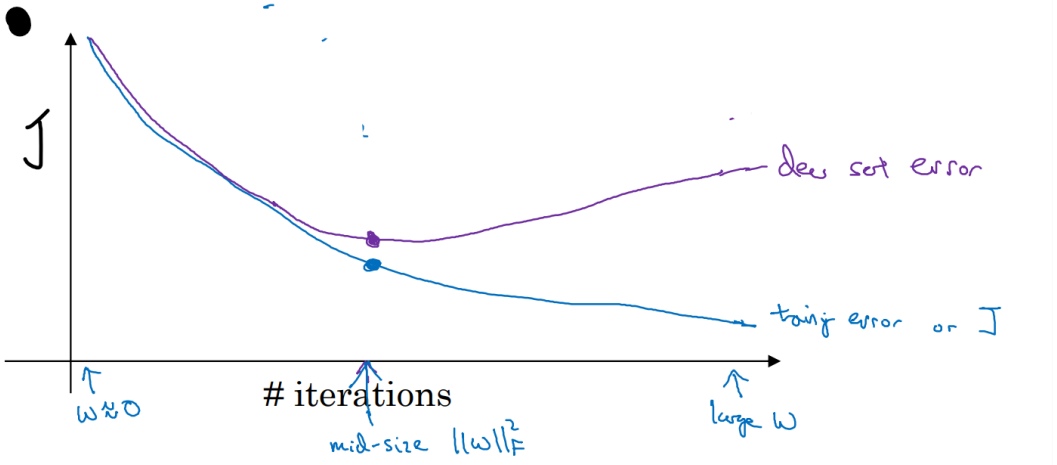
the output unit is less reliant on any specific input/unit before this unit so it spread out its weights and give a little weight to each input, w is lessened, now similar to what happened with logistic regarding less w preivously takes place now- simple network

Data augmentation



exposing the model to a wider range of variations and patterns of the original data so the model learns to generalise better

Early stopping



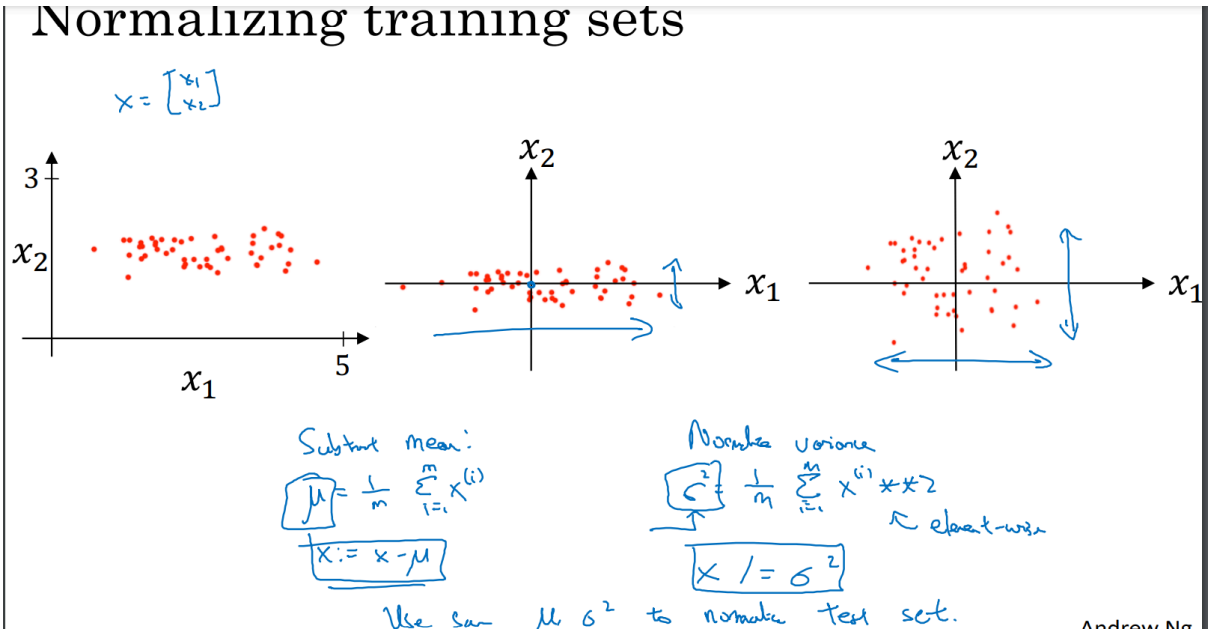
Stopping at mid size w rate suppose, lessens norm of w, thus helping regularisation

