Machine learning specialization

Course 1

WEEK 1

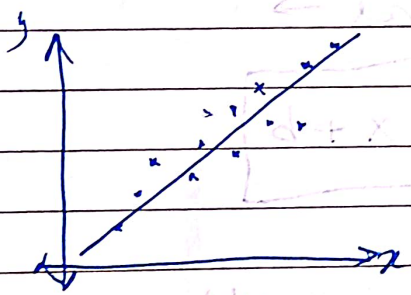
* Supervised Machine Learning

algos that go from input to output

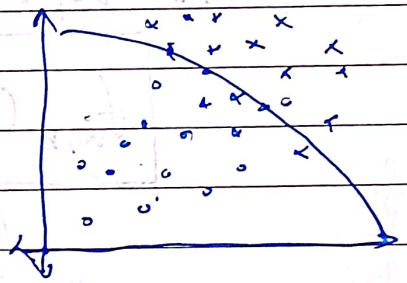
$x \rightarrow y$

→ Regression: predict a number, infinite outputs
→ Classification: output is limited, predict class

Regression



Classification



* Unsupervised

→ find something in unlabeled data

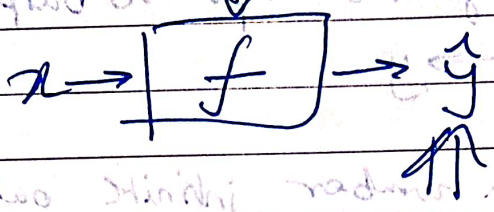
y is not given in learning data.

(i) clustering
(ii) anomaly detection
(iii) Dimensionality reduction

training set

↓

learning algorithm



It is the models prediction

f is represented as →

$$f_{w,b}(x) = wx + b$$

linear regression with one variable

Now we know how to draw the line, now we need to minimize error.

i.e. get $\hat{y}^{(i)}$ closer to $y^{(i)}$ for all $(x^{(i)}, y^{(i)})$

J is the cost function.

J gives the cost of a line

Cost function: Squared error cost function

$$J_{(w,b)} = \frac{1}{2m} \sum_{i=1}^{m} \left(\hat{y}^{(i)} - y^{(i)} \right)^2$$

2024/10/17 12:

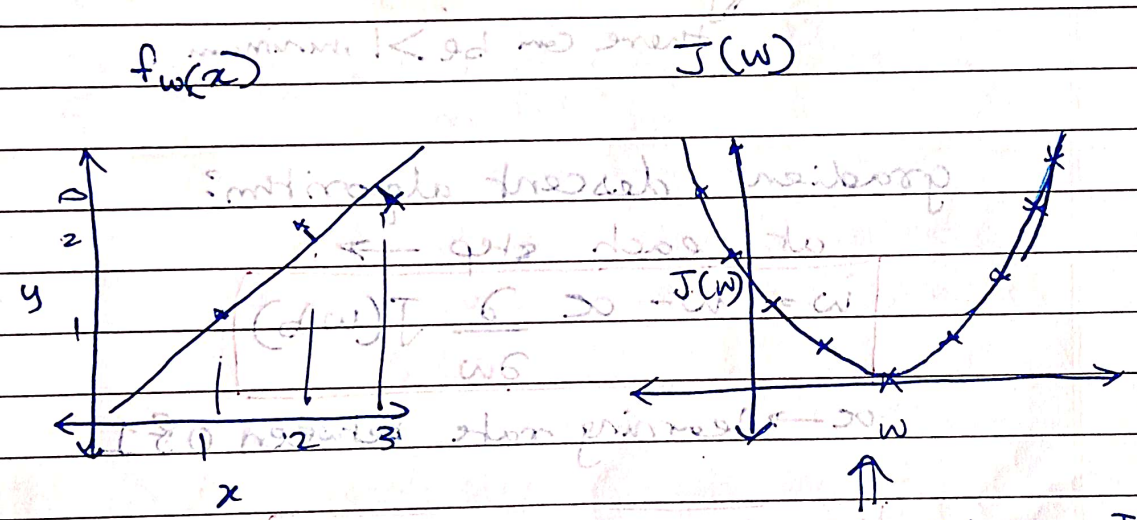
m = no. of training examples

$\hat{y}^{(i)} - y^{(i)}$ → error

then we square it and sum for all the for the many we divide by 2m
2 for neater calcs.

can also be written as:

$$J(w,b) = \frac{\sum_{i=1}^{m} \left(f_{w,b}(x^{(i)}) - y^{(i)}\right)^2}{2m}$$



INTUITION

Model: $f_{w,b}(x) = wx + b$

Parameters: w, b

Cost function: $J(w,b) = \frac{1}{2m} \sum_{i=1}^{m} \left(f_{w,b}(x^{(i)}) - y^{(i)}\right)^2$

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

* Gradient Descent

→ U can apply it to any function, to minimize that function.

Outline:

Start with some (w, b)
keep changing w, b to reduce $J(w,b)$
Until we settle at or near a minimum.

⇓ there can be > 1 minimum

gradient descent algorithm:

at each step →

$$w = w - \alpha \frac{\partial}{\partial w} J(w,b)$$

α → learning rate between 0 & 1

$$b = b - \alpha \frac{\partial}{\partial b} J(w,b)$$

keep updating w & b simultaneously
until w & b don't change much.

correct implementation in code:

$$temp\_w = w - \alpha \frac{\partial}{\partial w} J(w,b)$$

$$temp\_b = b - \alpha \frac{\partial}{\partial b} J(w,b)$$

$w = temp\_w$
$b = temp\_b$

2024/10/17 12:34

$$\frac{\partial}{\partial w} J(w,b) = \frac{\partial}{\partial w} \frac{1}{2m} \sum_{i=1}^{m} \left(f_{w,b}(x^{(i)}) - y^{(i)}\right)^2$$

$$= \frac{\partial}{\partial w} \frac{1}{2m} \sum_{i=1}^{m} \left(wx^{(i)} + b - y^{(i)}\right)^2$$

$$= \frac{1}{2m} \sum_{i=1}^{m} \left(wx^{(i)} + b - y^{(i)}\right) 2x^{(i)}$$

$$\frac{\partial}{\partial w} J(w,b) = \frac{1}{m} \sum_{i=1}^{m} \left(f_{w,b}(x^{(i)}) - y^{(i)}\right) x^{(i)}$$

$$\frac{\partial}{\partial b} J(w,b) = \frac{1}{m} \sum_{i=1}^{m} \left(f_{w,b}(x^{(i)}) - y^{(i)}\right)$$

notice how after derivative, the square over the bracket is gone.

At each step we can say,

$$w = w - \alpha \cdot \frac{1}{m} \sum_{i=1}^{m} \left(f_{w,b}(x^{(i)}) - y^{(i)}\right) x^{(i)}$$

$$b = b - \alpha \cdot \frac{1}{m} \sum_{i=1}^{m} \left(f_{w,b}(x^{(i)}) - y^{(i)}\right)$$

This gradient descent is also called "batch" gradient descent. At each step of gradient descent makes use of all training examples.

Date
Page

Week2

* Linear Regression with multiple variables.

col1	col2	col3	col4	output
x_1	x_2	x_3	x_4	y

x_j = j^{th} feature

n = no. of features

$\vec{x}^{(i)}$ = features of i^{th} example

$\vec{x}^{(2)} = [14 \quad 16 \quad 3 \quad 2 \quad 40]$

This is a vector

$x_j^{(i)}$ = value of feature j of i^{th} example.

$x_3^{(2)} = 2$

Scalar

Previously: $f_{w,b}(x) = wx + b$

Now:

$$f_{\vec{w},b}(\vec{x}) = w_1 x_1 + w_2 x_2 + \dots + w_n x_n + b$$

$\vec{w} = [w_1 \quad w_2 \quad w_3 \quad \dots \quad w_n]$

$\vec{x} = [x_1 \quad x_2 \quad x_3 \quad \dots \quad x_n]$

b is a number

$$f_{\vec{w},b}(\vec{x}) = \vec{w} \cdot \vec{x} + b$$

dot product of vectors

2024/10/17

Code implementation of f

Let $\vec{w} = [1.0 \quad 2.5 \quad -3.3]$

$b = 4$

$\vec{x} = [10 \quad 20 \quad 30]$

we can initialize these matrices or vectors in Python using NumPy as:

```
w = np.array([1.0, 2.5, -3.3])
b = 4
x = np.array([10, 20, 30])
```

Now, $f_{\vec{w},b}(\vec{x}) = w_1x_1 + w_2x_2 + w_3x_3 + b$

So WITHOUT VECTORISATION Python code to find out f:

```
f = w[0] * x[0] +
    w[1] * x[1] +
    w[2] * x[2] + b
```

write new line for all values

or if $f_{\vec{w},b}(\vec{x}) = \left(\sum_{j=1}^{n} w_j x_j\right) + b$

WITHOUT VECTORISATION:

```
f = 0
for j in range(0, n):
    f = f + w[j] * x[j]
f = f + b
```

really slow

Now using Vectorisation,

$$f_{\vec{w},b}(\vec{x}) = \vec{w} \cdot \vec{x} + b$$

One line + much faster

Code:

```
f = np.dot(w, x) + b
```

np.dot() uses vectorisation in the computer's hardware to run multiple calculations in parallel. Thus, significantly reducing compute time.

* Gradient Descent (implementation)

$\vec{w} = (w_1\ w_2\ w_3 \dots w_{16})$

$\vec{d} = (d_1\ d_2\ d_3 \dots d_{16})$ ← derivatives

```
w = np.array([0.5, 1.3, ..., 3.4])
d = np.array([0.3, 0.2, ..., 0.4])
```

compute $w_j = w_j - 0.1\, d_j$ for $j = 1 \dots 16$

(0.1 → learning rate α)

W10 vectorisation:

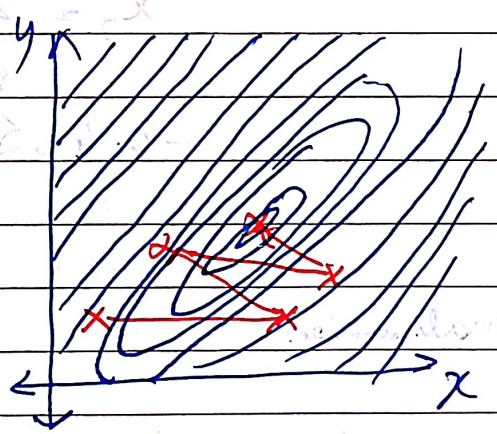
```
for j in range(0,16):
    w[j] = w[j] - 0.1 * d[j]
```

With vectorisation:

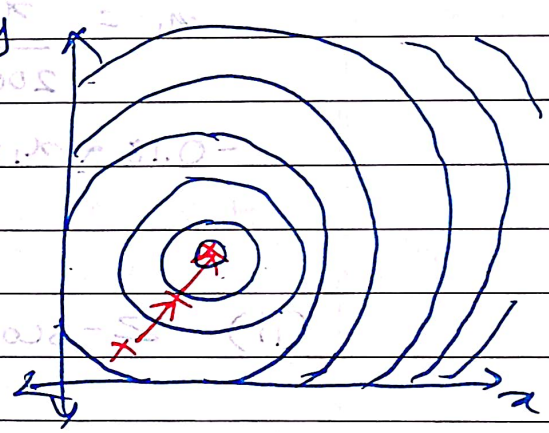
" $W = W - 0.1 * d$ " ← executes for all values of w and d parallely

Feature Scaling:

If difference betⁿ parameters is ~~two~~ too high, gradient descent could take a lot of time



skinny contour plot
Inefficient for GD



round Contour plot
Efficient for GD

hence we normalize incoming inputs betⁿ 0 to 1 so that its more efficient.

Feature scaling

for eg) $300 \leq x_1 \leq 2000$

$x_1 = \frac{x_1}{2000}$

$0.15 \leq x \leq 1$

$0 \leq x_2 \leq 5$

$x_2 = \frac{x_2}{5}$

$0 \leq x_2 \leq 1$

We can also feature scale using other methods.

