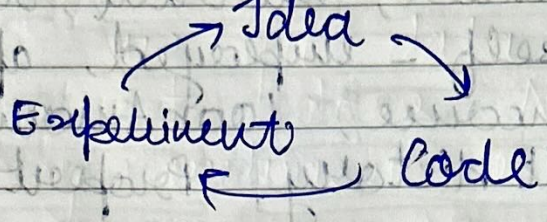
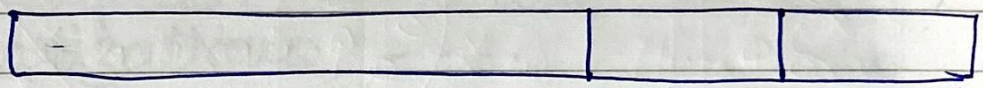
Course 2: IMPROVING DEEP NEURAL NETWORKS

- Applied ML is highly iterative process.



- Intuition from one application to another are not transferable; it depends on data, CPUs (GPUs) used & many other factors.

* Train/Dev/test sets:



- Train set
- Hold-out cross-validation set
- Development set 'Dev' set
- Test Set

Previous Practise: train test — 70/30; train dev test — 60/20/20

However, with the amount of data (upto 1,000,000 points) the size of dev & test set is decreasing rapidly.

Eg: 1,000,000 : 10,000, 10,000

{ 98/1/1 %.
{ 99.5/0.4/0.1 %.

~~Thumb~~

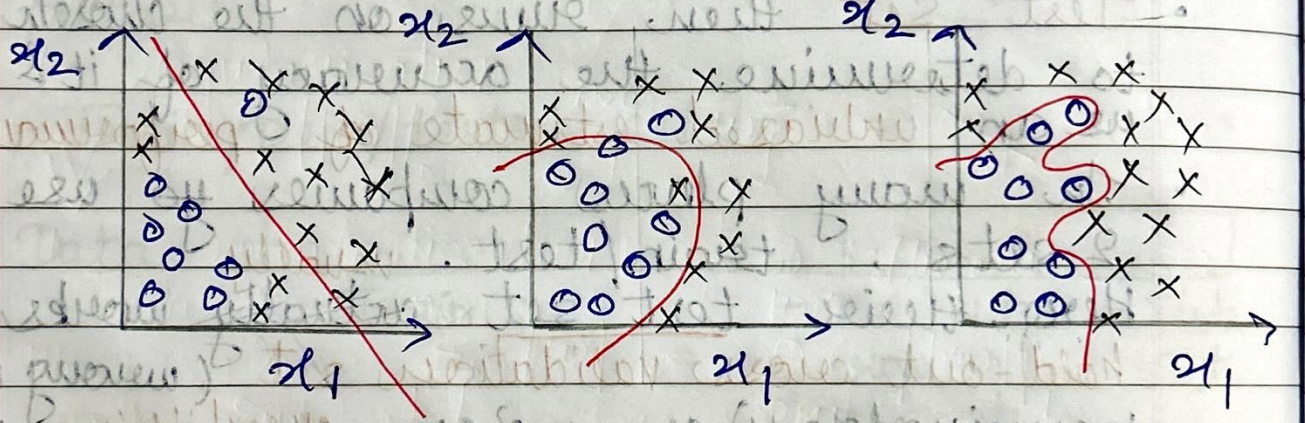
- Thumb Rule: Make sure dev & test sets come from same distribution.
- No test set might also be okay, (only dev test).
- Dev ~~Test~~ Set is set up to evaluate data on different models and find the best one for your dev set (and thus train set).
- Test set then runs on the chosen model to determine the accuracy of it & gives us an unbiased estimate of performance.
- In many places companies ~~go~~ use only 2 sets, train/test. usually Here, their test set ~~actually~~ works as hold-out cross validation set (wrong use of terminology) as we're overfitting the test ~~train~~ set data & not computing an unbiased estimate.
- However,

- ~~Dev Set:~~

TRAIN SET	DEV SET	TEST SET
• Max. ratio of our data set as we're training the model here.	Evaluate this data on different models to find perfect one for data. Estimate is not unbiased as we iterate through the diff. models to BEST fit the data.	Evaluate this data on the chosen model & get unbiased estimate.

- Thus, places where train/test is used in place of train/dev, it is seen as an okay practice when we don't need a completely unbiased estimate.
- Because we fit data to the dev set, the dev set does NOT give a completely unbiased estimate.

