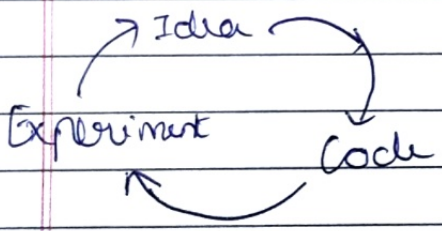
IMPROVING DEEP NEURAL NETWORK:

HYPERPARAMETER TUNING, REGULARISATION

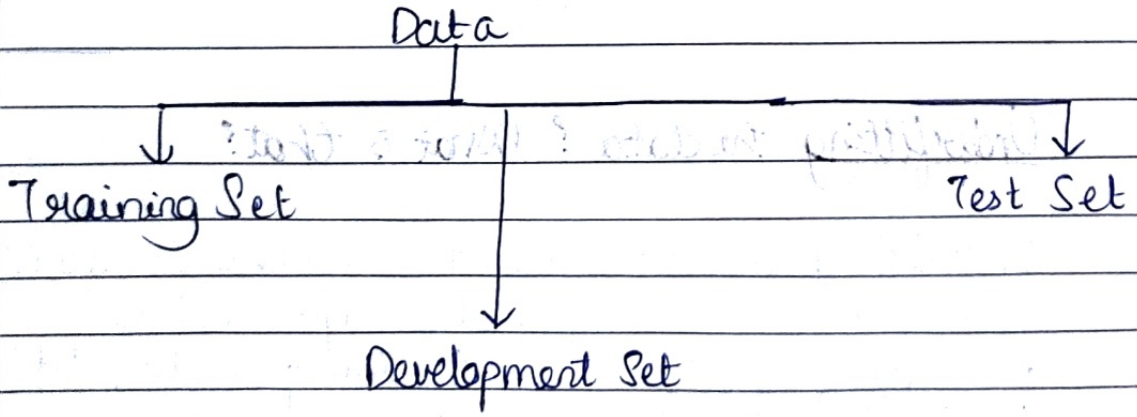
OPTIMISATION

Video 1: Train / Dev / Test Sets

* Dap learning is a highly iterative proces. You must decide variaes factors like: · layers
· hidden unit
· learning rates
· activation functions.



*



Eg. of spliting data

70 % / 30 %
↑ train ↑ test

60 % / 20 % / 20 %
Train Dev Test

* The goal of the development set is to see which algorithm works best.

* When there is ~~a big~~ very huge is a huge dataset, the dev & test sets are a small % age of the total data

* The goal of test set is that it gives you an estimate of accuracy

* Dev and test set should come from same distribution

* Since test set gives you an unbiased estimate of performance, it's not necessary.

Video 2: Bias & Variance

(Bias variance trade off?)

Underfitting the data? What is that?

Underfitting occurs when the machine learning model is too simple to capture the underlying patterns in training data. It fails to learn relationship between input features & target variable effectively.

The models ~~performs well~~ performance but and on training data & ~~poorly~~ on unseen data is poor. It oversimplifies problem and doesn't capture complex relations.

Cause: using a very simple model
insufficient training data
inadequate featural selection

Solution: use a more complex NN

Overfitting the data! What???

Overfitting occurs when the machine learning model is too complex / flexible and it fits the training data closely, capturing unnecessary stuff like noise & random fluctuations rather than true patterns.

The model performs well on training data but poorly on unseen data.
It has low bias and high variance
It is sensitive to small changes in training data.

Bias: Bias refers to the error introduced by approximating a real world problem, which may be complex, by a simplified model. The model makes strong assumptions about the data resulting in underfitting

Variance: Variance refers to the model's sensitivity to small fluctuations or noise in the training data. The models are very complex and tend to fit the noise in the data rather than underlying patterns

how well ur fitting bias

Train set	1%	15%	15%	0.5%
Dev set	11%	16%	30%	1%
↓				
variance problem	high variance	high bias high	high bias high variance	low bias low variance

human ≈ 0% error

Bayes error - lowest possible error achievable

Analysis is relative to human error

Video 3: Basic Recipe for Machine Learning

Q1: high bias? (based training data performance)
Solution: (i) Bigger network
(ii) train longer
(iii) NN architecture that suits your problem
(iv) best

Q2. high variance? (based on dev set performance)
Solution: (i) more data
(ii) regularisation
(iii) best NN architecture

Bias Variance Tradeoff:

In the pre-deep learning era, we didn't have as many tools to just reduce or just reduce variance.