(i) Mean normalisation:

$300 \leq x_1 \leq 2000$ $\qquad$ $0 \leq x_2 \leq 5$

μ_1 = average $\qquad$ $\mu_2 = 2.3$ (assume)

≈ 600 (assume)

$x_1 = \frac{x_1 - \mu_1}{2000 - 300}$ $\qquad$ $x_2 = \frac{x_2 - \mu_2}{5 - 0}$

$-0.18 \leq x_1 \leq 0.82$ $\qquad$ $-0.46 \leq x_2 \leq 0.54$

(ii) Z-score normalisation:

σ: standard deviation

σ_1: SD of 1st $\qquad$ σ_2: SD of 2nd

$x_1 = \frac{x_1 - \mu_1}{\sigma_1}$ $\qquad$ $x_2 = \frac{x_2 - \mu_2}{\sigma_2}$

$-0.67 \leq x \leq 3.1$ $\qquad$ $-1.6 \leq x_2 \leq 1.9$

$1 \geq x \geq 0$ $\qquad$ $1 \geq x \geq -1.0$

In feature scaling, aim for about

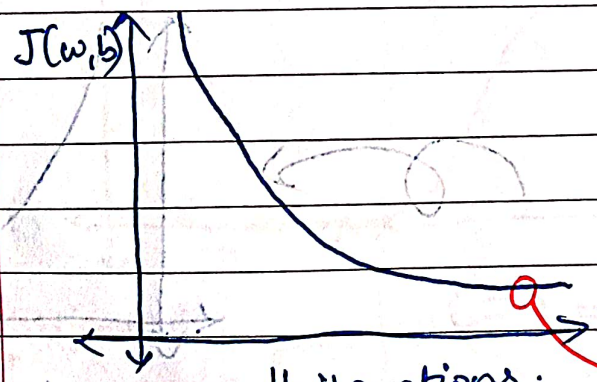
$-1 \leq x_j \leq 1$
$-3 \leq x_j \leq 3$
$-0.3 \leq x_j \leq 0.3$
} acceptable ranges.

$0 \leq x_1 \leq 3$
$-2 \leq x_2 \leq 0.5$
} okay, no rescaling

$-100 \leq x_3 \leq 100$ → too large, rescale

$-0.0001 \leq x_4 \leq 0.0001$ → too small, rescale.

→ To make sure GD is working correctly plot a graph of cost function $J(w,b)$ after each iteration.



← This called a learning curve

J has converged around here since its not longer decreasing.

→ Automatic convergence test

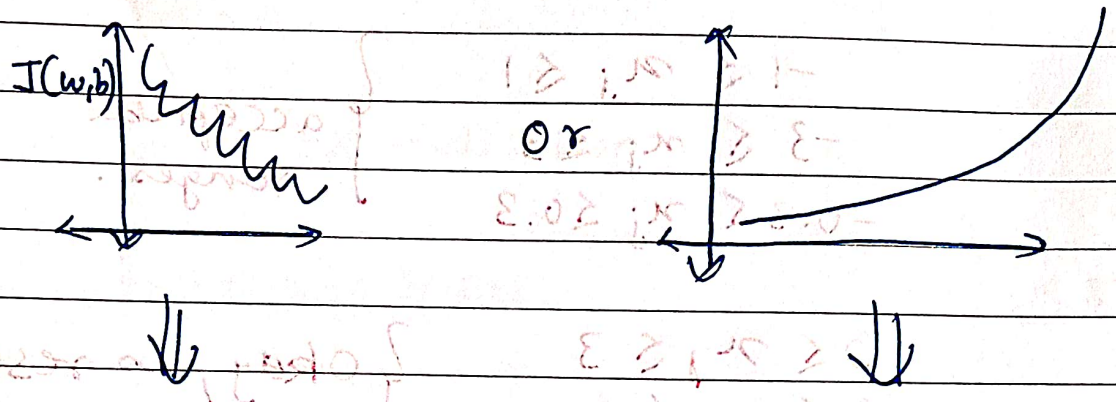
let $\varepsilon = 0.001$

if $J(w,b)$ decreased by $\leq \varepsilon$

then J has converged.

2024/10/17 12:35

selecting α

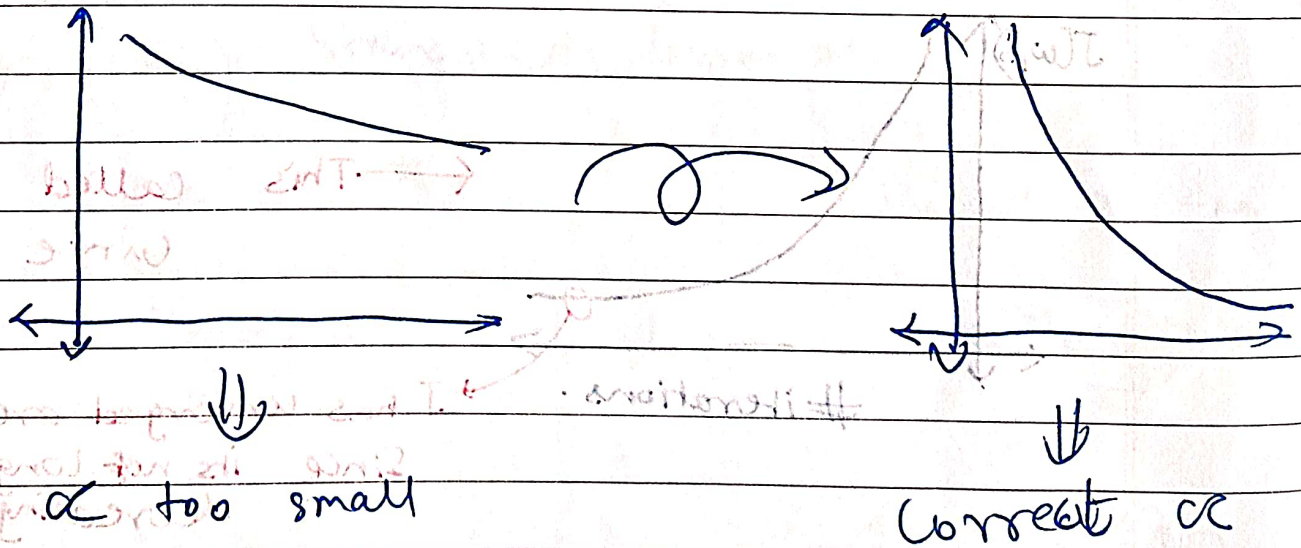


→ either
bug or
→ α is
too large

→ either
$w_1 = w_1 + \alpha d_1$
use minus sign
$w_1 = w_1 - \alpha d_1$
→ or α is too large

Ideal values of α to try

.... 0.001 ..... 0.1 ...



α too small

Correct α

Date
Page

WEEK 3

* Classification

→ limited output options for "y"

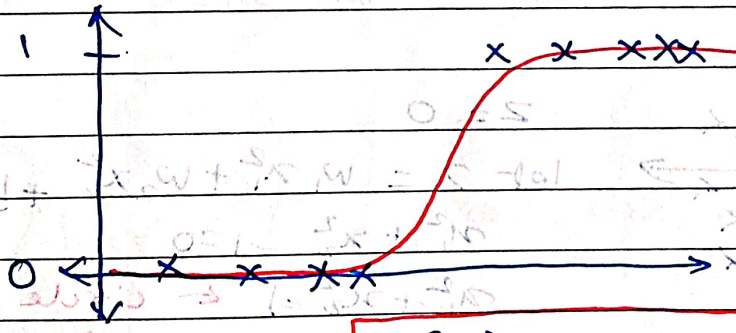
binary classification: y can only be one of 2 values

→ Binary classification problems are solved using Logistic regression.

→ although it is named regression it is still a classification algo.

in logistic regr. output is either 0 or 1

→ linear main line fit karte hai
logistic main S-shaped curve



← sigmoid function OR Logistic function

Sigmoid fn formula →

$$g(z) = \frac{1}{1+e^{-z}} \quad ; \quad 0 < g(z) < 1$$

Let $z = \vec{w} \cdot \vec{x} + b$ → linear regr. formula

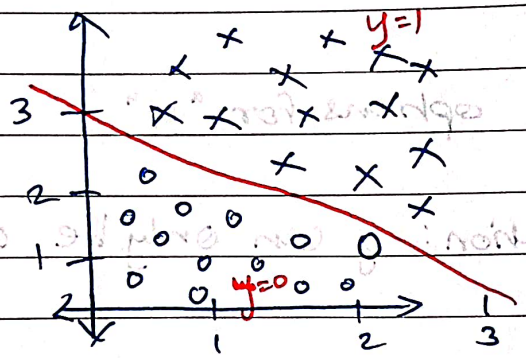
$$f_{\vec{w},b}(\vec{x}) = g(z) = g(\vec{w} \cdot \vec{x} + b) = \frac{1}{1+e^{-(\vec{w} \cdot \vec{x} + b)}}$$

↑ Logistic regr. formula

→ Its output is the probability that class is 1.

2024/10/17 12:35

Decision Boundary



→ It is the boundary where the result $z = \vec{w} \cdot \vec{x} + b = 0$

→ which means the result is ~~zero~~ neutral. i.e y is neither 0 or 1.

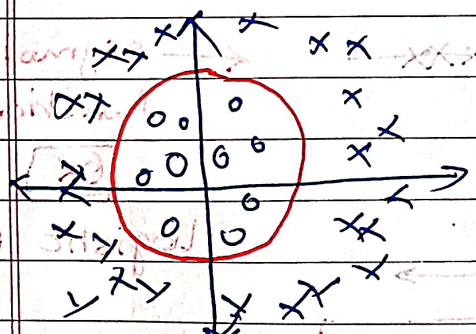
$z = 0$ at decision boundary.

$z = \vec{w} \cdot \vec{x} + b = 0$

$\vec{w} \cdot \vec{x} + b = 0$

$x_1 + x_2 - 3 = 0$

$x_1 + x_2 = 3$

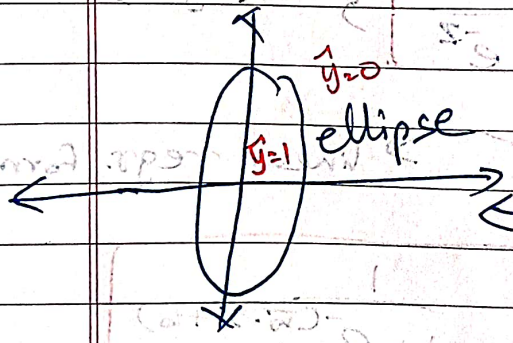


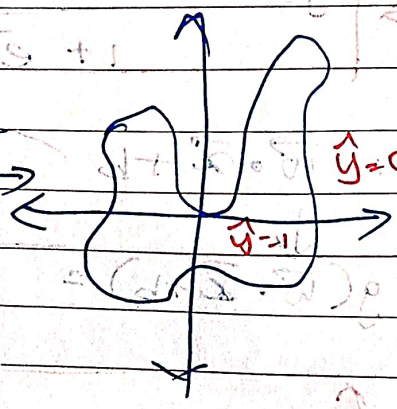
$z = 0$

let $z = w_1 x_1^2 + w_2 x_2^2 + b$

$x_1^2 + x_2^2 - 1 = 0$

$x_1^2 + x_2^2 = 1$ ← circle of radius 1





more complex boundaries can be formed if higher order polynomials are used.

2024/10/17 12

Cost function for logistic regression

→ The square error cost function does not work

loss of linear

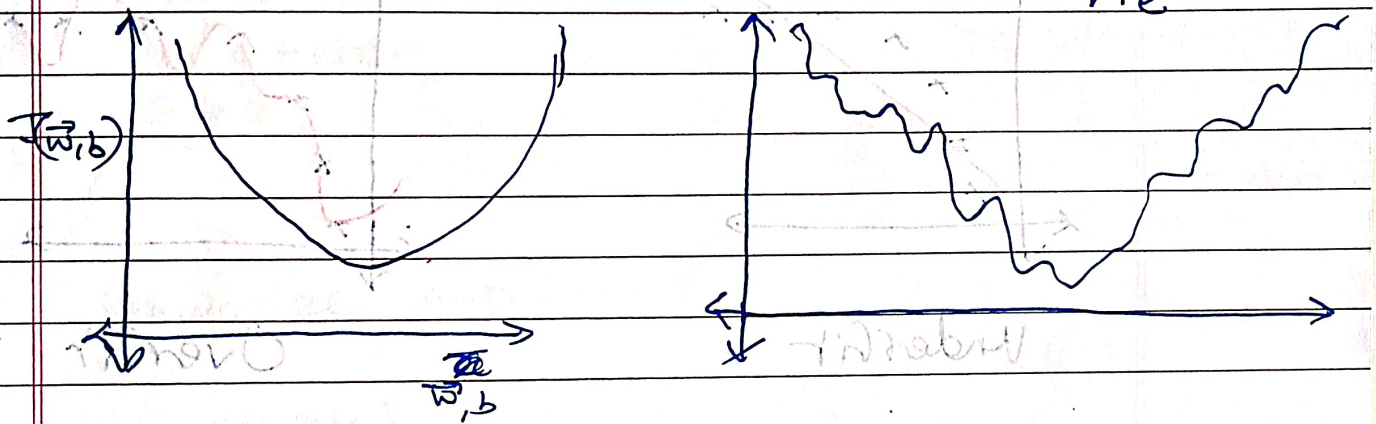
$$J(w,b) = \frac{1}{m}\sum_{i=1}^{m} \boxed{\frac{1}{2}\left(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)}\right)^2}$$

linear regression

$$f_{\vec{w},b}(\vec{x}) = \vec{w}\cdot\vec{x} + b$$

logistic regression

$$f_{\vec{w},b}(\vec{x}) = \frac{1}{1+e^{-(\vec{w}\cdot\vec{x}+b)}}$$



Only one minima always

multiple local minima.