* ~~Boost~~ Bias - Variance:



High Bias → underfitting | Just Right | High Variance → overfitting

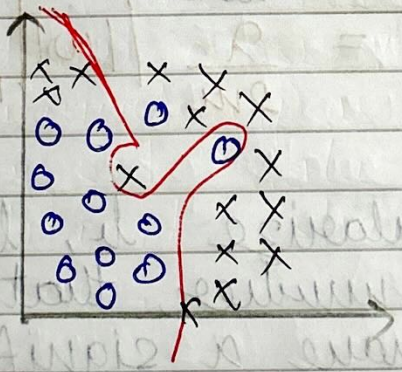
				low bias, low variance
Train Set Error	1%	15%	15%	0.5%
Dev Set Error	11%	16%	30%	1%
	high variance → here the model is overfitting the train set	high bias as it is not even fitting the training set properly →	high bias, high variance	

The below analysis is based off the fact that the human error in differentiating cat/dog is 0.
Thus, optimal (Bayes) error (here) : 0

- With the change in Bayes error the analysis may change.

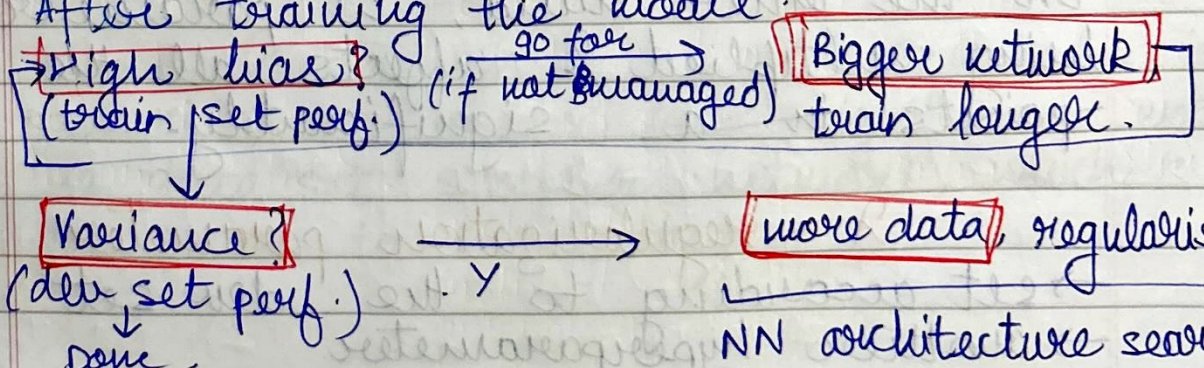
- Train Test Error:
in comparison to Bayes error tells us ~~if~~ ~~it~~ how well we're fitting training set & gives an idea about bias

- Dev Set Error:
depending on how higher it is from train set error we understand the variance of the model.



high bias,
high variance.

* Basic Recipie for ML :
After training the model :



High bias ⇒ more data won't help.

- As we've more option for managing bias & variation we do not need to perform a bias-variation trade-off.

* Regularisation in Logistic Regression:
Applying regularisation to Logistic Reg.:

$$J(w,b) = \frac{1}{m}\sum_{i=1}^{m} \mathcal{I}\left(\hat{y}^{(i)}, y^{(i)}\right) + \frac{\lambda}{2m}\|w_2\|^2$$

L_2 regularisation:

$$\|w\|_2^2 = \sum_{j=1}^{n_x} w_j^2 = w^T w$$

L_1 regularisation: → w is/becomes sparse(0) here.

$$\frac{\lambda}{2m}\sum_{i=1}^{n_x}|w| = \frac{\lambda}{2m}\|w\|_1$$

- We can also regularize b, but it's actually just a number that is added. Thus, does not have a significant role in overfitting.

- On the other hand, w parameter is being multiplied & affects all the data-points in a significantly.

λ = regularization parameter
→ set according to the dev set
→ another hyperparameter

* Regularization in a NN

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

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

Frobenius Norm (sum of square of elements of a matrix)

$w \rightarrow (n^{[l-1]}, n^{[l]})$ dimension.

Gradient Descent:

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

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

$$= w^{[l]} - \alpha \left[(\text{from backprop.}) + \frac{\lambda}{2m} w^{[l]} \right]$$

$$dw^{[l]} = w^{[l]} \left(1 - \frac{\alpha\lambda}{m}\right) - \alpha (\text{from back-prop.})$$

Thus, L_2 regularization $\equiv$ weight decay owing to $\left(1 - \frac{\alpha\lambda}{m}\right) w^{[l]}$.