But today,

train a big network —— reduces bias

↓

get more data } reduce variance

↓

regularise }

Regularising Your Neural Network

Video 1: Logistic Regression

$w \in \mathbb{R}^{n_x}$, $b \in \mathbb{R}$

$$\min_{w,b} J(w,b)$$

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

λ: regularisation parameter

L2 regularisation:

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

L1 regularisation $\frac{\lambda}{2m} \sum_{i=1}^{n_x} |w_i|$

$\frac{\lambda}{2m} ||w_1||$

λ - hyperparameter that requires tuning

Neural Network:

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

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

i $\rightarrow$ no. of neurons in current layer $n^{[l]}$

j $\rightarrow$ no. of columns of weight matrix $\rightarrow$ no. of neurons in previous layer $n^{[l-1]}$

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

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

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

$= w^{[l]} \left(1 - \frac{\alpha\lambda}{m}\right) - \alpha$

$\uparrow$ weight decay

Page No.
Date

Video 3: Why Regularisation Reduces Overfitting

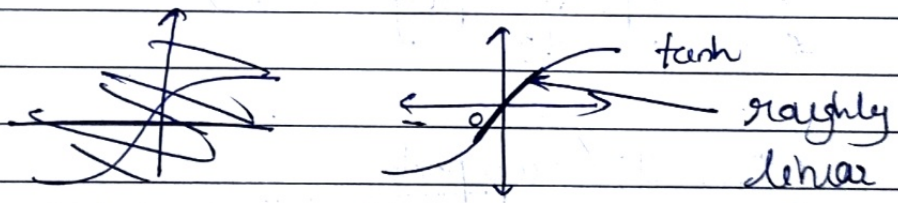
$$J(w^{[l]}, b^{[l]}) = \frac{1}{m}\sum_{i=1}^{i} \mathcal{L}\left[y^{(i)}, y^{(i)}\right] + \frac{\lambda}{2m}\sum_{l=1}^{l} ||w^{[l]}||_F^2$$

* If 'λ' is too large a lot of 'w's will be close to 0 which will make the NN simpler

* If λ is good enough it will just reduce some neural networks overfit

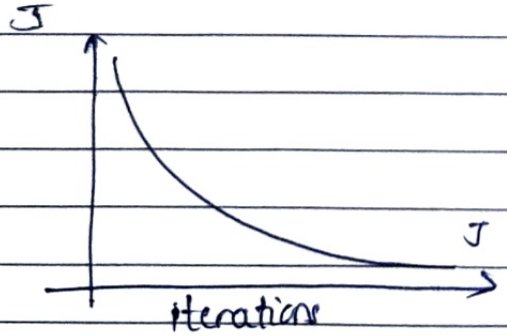
With tanh function:

* If λ is too large, w's will $\rightarrow 0$



* If λ is just right only some tanh functions will become roughly linear

Implementation tip:



Page No.
Date

Video 4: Dropout Regularisation

* Dropout Regularisation: It eliminates some neurons/weights on each iteration based on probability

* Inverted Dropout:

```
Keep_prob = 0.8
l = 3
# d3 = dropout vector for layer3
d3 = np.random.rand(a[l].shape[0], a[l].shape[1])
  < keep_prob
```

→ if number generated is less than 0.8, then it will be dropped

```
a3 = np.multiply(a3, d3)
# a3* = d3
```

```
a3 = a3/keep_prob
# This is so that expected value of a3 does not change
```

* Do not use dropout at test time. it adds noise to predictions

Video 5: Understanding Dropout

* One cannot rely on just one feature so you should spread out weights
* Input layer should have 1 or near 1 keep_prob because you don't want to eliminate a lot of features.
* Best way - have some layers where you apply dropout and some where you don't & then just one hyperparameter which is a keep_prob for the layers where you do not apply then
* Dropout is a regularisation technique to prevent overfitting.
* It is used in Computer Vision where overfitting occurs when a ML model performs well on training data but poorly on unseen data
* Drawback of dropout - cost function is not well defined.
  Solution: apply dropout during ~~test~~ training only.