∴ using the same cost f^n is not ideal.

→ loss: $L\left(f_{\vec{w},b}(\vec{x}^{(i)}), y^{(i)}\right) = \frac{1}{2}\left(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)}\right)^2$

↑ loss f^n for linear regr.

logistic loss function:

$$L\left(f_{\vec{w},b}(\vec{x}^{(i)}), y^{(i)}\right) = \begin{cases} -\log\left(f_{\vec{w},b}(\vec{x}^{(i)})\right) & \text{if } y^{(i)}=1 \\ -\log\left(1 - f_{\vec{w},b}(\vec{x}^{(i)})\right) & \text{if } y^{(i)}=0 \end{cases}$$

converts to convex.

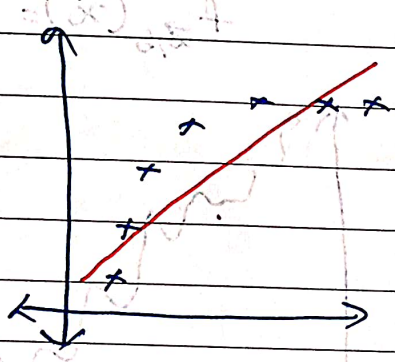
y can only be 1 or 0 in binary classification.

plug this loss f^n in $J(w,b) = \frac{1}{m}\sum L\left(f_{\vec{w},b}(\vec{x}^{(i)}), y^{(i)}\right)$

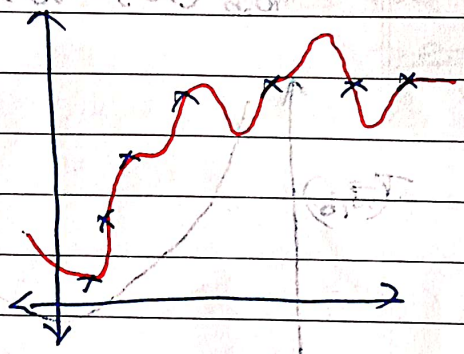
2024/10/17 12:35

→ Overfitting: When you fit the training set so well that the model gets worse at predicting values for new data

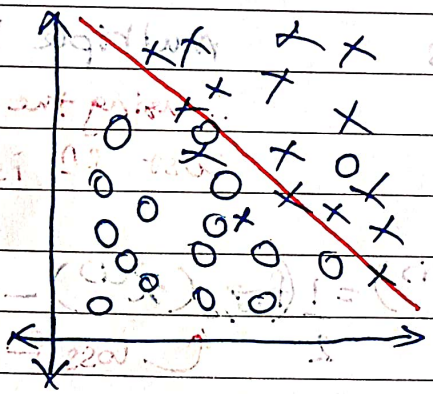
→ Underfit: When the model uses less features to fit data and results to high error

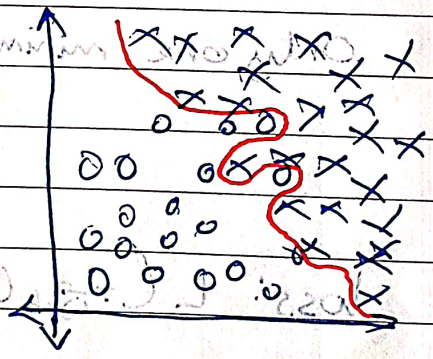


Underfit



Overfit





To Address Overfitting:

1) Collect more data
2) Select only a subset of features
   - feature selection
3) Reduce size of parameters
   - Regularization.

* Regularization cost fn

$$J(\vec{w}, b) = \frac{1}{2m} \sum_{i=1}^{m} \left(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)}\right)^2 + \boxed{\frac{\lambda}{2m} \sum_{j=1}^{n} w_j^2} + \frac{\lambda}{2m} b^2$$

- mean squared error
- no. of features
- regularisation term
- λ: regularisation parameter, $\lambda > 0$
- can include or exclude b bcus it makes very less diff.

* Regularized GD

repeat {

$$w_j = w_j - \alpha \left[\frac{1}{m} \sum_{i=1}^{m} \left[\left(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)}\right) x_j^{(i)}\right] + \frac{\lambda}{m} w_j\right]$$

$$b = b - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)}\right)$$

} simultaneous update.

- we don't add anything bcus we aren't changing how b is updated

Date
Page

*Regularized cost fn for logistic regr.

$$J(w,b) = \frac{-1}{m}\sum_{i=1}^{m}\left[y^{(i)}\log\left(f_{\vec{w},b}(\vec{x}^{(i)})\right) + (1-y^{(i)})\log\left(1-f_{\vec{w},b}(\vec{x}^{(i)})\right)\right] + \underbrace{\frac{\lambda}{2m}\sum_{j=1}^{n} w_j^2}$$

→ Regularized GD for logistic regr is the same for linear regr

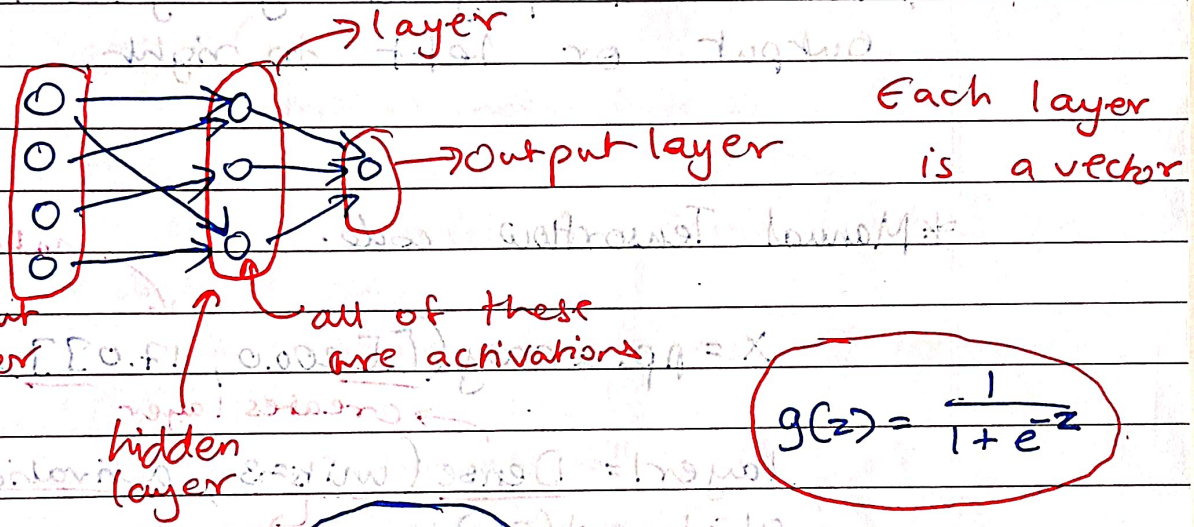
Course 2 week 1

Advanced Learning Algorithms

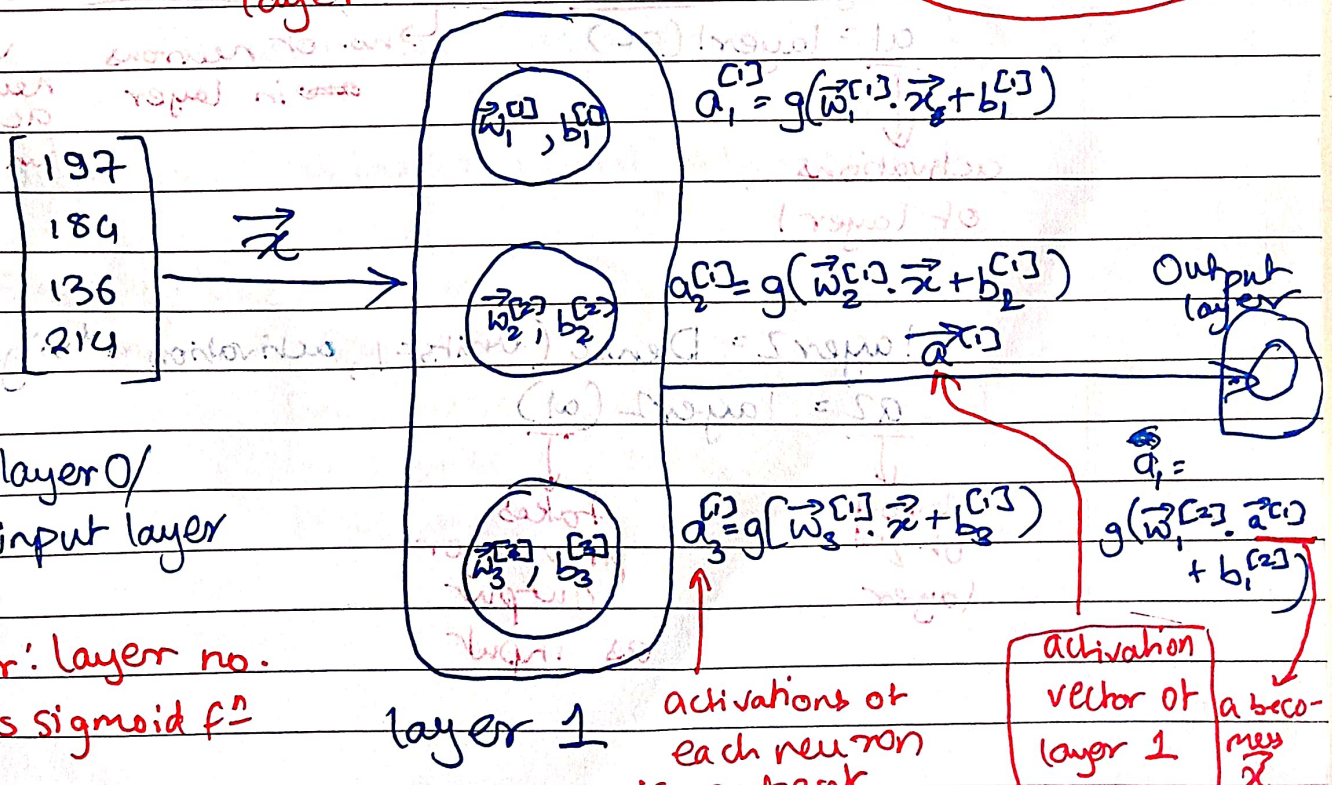
a neuron ⇒ ○

input → ○ → probability

all neurons use a function for output so neurons are just a bunch of logistic regression models bundled together



$$g(z) = \frac{1}{1+e^{-z}}$$



[ ] super: layer no.

g(z) is sigmoid fn

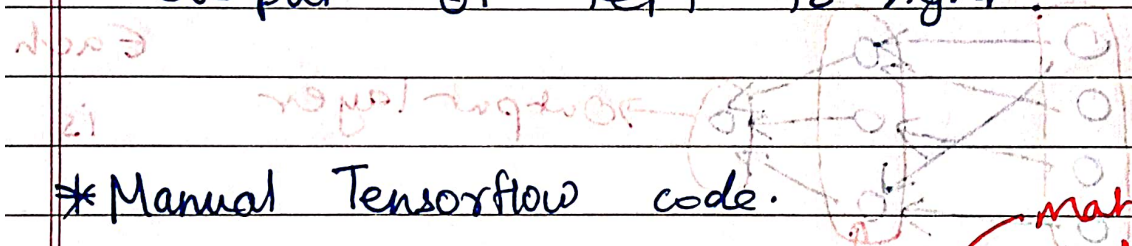
General Notation of one neuron's activation

$$a_j^{[l]} = g(w_j^{[l]} \cdot \vec{a}^{[l-1]} + b_j^{[l]})$$

j^{th} neuron of l^{th} layer

$g()$: "activation function"

Forward Propagation: going from input to output or left to right.