Early stopping and data augmentation are Regularization technqs

Normalising training set

As shown in the image below, on subtracting we get 2nd plot n on normalising variance, we get 3rd plot

The distribution of the parameters is symmetric in last plot, i.e., they are normalised



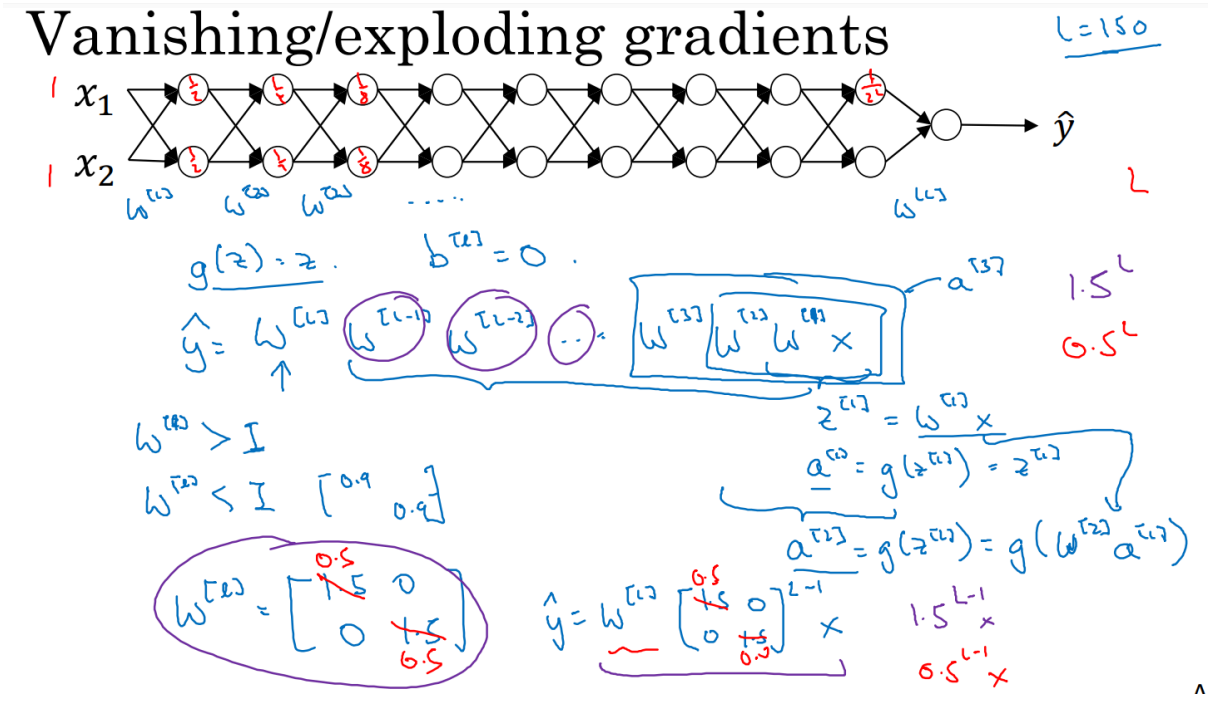
Normalise variance -

Sigma = 1/m * sum(x_i^2) , element-wise . x/=sigma [not sq sigma]

Why normalisation is needed -

If x1 ranges in 1 to 1000, and x2 in 0 - 1, they will cause difficulty in w computations

use the same mean and sigma values calculated from the training set for test set



Neglect b[L], Y hat original formula = …w[L].w[L-1].X , as seen in the bottommost eg, it gives us