- If $\lambda >>$ high, $w^{[l]} \approx 0$, that is, few of the features lose significance, making the data set simpler network, which is less prone to overfitting.

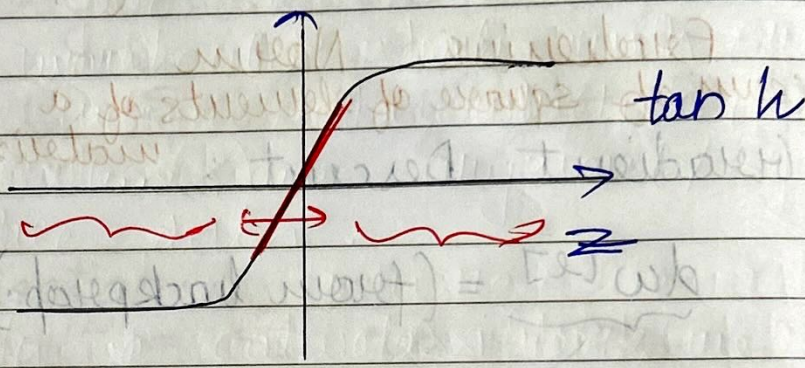
$\lambda \uparrow$ new $w^{[e]} \downarrow$

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

$\Downarrow$

$z^{[e]} \downarrow$

→ thus activation function will be relatively linear



* Dropout Regularization:
- Inverted Dropout

We realise that in d3 matrix we find hidden features to zero-out (randomly) at each pass.

keep_prob = 0.8 (here)

```
d3 = np.random.rand(a3.shape[0],
                    a3.shape[1])
d3 < keep_prob
a3 = np.multiply(a3, d3)
a3 /= keep_prob
```

ensures to not add extra scaling during test time.

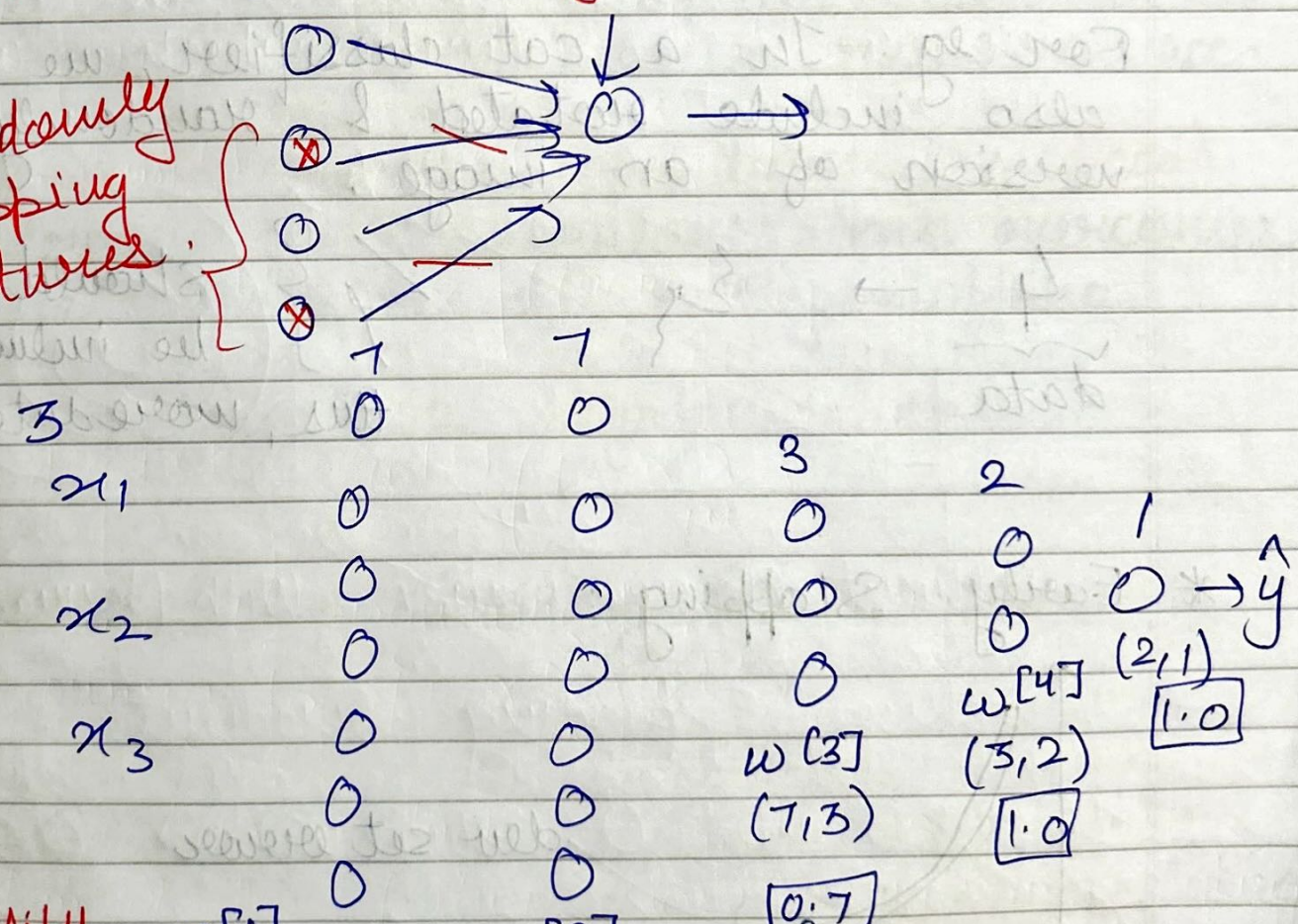
to standardise / give the desired output despit zeroing out 20% (0.2) features.