* Manual Tensorflow code.

matrix of 1 row

```
X = np.array([[200.0, 17.0]])

layer1 = Dense(units=3, activation="sigmoid")
a1 = layer1(x)

layer2 = Dense(units=1, activation="sigmoid")
a2 = layer2(a1)
```

creates layer

units = no. of neurons in layer

activation = neuron activation function

a1 = activations of layer 1

a2 = Output of 2nd layer

(a1) takes prev layer output as input

Date
Page

Course 2 week 2

* Auto tensorflow code

```
layer_1 = Dense(units=3, activation="sigmoid")
layer_2 = Dense(units=1, activation="sigmoid")
model = Sequential([layer_1, layer_2])
```

↳ joins layers automatically.

```
x = np.array([[200.0, 17.0]
              :
              :
              [212.0, 18.0]])
y = np.array([1,0,0,1])
model.compile(...)
```

↳ compiles model layers

```
model.fit(x,y)
```

↳ train model on given inputs & o/ps

```
model.predict(x_new)
```

↳ predict o/p of new x input.

```
model = Sequential([
    Dense(units=3, activation="sigmoid"),
    Dense(units=1, activation="sigmoid")])
```

* Steps in training a Neural Network model

Step 1: Make a model

```
model = Sequential([
    Dense(units=  , activation= )
    Dense(units=  , activation= )
    Dense(units=  , activation= )])
```

Step 2: Compile / specify loss & cost

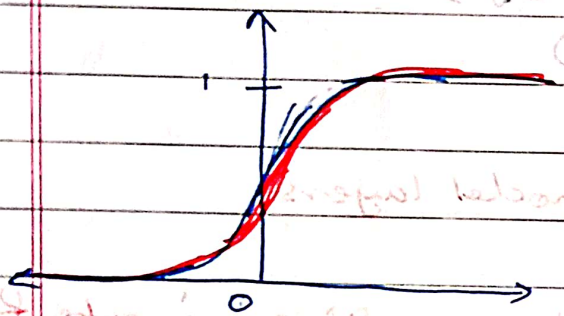
Binary cross entropy: the same loss as in binary classification

```
model.compile(loss=BinaryCrossentropy(); MeanSquaredError())
```

Step 3: fit model / minimize cost

```
model.fit(x, y, epochs=)
```

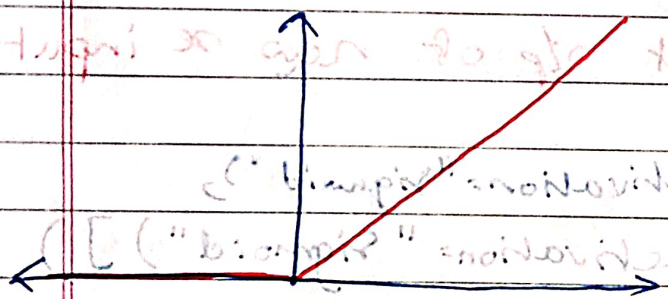
* Types of activation functions



Sigmoid:

$$g(z) = \frac{1}{1+e^{-z}}$$

range: 0 to 1

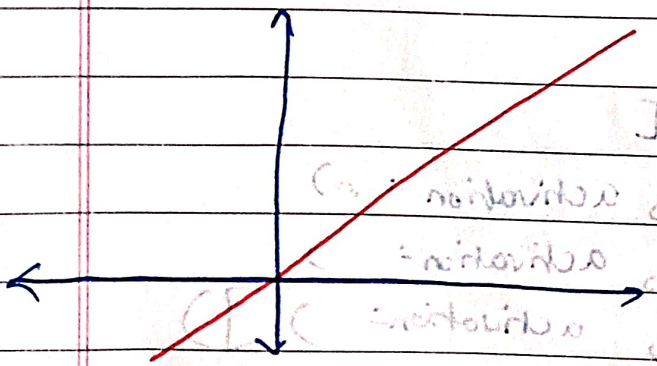


ReLU:
(rectified linear unit)

$g(z) = \max(0, z)$

if $z<0$ $\quad$ if $z>0$

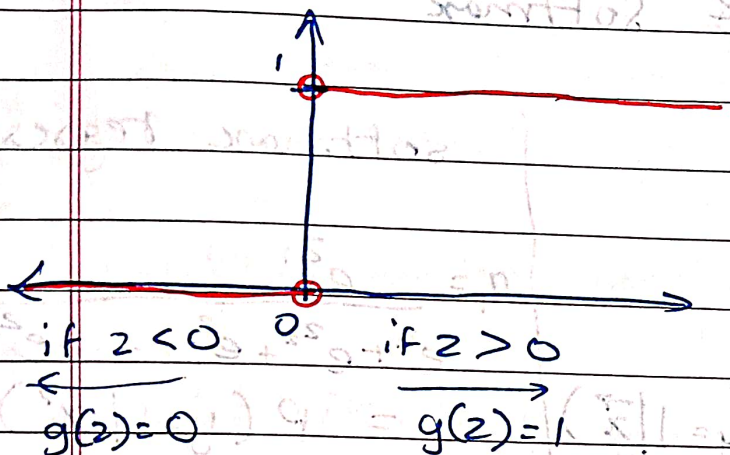
$g(z)=0$ $\quad$ $g(z)=z$



linear activation fn:

$g(z) = z$

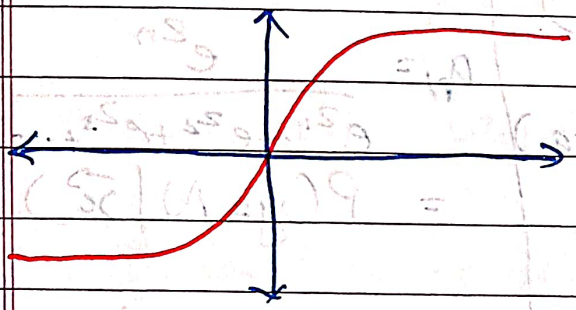
$g(z) = \vec{w} \cdot \vec{x} + b$



Binary step fn:

range: 0 to 1

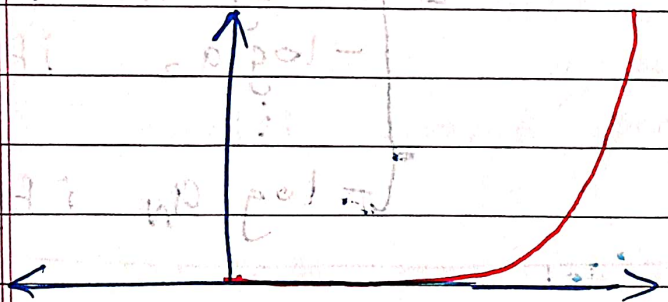
$$g(z) = \begin{cases} 0 & \text{if } z < 0 \\ 1 & \text{if } z \geq 0 \end{cases}$$



tanh (hyperbolic tangent)

range: -1 to 1

$$g(z) = \frac{2}{1+e^{-2x}} - 1$$



Softmax:

range: 0 to 1

$$g(z) = \frac{e^z}{\sum_{i=1} e^{z_i}}$$

Choosing the right fn

(i) Regression: Linear activation
(ii) Binary classification: Sigmoid
(iii) Multiclass classification: Softmax
(iv) Multilabel classification: sigmoid.
(v) CNN: ReLU
(vi) RNN: Tanh or sigmoid

* Cost of logistic & Softmax

logistic regression	softmax regression
$z = \vec{w} \cdot \vec{x} + b$	$a_1 = \frac{e^{z_1}}{e^{z_1} + e^{z_2} + e^{z_3} + \cdots + e^{z_N}}$
$a_1 = g(z) = \frac{1}{1+e^{-z}} = P(y=1 \mid \vec{x})$	$= P(y=1 \mid \vec{x})$
$a_2 = 1 - a_1 = P(y=0 \mid \vec{x})$	$\vdots$
	$a_N = \frac{e^{z_N}}{e^{z_1} + e^{z_2} + e^{z_3} + \cdots + e^{z_N}}$
	$= P(y=N \mid \vec{x})$
$loss = -y \log a_1 - (1-y) \log(1-a_1)$	
$= -y \log a_1 - (1-y) \log a_2$	
$J(\vec{w}, b)$ = average loss	$loss(a_1, \cdots a_N, y) = \begin{cases} -\log a_1 & \text{if } y=1 \\ -\log a_2 & \text{if } y=2 \\ \vdots & \\ -\log a_N & \text{if } y=N \end{cases}$

* MNIST with softmax regression

Step 1 specify model

```
import tensorflow as tf
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense
model = Sequential([
    Dense(units=25, activation='relu'),
    Dense(units=10, activation='relu'),
    Dense(units=10, activation='softmax')])
```

Step 2

Specify loss & cost

```
from tensorflow.keras.losses import SparseCategoricalCrossentropy
model.compile(loss=SparseCategoricalCrossentropy())
```

Steps train

```
model.fit(X, Y, epochs=100)
```

* Improved Implementation of softmax

make last layer linear instead of softmax

```
Dense(units=10, activation='linear')
```

→ use from_logits

```
model.compile(loss=SparseCategoricalCrossentropy(
                    from_logits=True))
```

→ predict not using predict()

```
logits = model(X)
f_x = tf.nn.softmax(logits)
```

* Adam algorithm

→ Adam is an adaptive learning rate algorithm designed to improve training speeds in deep neural networks and reach convergence quickly.

→ Its an alternative to gradient descent and works faster than gradient descent.

→ If w_j (or b) keeps moving in the same direction, increase α_j

→ If w_j (or b) keeps oscillating, reduce α_j

→ MNIST Adam to compile

```
model.compile(
    optimizer= tf.keras.optimizers.Adam(
        learning_rate = 1e-3),
    loss= tf.keras.losses.SparseCategoricalCrossentropy(
        from_logits= True ))
```

Date
Page

* Additional layer types

(i) Convolutional Layer

→ Each neuron only looks at part of the previous layer's outputs.

Why?

→ faster computation

→ Need less training data
(Less prone to overfitting)

→ A neural network made using these layers is called Convolutional Neural Network (CNN)

Course 2 week 3

Deciding what to try Next

Options

high variance → Get more training examples
high variance → Try smaller set of features
high bias → Try getting additional features
high bias → Try adding polynomial features
high bias → Try decreasing λ
high variance → Try increasing λ
→ anything else?

Selecting what to do to debug ur model is important

(i) If $J_{train}(\vec{w}, b)$ is low but $J_{test}(\vec{w}, b)$ is high,
Then the model is overfit.

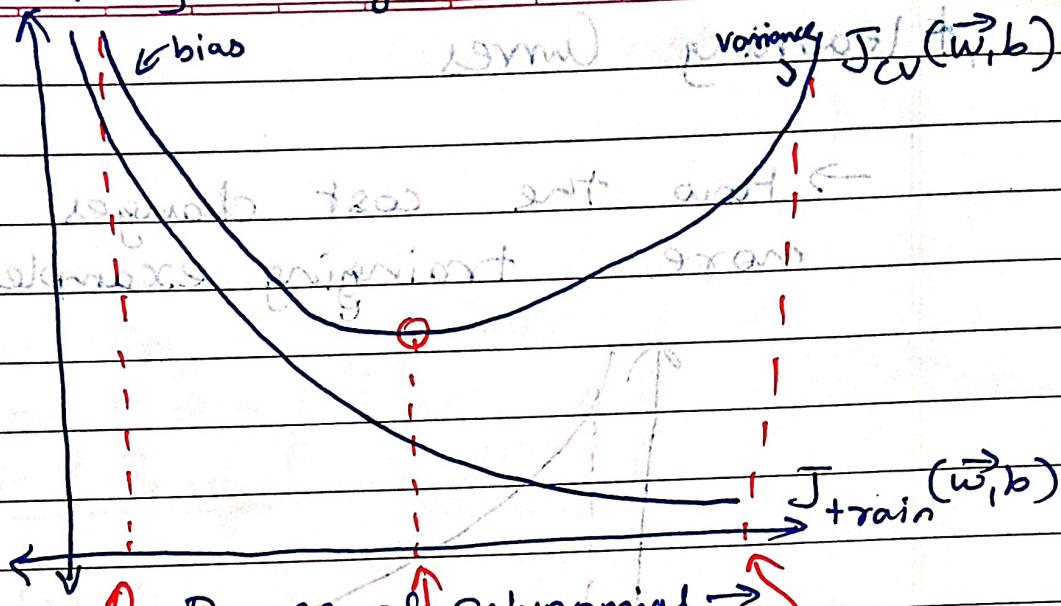
→ Train, test, valid are 3 sets to divide our data into.

first train on different models and compare with valid.
then pick model with least valid cost error.
then use test to check performance

If the J_{train} & J_{cv} is low, then the model is right.