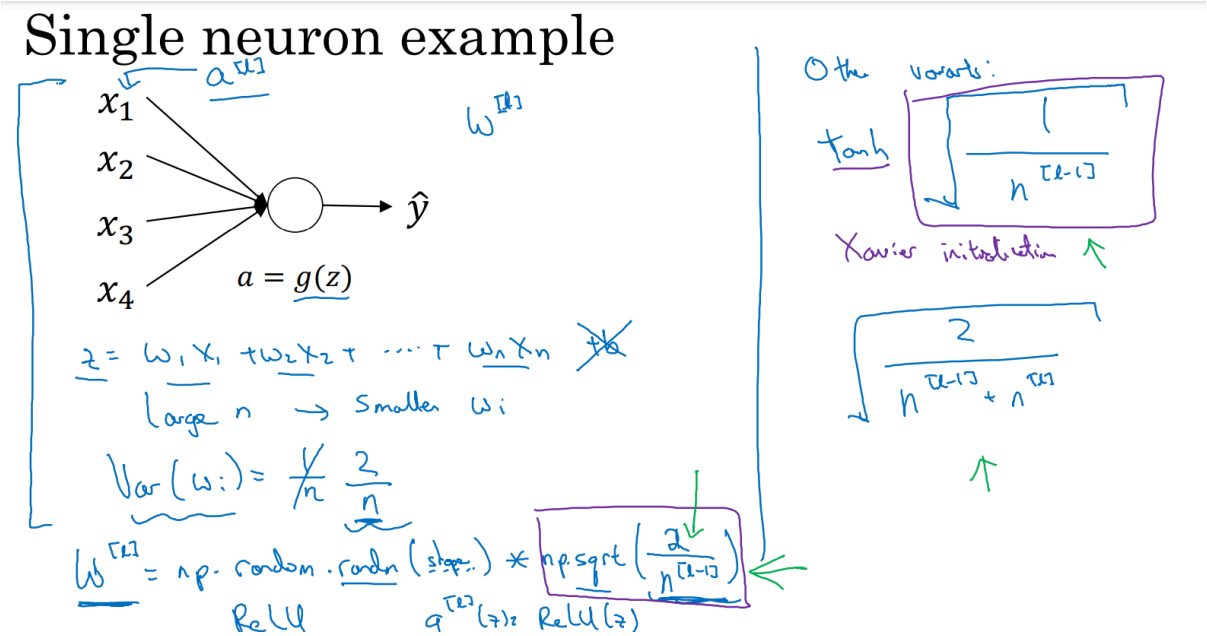
w[L].(X)*(0.5)^L-1 ; i.e., x or the parameters now get divided by powers of 0.5 and

as seen in the NN at top, by the end of the nn, the nn are a power of 1/22 smth

weight initialization value slightly lesser than 1 can solve the prb of vanishing gradients - occurs when the gradients become very small as they propagate backward

Gradient descent takes longer if gradients are very minute

This tactic helps to increase /decrease activn funcs and gradients exponentially inc/dec. Its inc if w<I, I is identity mx



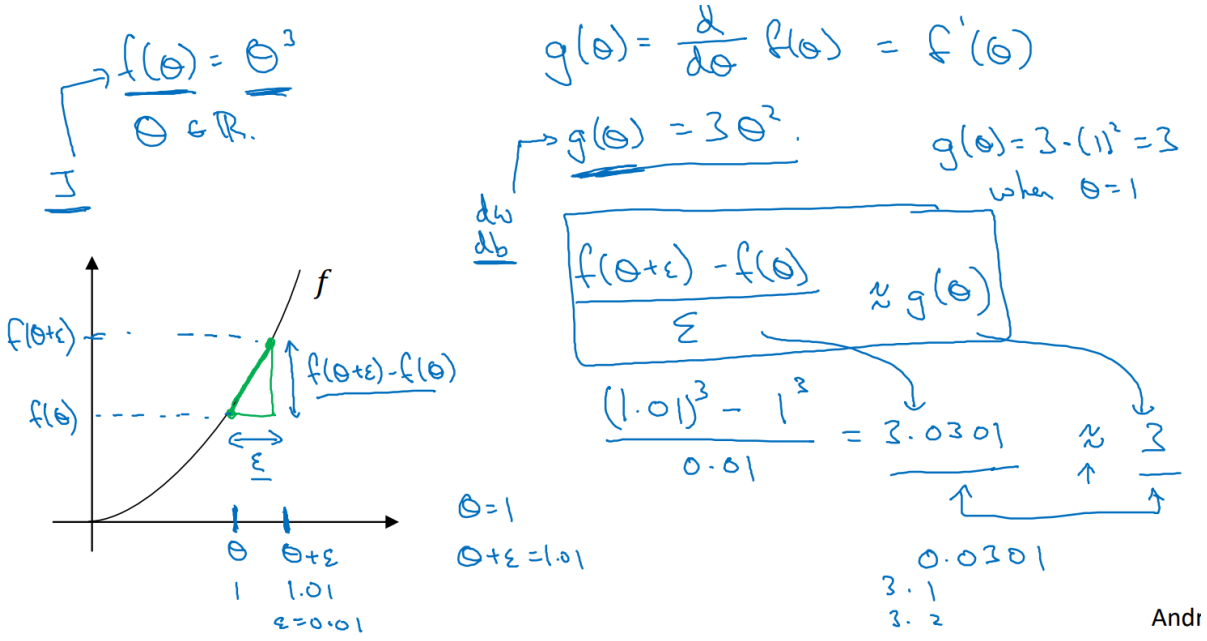
formula for initializing the weight matrix W for layer L is given and