We do not use dropout at test time as that would give us a random output & introduce noise.

* Why does drop-out work?
(as randomly any can be dropped)

- Intuition: Can't rely on any one feature, so have to spread out weights.
  → shrinking weights
  → dealing with overfitting

- Drop-out was considered an adaptive form of L2 regularisation.



Downside of dropout is that the cost function J is not well-defined thus with each iteration of gradient descent we can get different loss.

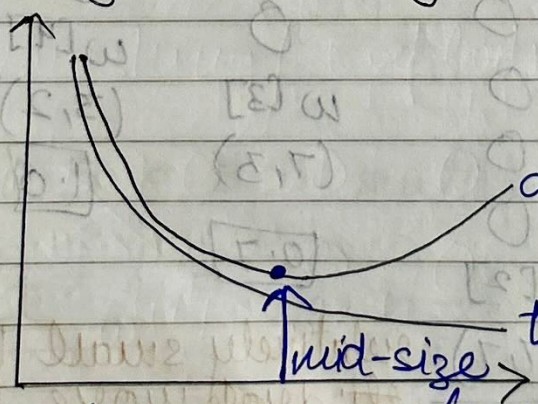
* Data Augmentation:

- Technique for regularisation.
- If we want to deal with overfitting, we need more data, which can be expensive. Thus, we augment/synthesise the available data.

For eg: In a cat classifier, we can also include rotated & randomly zoomed version of an image.

4 → {4} {4} should also be included.
data
thus, more data.

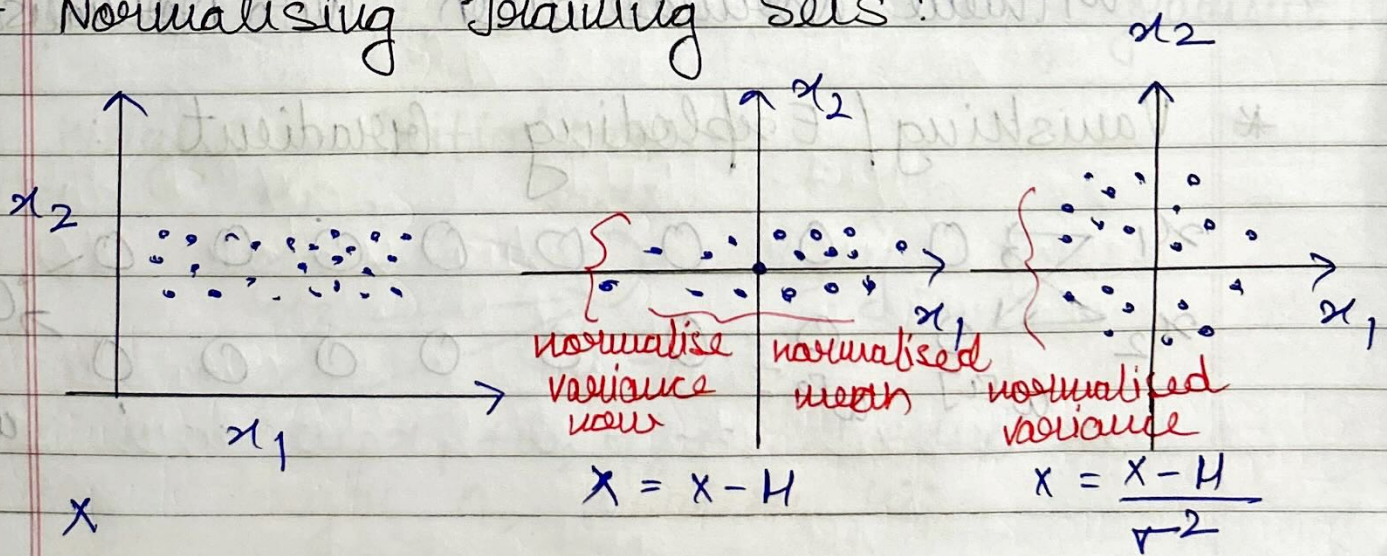
* Early Stopping:



* Orthogonalisation:

- Taking a particular set to solve a given problem, one problem at a time.
- Downside of early stopping as it tries to solve the problem of overfitting but affects the cost func^n (reduces its iterations) and gradient descent in process.
- Advantage of early stopping is that it tries all range of values for w in a single gradient descent, which in case of L2 regularisation we'll have to iterate many times as acc. to adjusting lamda.
  (which increases computation time/load).
- However, L2 regularisation overcomes the downside of early stopping.
- Preferred: L2 regularisation

* Normalising Training Sets:

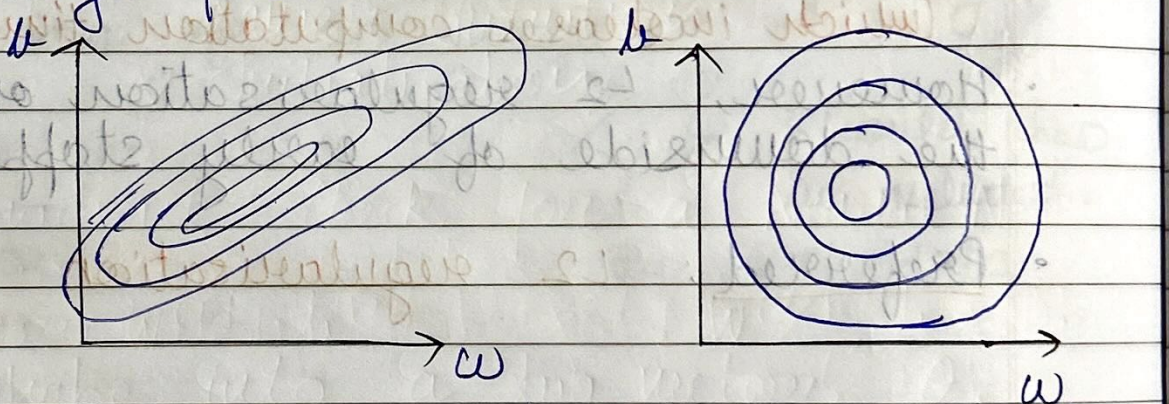


$$\sigma^2 = \frac{1}{m} \sum_{i=1}^{m} x^{(i)} ** 2 ;$$

→ use **same** μ, σ^2 for test set

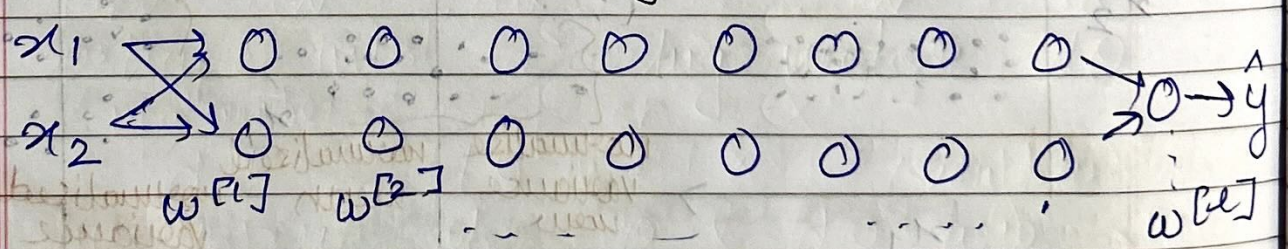
* Why Normalise?