Date
Page

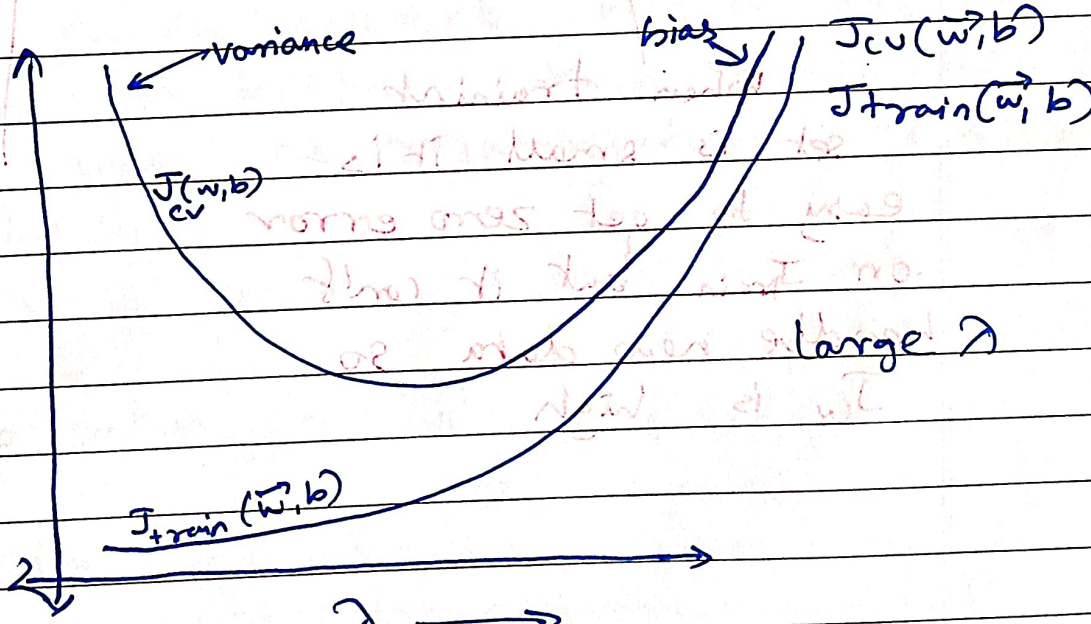
Selecting degree of polynomial



both high
(underfit)
high bias

both low
(just right)

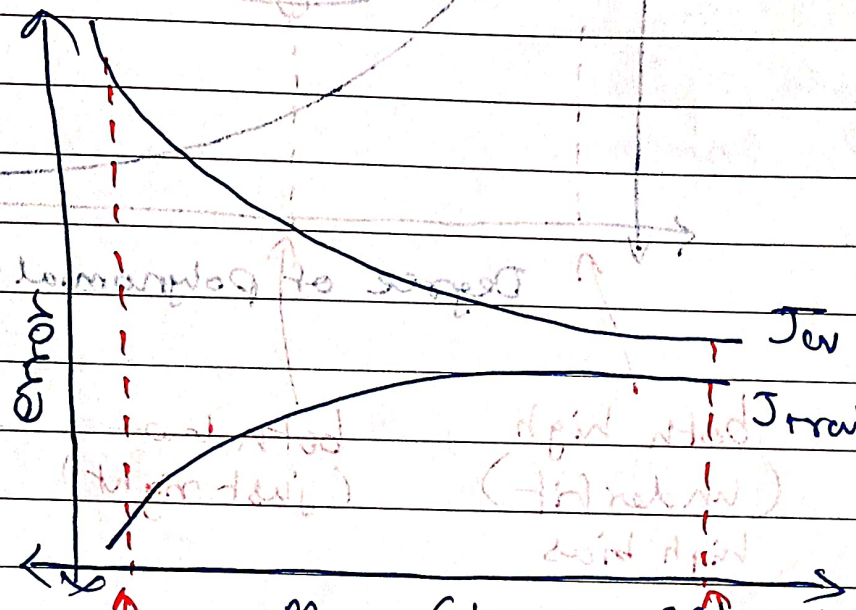
J_{cv} high
but J_{train}
low
(overfit)
high variance



Date
Page

* learning Curves

→ How the cost changes as we add more trainging examples.

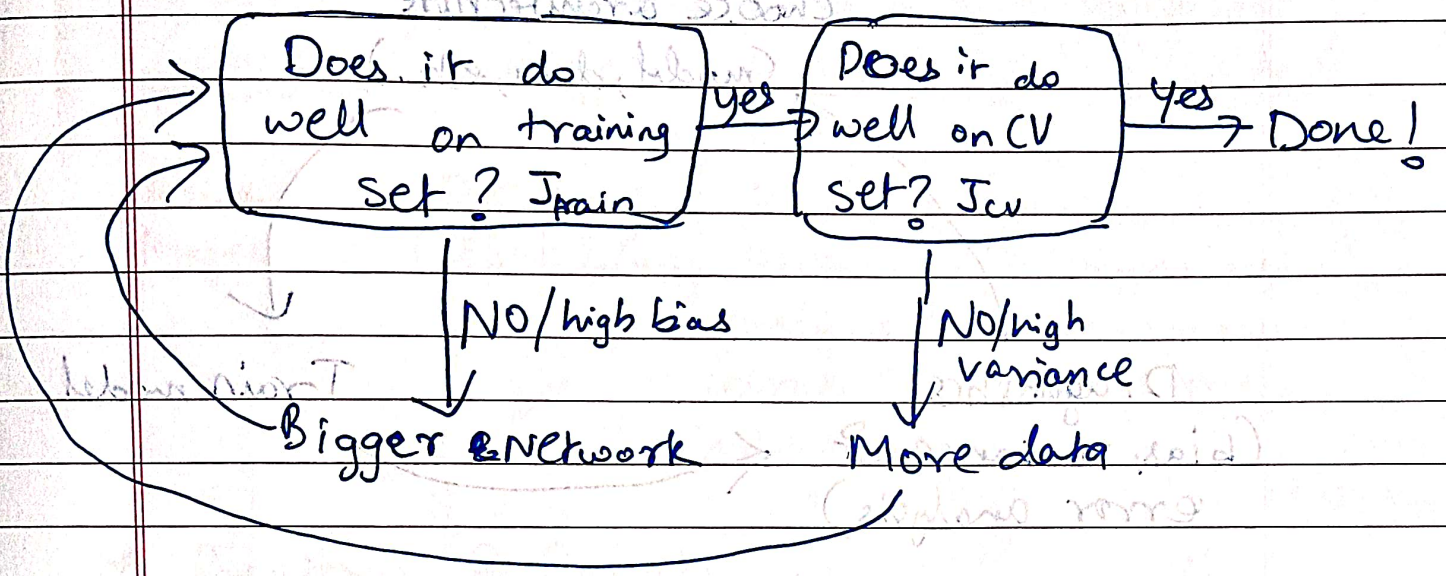


When trainint set is small it is easy to get zero error on J_{train} but it can't handle new data so J_{cv} is high

When it is more both come closer

Date
Page

* Bias & variance in Neural networks *

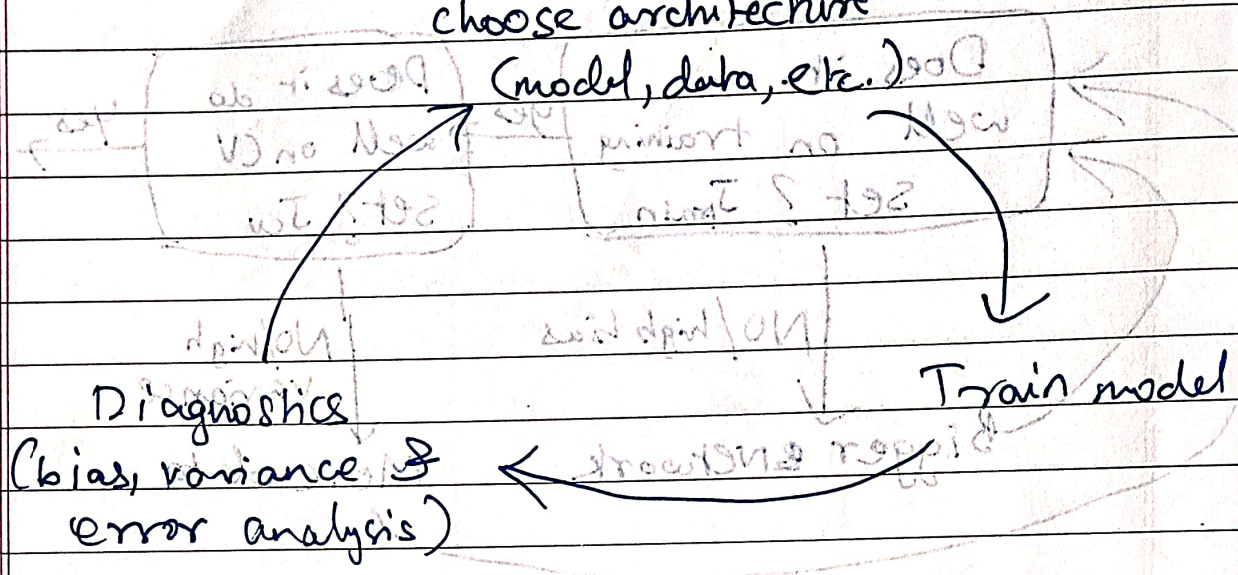


→ A large neural network will usually do as well or better than a smaller one so long as regularisation is chosen appropriately.

⇒ Regularisation in NN's

```
model = Sequential([
    Dense(units=, activation=, kernel_
        regularisation=
        L2(0.01),
    Dense(U=, a=, kernal_regularisation= L2(0.01),
    Dense(U=, a=, 4)])
```

* Iterative loop of machine learning:



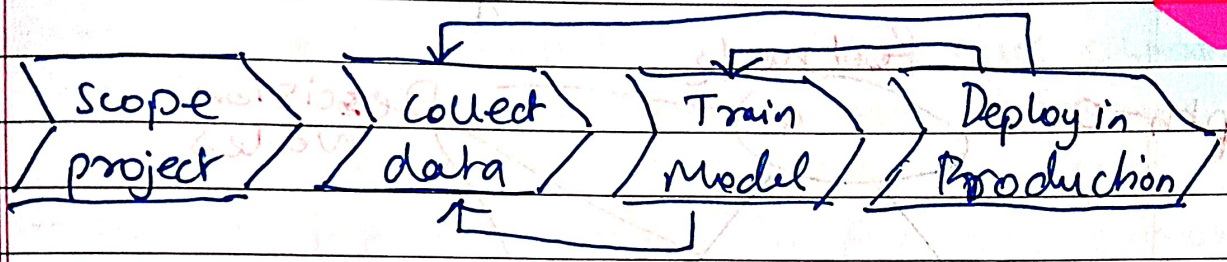
Data Augmentation: modifying an existing training example to create a new training example.

* Transfer Learning

→ Downloading neural network parameters pretrained on a large dataset with same input type as your own application.

→ further train the network on your own data.

* All steps of a machine learning project.



Define Project

Define & Collect data

Training, error analysis & iterative improvement

Deploy, monitor & maintain system

* Precision & Recall.

y=1 in presence of rare class we want to detect.