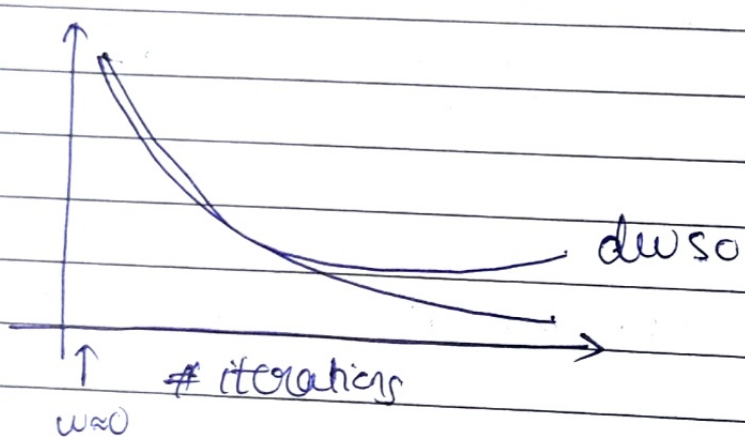
Page No
Date

Video 6: Other Regularisation methods

1. Data Augmentation:

* Flip pictures horizontally
  Apply random position / rotation
* In OCR - introduce rotations / distortions

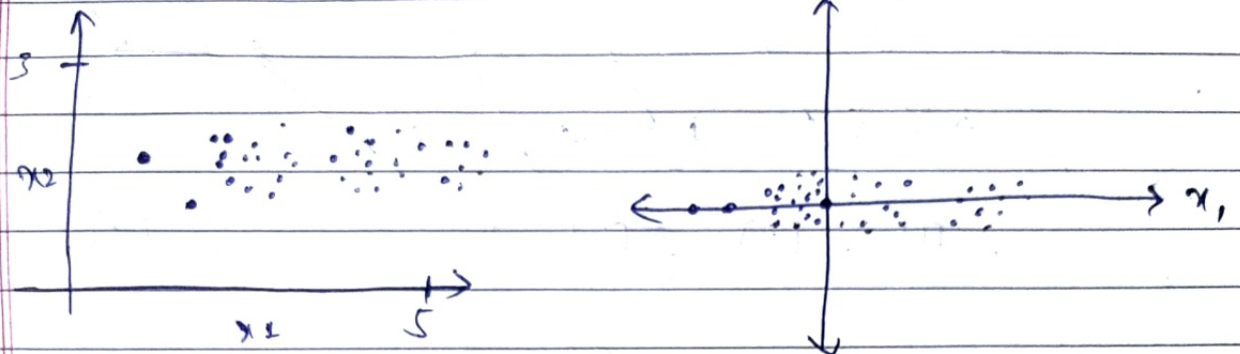
2. Early Stopping

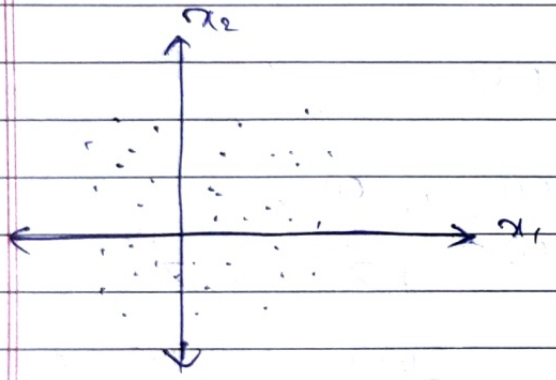


Page No.
Date

Week 1: Setting Up Optimisation Problem

Video 1: Normalise training sets





Zero out mean

Subtract mean:

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

$$X = X - \mu$$

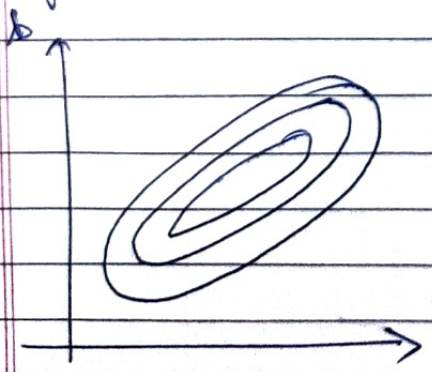
Normalise variance:

vector with variance of each feature $\longrightarrow$

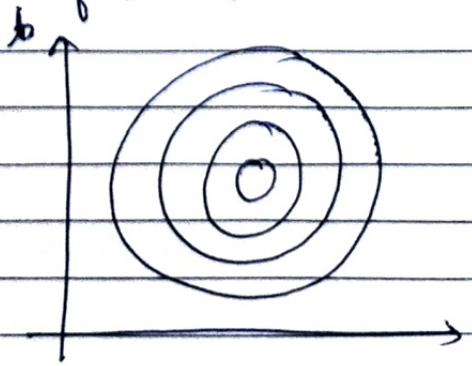
$$\sigma^2 = \frac{1}{m} \sum_{i=1}^{m} x^{(i)} ** 2$$

$$X /= \sigma^2$$

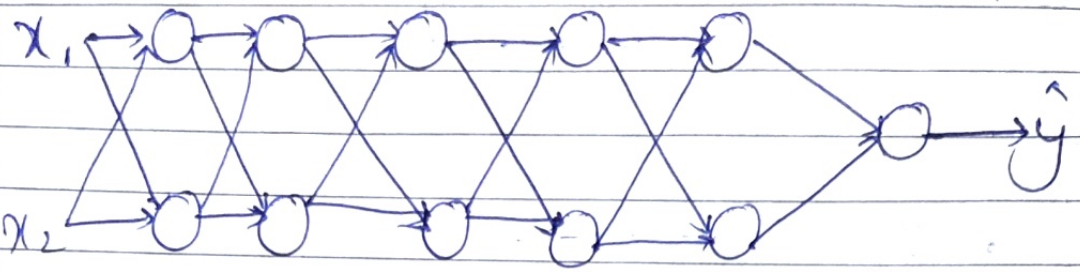
If we don't normalise:



If we normalise



Video 2: Vanishing / Exploding Gradients



$w^{[1]}$ $w^{[2]}$ $w^{[3]}$ . . . . . $w^{[l]}$

$g(z) = z$ $\quad b^{[l]} = 0$

$\hat{y} = w^{[l]} w^{[l-1]} w^{[l-2]} \ldots w^{[3]} w^{[2]} w^{[1]} x$

$z^{[1]} = w^{[1]} x$

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

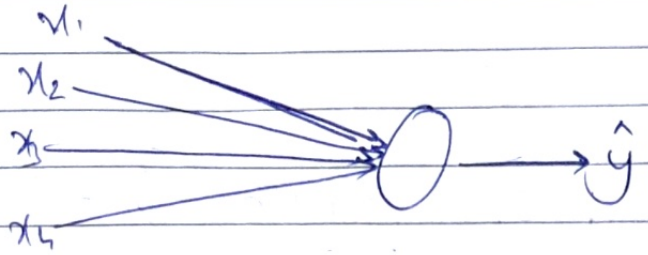
$a^{[2]} = g(z^{[2]}) = g(w^{[2]} a^{[1]})$

$w^{[l]} > I \rightarrow$ activations can explode

$w^{[l]} < I \rightarrow$ activations decrease exponentially

Page No.
Date

Video 3: Weight Initialisation for Deep Networks



$$a = g(z)$$

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

large n → smaller w_i

$$Var(w) = \frac{1}{n} \quad \text{or} \quad \frac{2}{n}$$

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

ReLU

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

Video 4: Numerical Approximation of Gradients

Video 5: Gradient Checking

* Gradient checking approximates the gradients and is used to find errors in backpropagation but is slower than gradient descent

* Take $W^{[1]}, b^{[1]}, \ldots, W^{[l]}, b^{[l]}$ → reshape to a vector of size(θ)

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

Page No.
Date

Take $dw^{[1]}$ $db^{[1]}$, ..., $dw^{[l]}$, $db^{[l]}$ & reshape to a vector $d\theta$.

For each i :

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

$$\approx d\theta[i] = \frac{\partial J}{\partial \theta_i} \quad | \quad d\theta_{approx} \approx d\theta$$

Check:

$$\frac{\|d\theta_{approx} - d\theta\|_2}{\|d\theta_{approx}\|_2 + \|d\theta\|_2} \quad \begin{matrix} \approx 10^{-7} : \text{great} \\ : \\ : \\ : \text{worry} \end{matrix}$$

$\varepsilon = 10^{-7}$

Video 5: Gradient Checking Implementation Notes

- Don't use in training - only to debug [really slow process]
- If algorithm fails grad check, look at components to identify bug : look at the different values of i to see which are the values of $d\theta$ approx really different from $d\theta$
- Regularisation

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

- Doesn't work with dropout
- run at random initialisation ; perhaps again after training.