- Cost function (J) is very varied & oscillating without normalisation, thus then we'll have to use a slow learning rate (α).
- On normalisation, however, we get a symmetric -like- func^n which will converge faster



Without Normalisation With Normalisation

* Vanishing / Exploding Gradient :



let $b^{[l]} = 0$, $g(z) = z$

$\hat{y} = w^{[l]} \cdot w^{[l-1]} \cdot \ldots \ldots w^{[2]} w^{[1]} X$

$g^{(1)}(z) = a^{[1]}$

$a^{[2]} = w^{[2]} a^{[1]}$

$a^{[2]} = w^{[2]} w^{[1]} X$

thus, $\hat{y} = a^{[l]}$

~~$= w^{[l]}$~~

let $w^{[n]} = \begin{bmatrix} 1.5 & 0 \\ 0 & 1.5 \end{bmatrix}$ (a little greater than identity matrix)

$n : 1 \to l$.

$\hat{y} = w^{[l]} \begin{bmatrix} 1.5 & 0 \\ 0 & 1.5 \end{bmatrix}^{l-1} X$ (0.5)

$\Rightarrow$ very large value, & gradient

very small value, & gradient

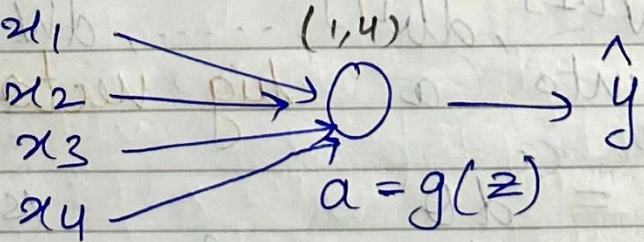
$w^{[n]} > I \Rightarrow$ exploding gradient

$w^{[n]} < I \Rightarrow$ vanishing gradient

I = identity matrix.

We observe that deep NN can have the problem of exploding & vanishing gradient.

* Weight Initialisation:



$z = w_1 x_1 + w_2 x_2 + \ldots + w_n x_n$.

larger n, smaller w_i

$Var(w_i) = \frac{2}{n}$ } ReLU

$= \frac{1}{n}$ } tanh & few other.

$w^{[l]} = np.random.randn(shape) * \underbrace{np.sqrt\left(\frac{2}{n^{[l-1]}}\right)}_{\text{for ReLU (most used)}}$

Other Variants:

$\tanh = \underbrace{\sqrt{\frac{1}{n^{[l-1]}}}}_{\text{Xavier Initialisation}}$ or $\sqrt{\frac{2}{n^{[l-1]} + n^{[l]}}}$

* Gradient Check for a Neural Network:

1. Take $W^{[1]}, b^{[1]}, \ldots, W^{[l]}, b^{[l]}$ and reshape into a big vector θ. (concatenate)

$J(W^{[1]}, b^{[1]}, \ldots, W^{[l]}, b^{[l]}) = J(\theta)$

2. Take $dW^{[1]}, db^{[1]}, \ldots, dW^{[l]}, db^{[l]}$ and reshape into a big vector $d\theta$ (concatenate)

$dJ = d\theta$ (should be)

- dimensions of θ & $d\theta$ are same.

Checking:

for each i:

$$d\theta_{approx}^{[i]} = \frac{J(\theta_1, \theta_2, \ldots \theta_i + \varepsilon, \ldots \theta_n) - J(\theta_1, \theta_2, \ldots \theta_i - \varepsilon, \ldots \theta_n)}{2\varepsilon}$$

$d\theta_{approx} \approx d\theta$.

$\approx \frac{dJ}{d\theta_i} \left(\frac{\partial J}{\partial \theta_i}\right)$

check: $\frac{\| d\theta_{approx} - d\theta \|_2}{\| d\theta_{approx} \|_2 + \| d\theta \|_2}$ — euclidean distance b/w $d\theta_{approx}$ & $d\theta$.

euclidean length of $d\theta_{approx}$

$\approx 10^{-7}$ → great

10^{-5} → may get better, check

10^{-3} → worry

* Implementation Notes:

- Don't use in training → only to debug (as it is too slow).
- If algo fails grad check, look at components to try to identify bug.
- Remember regularisation term in J, if using regularisation in train set.
- Doesn't work with dropout, as it randomly changes the features in every iteration. If using dropout:
  - → turn-off dropout
  - → use grad-check to ensure algo is correct
  - → turn on dropout.
- Run at random initialization, perhaps again after some training, so that w & b wander away from 0.

not used often