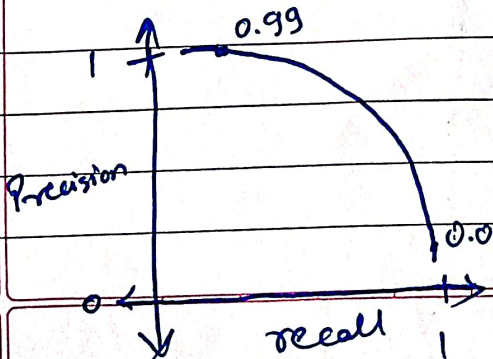
Predicted class	actual class 1	actual class 0
1	True positive	False +ve
0	False -ve	True neg-ve

Precision

$$\frac{\text{True +ves}}{\#\text{pred +ves}} = \frac{\text{True +ves}}{\text{True +ves + false +ves}}$$

Recall

$$\frac{\text{True +ves}}{\#\text{ actual +ves}} = \frac{\text{True +ves}}{\text{True +ves + false -ves}}$$



F1 Score:

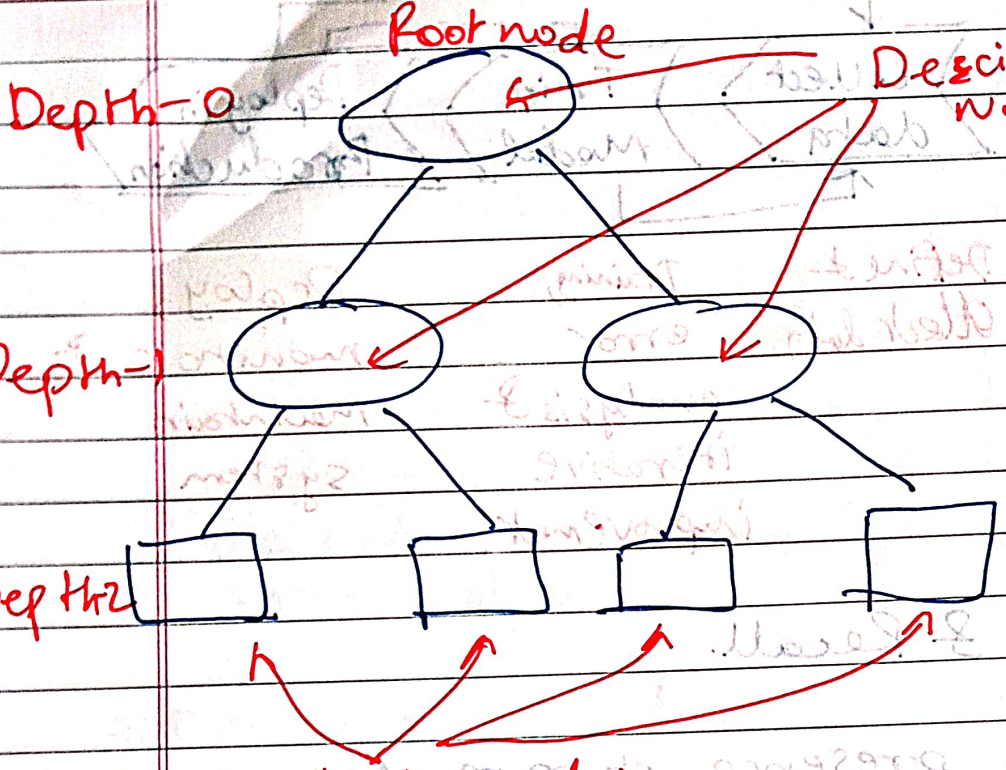
$$\text{f score} = \frac{1}{\frac{1}{2}\left(\frac{1}{P}+\frac{1}{R}\right)} = \frac{2PR}{P+R}$$

Harmonic mean

Date
Page

Course 2 week 4

★ features take categorical or discrete values



★ A Decision tree is a non-parametric supervised machine learning algorithm.

Decision 1: ~~Which~~ How to chosse what feature to split on at each node?

- Maximize Purity (minimize impurity)

Decision 2: When do u stop splitting.

- When a node is 100% one class.
- When splitting a node will result in the tree exceeding a maximum depth
- When improvements in purity score are below ~~one other~~ a threshold.
- When number of examples in a node is below a threshold.

Measuring purity

→ Entropy as a measure of purity

P_0, P_1 → 2 classes in binary classification

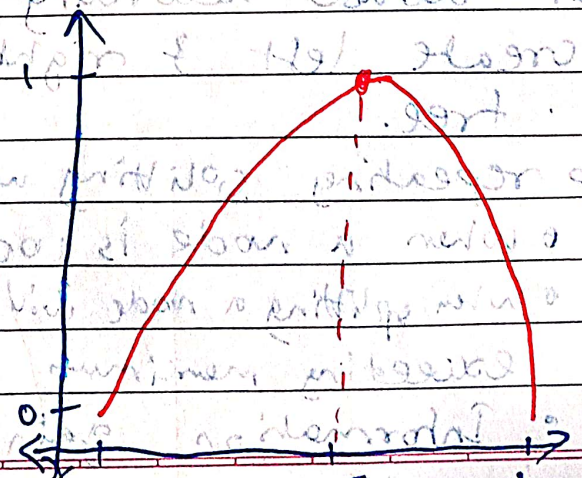
$P_0 = 1 - P_1$

$$H(p_1) = -p_1 \log_2(p_1) - p_0 \log_2(p_0)$$

OR

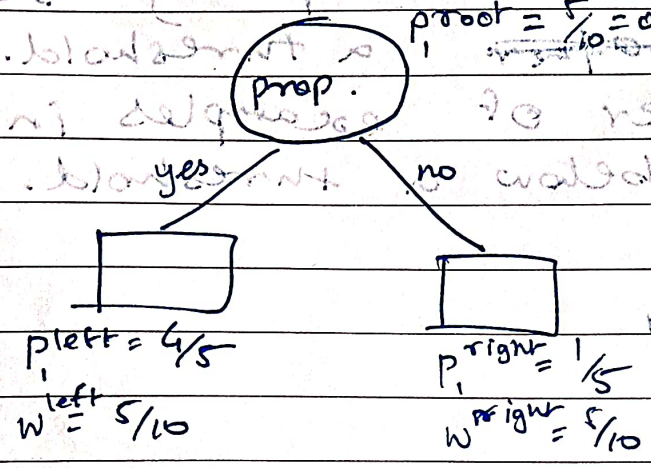
$$H(p_1) = -p_1 \log_2(p_1) - (1-p_1) \log_2(1-p_1)$$

Entropy function



* Choosing a split

→ A split is chosen based on information gain.
Highest information gain is chosen.



Information gain

$$H(p_1^{root}) - \left(w^{left} H(p_1^{left}) + w^{right} H(p_1^{right})\right)$$

* Decision Tree Training

- Start with all examples at the root node.
- Calculate information gain for all possible features, & pick the one with the highest information gain.
- Split dataset according to selected feature, & create left & right branches of the tree.
- keep repeating splitting until stopping criteria is met:
  - when a node is 100% one class.
  - When splitting a node will result in the tree exceeding maximum depth
  - Information gain from additional splits is less than the

Recursive splitting algorithm

Date
Page

→ ○ When the no. of examples in a node is below a threshold.

Tree Ensemble: collection of decision trees.

→ Random forest algorithm.

Given a training set of size m:

for b=1 to B:
Use sampling with replacement to create a new training set of size m.
Train a new decision tree on this dataset.

★ At each node, when choosing a feature to split, if n features are available, pick a random subset of k<n features and allow the algorithm to only choose from that subset of features.

$$k = \sqrt{n}$$

Date
Page

XGBoost:

- Most popular way of building trees.
- used in many ML competitions.

Boosted trees intuition:

Given training set of size m:
for b=1 to B:
- Use sampling with replacement to create a new training set of size m.

But instead of picking all examples with equal (1/m) probability, make it more likely to pick misclassified examples from previously trained trees.

- Train a new decision tree.

XGBoost → eXtreme Gradient Boosting

- open source implementation of boosted trees.
- fast efficient implementation,
- good choice of default splitting criteria & criteria when to stop splitting
- Built in regularization for overfitting.
- highly competitive algorithm for ML competitions

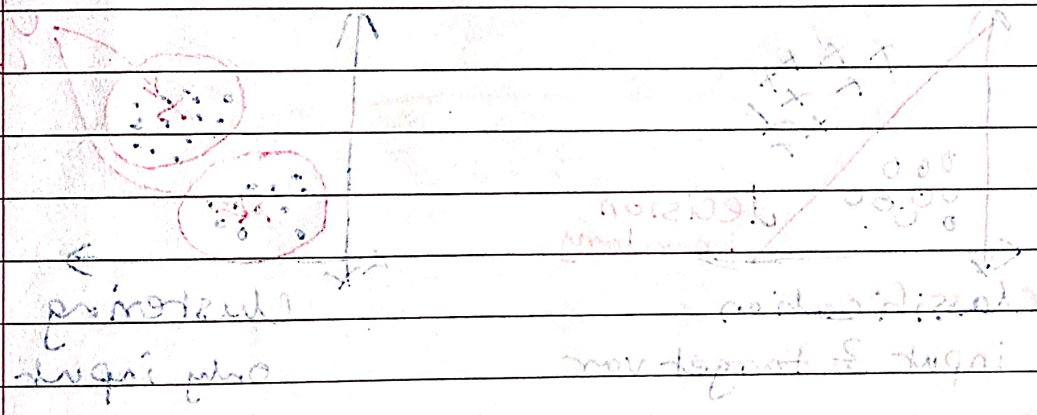
When to use Decision Trees?

DT:

- works well on tabular (structured) data. (not in images, audio, text).
- fast to train

NN:

- works well on all types of data.
- slower than a DT.
- works with transfer learning.
- Easier to string together multiple NN.



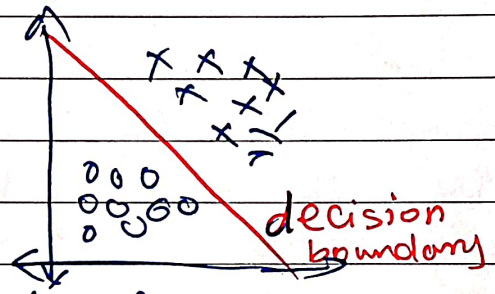
Course 3 Week 1

Unsupervised Learning

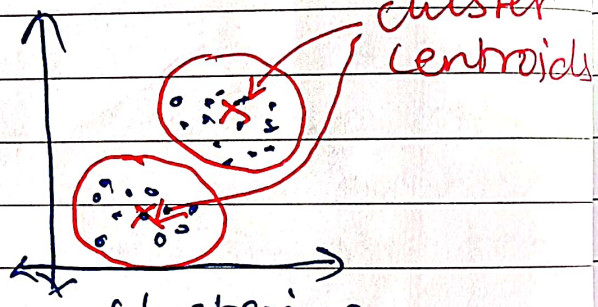
1) Clustering Algorithms
2) Anomaly detection
- Recommender systems (week 2)
- Reinforcement learning (week 3)

* Clustering

→ Grouping similar data points together.



Classification
input & target var



Clustering
only input

K-means clustering

- Randomly initialize K cluster centroids

$\mu_1, \mu_2, \mu_3, \cdots, \mu_K$

with each being a vector of n features

repeat {

for $i = 1$ to m

$c^{(i)}$:= index (from 1 to K) of cluster centroid closest to $x^{(i)}$

$\min_k \|x^{(i)} - \mu_k\|^2$ — it is squared.

for $k = 1$ to K

μ_k := average (mean) of points assigned to cluster k

}

Date
Page

K-means optimization objective

$c^{(i)}$ = index of cluster $(1, 2, \ldots K)$ to which example $x^{(i)}$ is currently assigned.

μ_k = cluster centroid k

$\mu_{c^{(i)}}$ = cluster centroid of cluster to which example $x^{(i)}$ has been assigned.