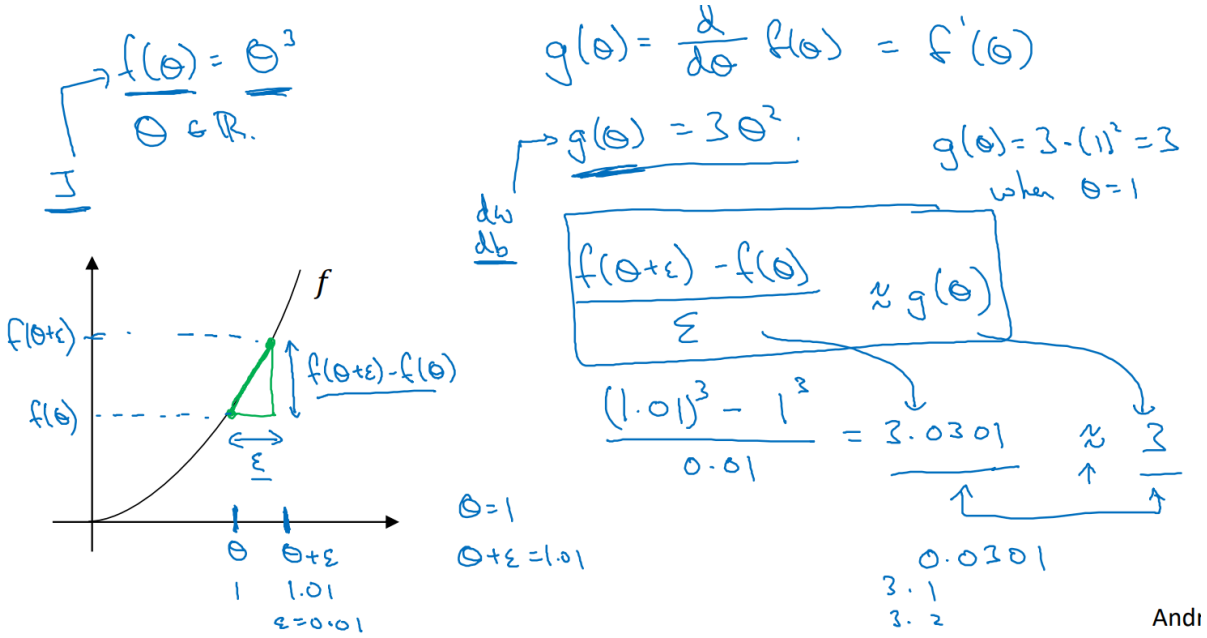
Xavier initialization provides a different scaling factor

Checking your derivative computation

formula used in gradient involves evaluating the function at two points: one slightly to the right and one slightly to the left of the current point. The difference in the function values at these points is divided by the distance between them to approximate the gradient.

This formula helps us verify the correctness of our implementation of backpropagation





Gradient check for a neural network

Take w[1], b[1],.., w[l], b[l] and reshape into a big vector

Gradient checking verifies the correctness of the implementation of backpropagation in neural networks.

It approximates gradients and compares them to the gradients calculated using backpropagation to ensure accuracy.

Gradient checking implementation notes

- Don't use in training – only to debug

- If algorithm fails grad check, look at components to try to identify bug.

- Remember regularization.

- Doesn't work with dropout.

- Run at random initialization; perhaps again after some training.