Cost function:

$$J(c^{(1)}, \ldots, c^{(m)}, \mu_1, \ldots, \mu_K) = \min \frac{1}{m} \sum_{i=1}^{m} \|x^{(i)} - \mu_{c^{(i)}}\|^2$$

It is called distortion function

Anomaly Detection

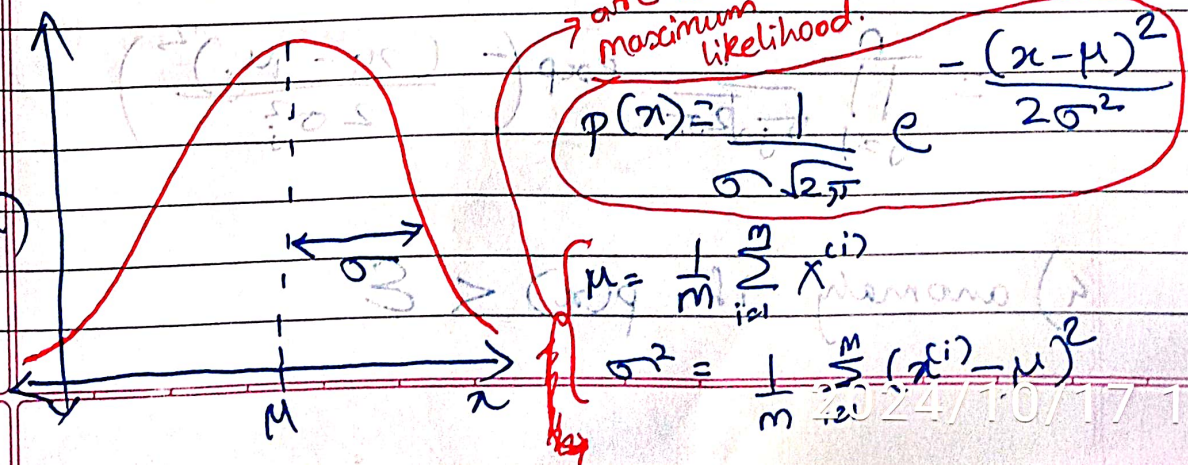
#Gaussian (Normal distribution)

say x is a number

prob. of x is determined by a gaussian with a mean μ, variance σ^2

are called as maximum likelihood.

$$p(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$



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

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

2024/10/17 12:38

Density Estimation

Training set = $\{\vec{x}^{(1)}, \vec{x}^{(2)}, \ldots, \vec{x}^{(m)}\}$

Each example $\vec{x}^{(i)}$ has n features

$$p(\vec{x}) = p(x_1; \mu_1, \sigma_1^2) * p(x_2; \mu_2, \sigma_2^2) * p(x_3; \mu_3, \sigma_3^2) * \ldots * p(x_n; \mu_n, \sigma_n^2)$$

$$p(\vec{x}) = \prod_{j=1}^{n} p(x_j; \mu_j, \sigma_j^2)$$

Anomaly detection algorithm

1) Choose parameters x_i that you think might be indicative of anomalous examples.

2) Fit parameters $\mu_1, \ldots, \mu_n, \sigma_1^2, \ldots, \sigma_n^2$

$$\mu_j = \frac{1}{m} \sum_{i=1}^{m} x_j^{(i)} \qquad \sigma_j^2 = \frac{1}{m} \sum_{i=1}^{m} (x_j^{(i)} - \mu_j)^2$$

3) Given a new example x, compute $p(x)$

$$p(x) = \prod_{j=1}^{n} p(x_j; \mu_j, \sigma_j^2)$$

$$= \prod_{j=1}^{n} \frac{1}{\sigma_j \sqrt{2\pi}} \exp\left(-\frac{(x_j - \mu_j)^2}{2\sigma_j^2}\right)$$

4) anomaly if $p(x) < \varepsilon$

2024/10/17

Course 3 week 2

* Recommender systems.

If you predicting for a single user, It is similar to linear regression.

fit a line $wx+b$ based on users choices and the predict what to recommend.

To learn parameters $w^{(j)}, b^{(j)}$ for the user j

Cost f^{n} of Lin. regr.

$$J(w^{(j)}, b^{(j)}) = \frac{1}{2} \sum_{i:r(i,j)=1} \left(w^{(j)} \cdot x^{(i)} + b^{(j)} - y^{(i,j)}\right)^2 + \frac{\lambda}{2} \sum_{k=1}^{n} \left(w_k^{(j)}\right)^2$$

Only if user has rated something

regularisation ~~parameter~~ term

k = no. of features

To learn parameters $w^{(1)}, b^{(1)}, w^{(2)}, b^{(2)}, \ldots, w^{(n_u)}, b^{(n_u)}$ for all users;

$$J\left(\begin{matrix} w^{(1)}, w^{(2)}, \cdots w^{(n_u)} \\ b^{(1)}, b^{(2)}, \cdots b^{(n_u)} \end{matrix}\right) = \frac{1}{2} \sum_{j=1}^{n_u} \sum_{i:r(i,j)=1} \left(w^{(j)} \cdot x^{(i)} + b^{(j)} - y^{(i,j)}\right)^2 + \frac{1}{2} \sum_{j=1}^{n_u} \sum_{k=1}^{n} \left(w_k^{(j)}\right)^2$$

Cost fn to learn $x^{(1)}, \ldots, x^{(n_m)}$:
for all features

$$\min_{x^{(1)}, \ldots, x^{(n_m)}} \frac{1}{2} \sum_{i=1}^{n_m} \sum_{j: r(i,j)=1} \left(w^{(j)} \cdot x^{(i)} + b^{(j)} - y^{(i,j)} \right)^2 + \frac{\lambda}{2} \sum_{i=1}^{n_m} \sum_{j: r(i,j)=1} \left(x_k^{(i)} \right)^2$$

Putting them together we get cost fn

$$\min_{w^{(1)}, \ldots, w^{(n_u)}, b^{(1)}, \ldots, b^{(n_u)}, x^{(1)}, \ldots, x^{(n_m)}} J(w, b, x) = \frac{1}{2} \sum_{(i,j): r(i,j)=1} \left(w^{(j)} \cdot x^{(i)} + b^{(j)} - y^{(i,j)} \right)^2 + \frac{\lambda}{2} \sum_{j=1}^{n_u} \sum_{k=1}^{n} \left(w_k^{(j)} \right)^2 + \frac{\lambda}{2} \sum_{i=1}^{n_m} \sum_{k=1}^{n} \left(x_k^{(i)} \right)^2$$

2024/10/17

Gradient descent

Collaborative filtering

using linear regr algo,

repeat {

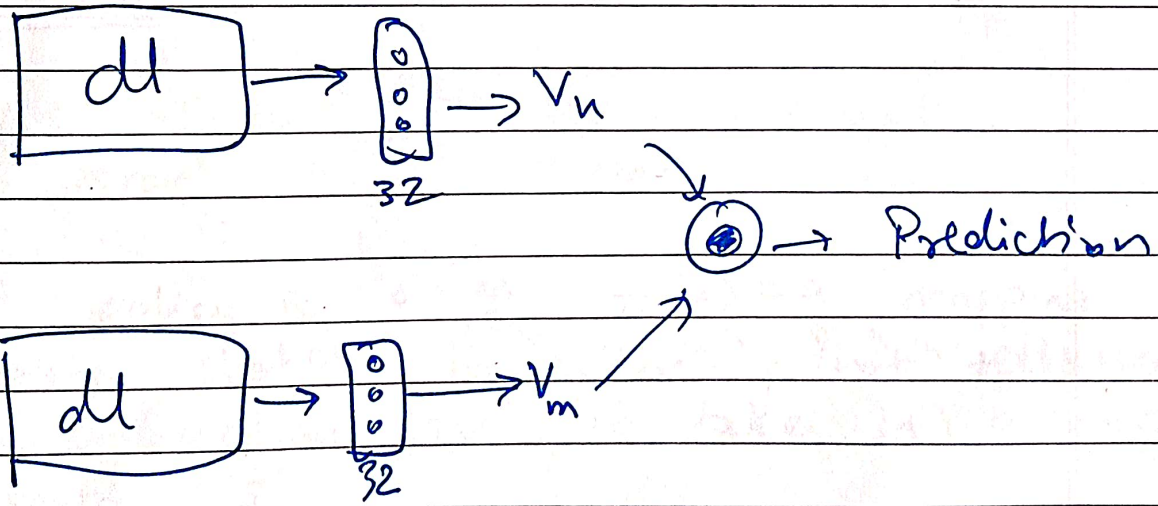
$$w_i^{(j)} = w_i^{(j)} - \alpha \frac{\partial}{\partial w_i^{(j)}} J(w, b, x)$$

$$b^{(j)} = b^{(j)} - \alpha \frac{\partial}{\partial b^{(j)}} J(w, b, x)$$

$$x_k^{(i)} = x_k^{(i)} - \alpha \frac{\partial}{\partial x_k^{(i)}} J(w, b, x)$$

} simultaneous updates.

Content-based filtering



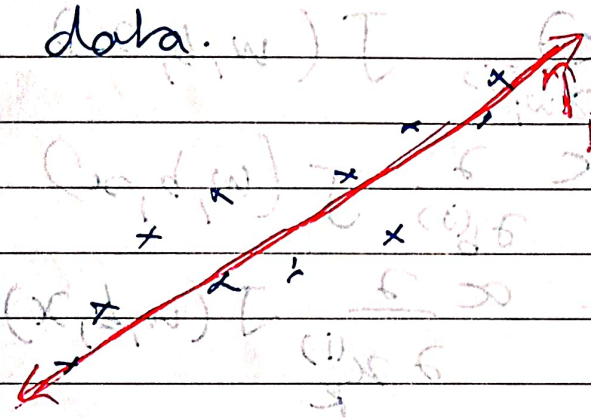
Cost fn:

$$J = \sum_{(i,j):r(i,j)=1} \left(v_u^{(j)} \cdot v_m^{(i)} - y^{(i,j)}\right)^2$$

+ NN regularisation term.

* Principal component analysis:
→ It is an unsupervised learning algo
PCA algorithm: Take high dimension data and reduce it to 2D data for visualization.

→ Find axis with maximum variance of data.



→ PCA is not linear regression.

Course 3 week 3

* Reinforcement learning.

$x \longrightarrow y$

reward function

Applications:

→ controlling robots.
→ factory optimization
→ financial (stock) trading
→ Playing games (including video games)

discount factor: makes the algo impatient

$\gamma = 0.9$ or 0.99 or 0.995.

* Policy (π)

$S \xrightarrow[\pi]{\text{Policy}} a$

state → action

⇒ A policy is a fn $\pi(s) = a$ mapping from states to actions, that tells you what action a to take in a given state s.

→ find a policy π that tells you what action ($a = \pi(s)$) to take in every state (s) so as to maximize return.

Date
Page

Markov Decision Process (MDP)

→ Actions to be taken depends on the current state of the system only.

→ which means that results prev or after this current state don't matter.

state - action value function

Q(s, a) = return if you
- Start in state S.
- take action a (once)
- then behave optimally after

The best possible return from state s is $\max_a Q(s,a)$

The best possible action a in state s is the action 'a' that gives $\max_a Q(s,a)$

$$Q(s,a) = R(s) + \gamma \max_{a'} Q(s',a')$$

Bellman equation

Date
Page

* Random (stochastic) environment:

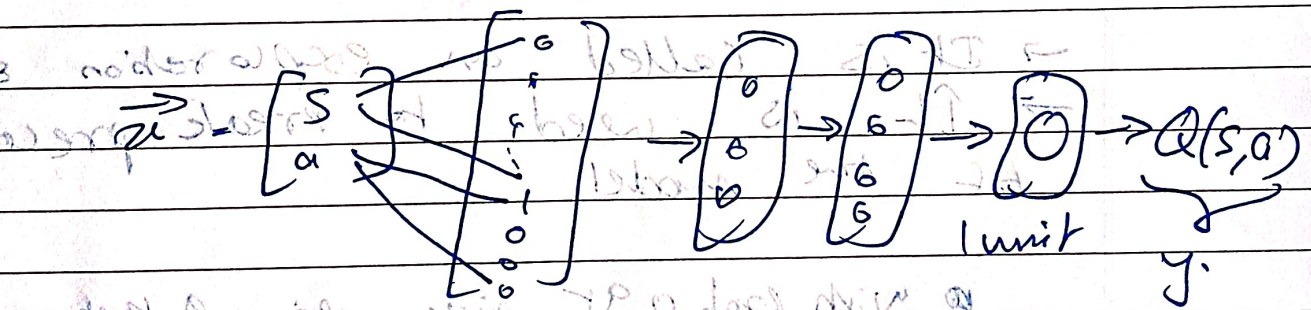
$$Q(s,a) = R(s) + \gamma E\left[\max_{a'} Q(s',a')\right]$$

* Continuous state space:

current state is denoted by a vector

$$S_1 = \begin{bmatrix} \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \end{bmatrix}$$

Deep reinforcement learning.



Deep Q-network algorithm: (DQN)

repeat {

→ take actions in the lunar ~~code~~ lander. get $(s, a, R(s), s')$

→ store 10,000 most recent $(s, a, R(s), s')$ tuples.

→ Train NN:

create training set of 10,000 eg using

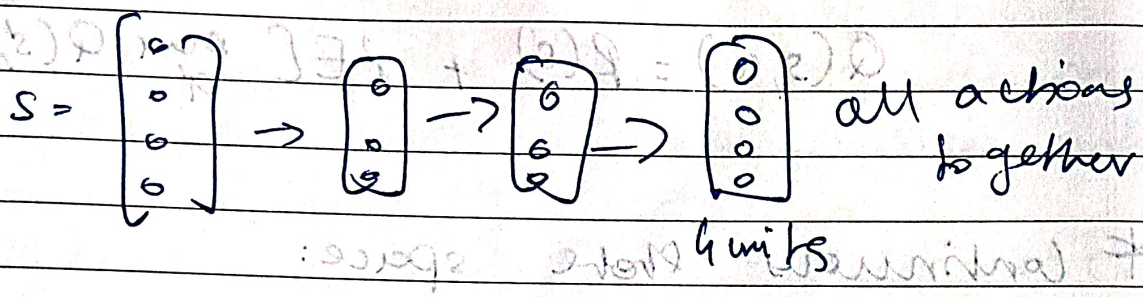
$x = (s,a)$ & $y = R(s) + \gamma \max_{a'} Q(s',a')$

Train Q_{new} such that $Q_{new}(s,a) \approx y$.

→ set $Q = Q_{new}$.

2024/10/17 12:38

Refined DQN



* ε-greedy policy.

→ its another refinement.

→ Epsilon greedy ~~pto~~ policy is used to take actions while learning.

→ make the algo randomly try new actions to make open to new options

→ It is called an exploration step.

→ It is used to break preconceptions of the model.

- with prob 0.95 pick action a that maximizes $Q(s, a)$
- with prob 0.05 pick an action randomly.

Start ε high

↓

gradually decrease ε

* Minibatch & soft updates
→ more refinements.

Minibatch: instead of using all eg of training set at each iteration, use only a subset of the training set.

Soft update: don't change new Q drastically.

$$w = 0.01\, w_{new} + 0.99\, w$$
$$b = 0.01\, b_{new} + 0.99\, b.$$

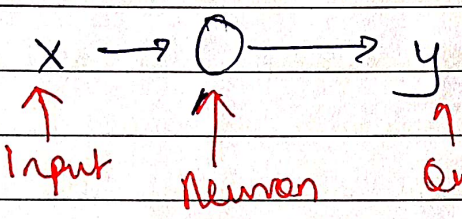
—x o x—

Fin

Deep Learning Specialization

Course 1 : Neural Networks & Deep Learning

Week 1 : Introduction



Simplest neural network

Week 2 : Neural Networks Basics

Week 3 : Shallow Neural Networks ~~overview~~

ReLU is the most useful activation fn

→ Why do u need non-linear fn

- In order to learn interesting features
- If all layers are linear, then no. matter layers or nodes u use, the overall model is just linear

Week 4 : Deep Neural Networks

* All the weights should be initialized randomly instead of all weights to zeroes.

2024/10/17

Date
Page

Parameters vs Hyperparameters.

→ Parameters: $W^{[1]}, b^{[1]}, W^{[2]}, b^{[2]}, \ldots$

→ Hyperparameters: learning rate α
- # no. of iterations.
- # hidden layers
- # hidden units.
- choice of activation fn's.
- Momentum term
- Minibatch size
- regularizations.
- :
- :

Date
Page

Course 2: Improving Deep Neural Networkes

Week 1: Practical aspects of Deep learning

(1) → overfitting / high variance can be solved by regularization.

(2) → Dropout regularization is also very powerful.

Why does dropout work?

→ Intuition: Can't rely on any one feature, so have to spread out weights.

→ This will shrink weights.

* The more the nodes in a layer, the lesser should be the dropout probability. The lesser the nodes, the more the probability.

(3) → Data augmentation can also help in overfitting

(4) → Early stopping

optimizing.

① → normalizing the inputs..
- It decreases the skewness of the cost fn.

② → If you have a very deep neural network, the values of y can explode. This is called "exploding gradient".

③ → On the reverse if the values of weights is small then the value of y can vanish. This is called "vanishing gradient".

weight initialization can help with this.

larger n ⟶ smaller w_i

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

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

for tanh activation → use constant one.

This is called a "Xavier initialization"

Date
Page

Week 2: Optimization algorithms:

optimization algorithms help train models faster.

① Mini-Batch Gradient Descent.

computing cost on a part of your data.
→ This saves memory.

If mini-batch size = m : Batch gradient descent
size = 1 : stochastic gradient descent
size bet^n 1 to m that is
not too big or too small,
gives the fastest learning

how to choose mini-batch size?

→ If target set is (≤ 2000):
use batch grad desc.

typical mini batch sizes:

64, 128, 256, 512, 1024

* Just make sure mini batch size fits in CPU / GPU memory

2024/10/17

Date
Page

Exponentially weighted (moving) averages:

→ It is used to describe a time series

→ It decides how important the current observation is, as compared to previous observations.

$$V_t = \beta V_{t-1} + (1-\beta)\theta_t$$

β is selected by user.

→ If you expand this for 100 eg

$$V_{100} = 0.1\theta_{100} + (0.1)(0.9)\theta_{99} + (0.1)(0.9)^2\theta_{98} + (0.1)(0.9)^3\theta_{97} + \cdots$$

so this decays exponentially.

It averages $\approx \frac{1}{1-\beta}$ examples.

$\varepsilon = 1-\beta$

Bias correction in Exponentially weighted averages.

for initial weights →

instead of V_t, calculate $\frac{V_t}{1-\beta^t}$

2024/10/17 12:39

Gradient Descent with momentum

→ It uses the exponentially weighted averages.

momentum:

On iteration t:

Compute dw, db on current mini batch.

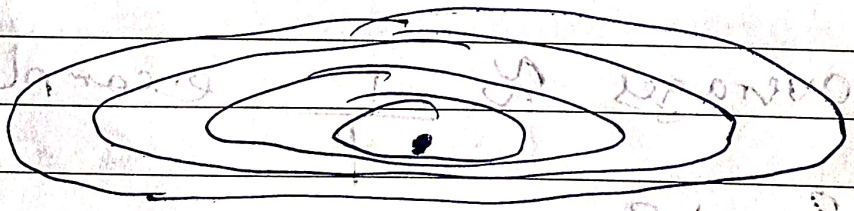
$$V_{dw} = \beta V_{dw} + (1-\beta) dw$$

$$V_{db} = \beta V_{db} + (1-\beta) db.$$

$$w = w - \alpha V_{dw}$$

$$b = b - \alpha V_{db}$$

→ It smooths out the grad descent and prevents oscillations.



↕ slower learning

↔ faster learning.

this is what it does to prevent oscillations.

2024/10/17

Date
Page

③ RMS prop.

→ Root mean square prop.

On iteration t:

compute dW, db on current mini-batch.

$S_{dW} = \beta S_{dW} + (1-\beta)\, dW^2$

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

$W = W - \alpha \frac{dW}{\sqrt{S_{dW}+\varepsilon}}$ $\quad \boxed{\varepsilon = 10^{-8}}$

$b = b - \alpha \frac{db}{\sqrt{S_{db}+\varepsilon}}$

④ Adam algorithm

→ combines momentum & rms prop.

→ Adaptive moment estimation

$V_{dW} = 0,\ S_{dW} = 0,\ V_{db} = 0,\ S_{db} = 0.$

on iteration t:

compute dW, db on curr mini batch.

momentum:
$V_{dW} = \beta_1 V_{dW} + (1-\beta_1)\, dW$
$V_{db} = \beta_1 V_{db} + (1-\beta_1)\, db$

rmsprop:
$S_{dW} = \beta_2 S_{dW} + (1-\beta_2)\, dW^2$
$S_{db} = \beta_2 S_{db} + (1-\beta_2)\, db^2$

for adam, α: needs to tune
$\beta_1 = 0.9$ (dW)
$\beta_2 = 0.999$ (dW^2)
$\varepsilon = 10^{-8}$

$V_{dW}^{corrected} = V_{dW} / (1-\beta_1^t)$, $V_{db}^{corrected} = V_{db} / (1-\beta_1^t)$

$S_{dW}^{corrected} = S_{dW} / (1-\beta_2^t)$, $S_{db}^{corrected} = S_{db} / (1-\beta_2^t)$

$W = W - \alpha \frac{V_{dW}^{corrected}}{\sqrt{S_{dW}^{corrected}+\varepsilon}}$, $b = b - \alpha \frac{V_{db}^{corrected}}{\sqrt{S_{db}^{corrected}+\varepsilon}}$

05

learning rate decay.

→ slowly decrease learning over time.

1 epoch = 1 pass through data.

$$\alpha = \frac{1}{1 + \text{decay rate} \times \text{Epoch no.}} \times \alpha_0$$

$\alpha_0 = 0.2$

decay rate = 1

Epoch	α
1	0.1
2	0.067
3	0.05
4	0.04

Exponential decay:

→ $\alpha = 0.95^{\text{EpochNum}} \cdot \alpha_0$

$\alpha = \frac{k}{\sqrt{\text{Epochnum}}} \cdot \alpha_0$ or $\frac{k}{\sqrt{t}} \cdot \alpha_0$

discrete staircase:

half the α after some time.

2024/10/17

Date
Page

Problem of local optima.

→ Its very unlikely that the grad desc gets stuck in a local optima in a higher dim space.

→ This problem is only possible in lower dim data.

Week 3: Hyperparameter tuning

appropriate scale for hyperparameters

$n^{[l]}$ = 50 - 100

L = 2 - 4

α = 0.0001 — 1 use log scale.

β = 0.9 — 0.999 using log scale of $1-\beta$

Batch Normalisation

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

$$x = x - \mu$$

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

$$x = x / \sigma$$

→ we can normalize activations.

Given some intermediate values in NN
$z^{(1)}, \ldots, z^{(m)}$:

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

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

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

FF

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

$\beta - \mu$

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

this beta is not the momentum beta hyperparameter.

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

learnable parameters of model

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

→ sandwich batch norms bet^n. layers to normalize activations.

→ Batch norm also has a regularization side effect. because it increases the noise in values. This spreads weights across nodes

→ Batch Norm increases learning speed.

Course 3: Structuring Machine Learning Projects

Week 1: ML Strategy

Chain:

Fit training set well on cost f^n

↓

Fit dev set well on cost f^n

↓

Fit test set well on cost f^n.

↓

Performs well in real world.

Date
Page

Using a single number evaluation metric

Instead of trying to improve both precision or recall, just try improving the F1 score.

precision: Of the examples recognised, what are actual.

recall: out of actual, how many recognised correct.

F1 score = "average" of P & R

$\left(\frac{2}{\frac{1}{P}+\frac{1}{R}}\right)$: harmonic mean

→ This applies everywhere

Instead of improving every metric, take a single metric as a benchmark for model performance.

Also compare your errors to human-level performance.

2024/10/17

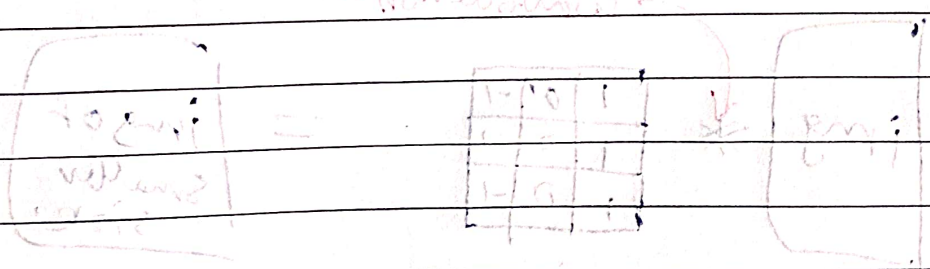
Human-level
↕ avoidable bias
Training error
↕ Variance
Dev error

Train bigger model
Train longer / better optimization algorithms.
NN architecture / hyperparameters search
RNN / CNN

more data.
Regularisation
- L2, dropout, data augmentation
NNorch / hyperparameter search.

Week 2: ML Strategy

* Build the first System quickly, then iterate



Date
Page

Course 4: CNNs

Week 1: Foundations of Convolutional Neural Networks

→ Used to train neural networks with large inputs and large number of parameters.

→ CNNs consist of kernels or "filters".

→ Each kernel in detects a certain pattern in the image.

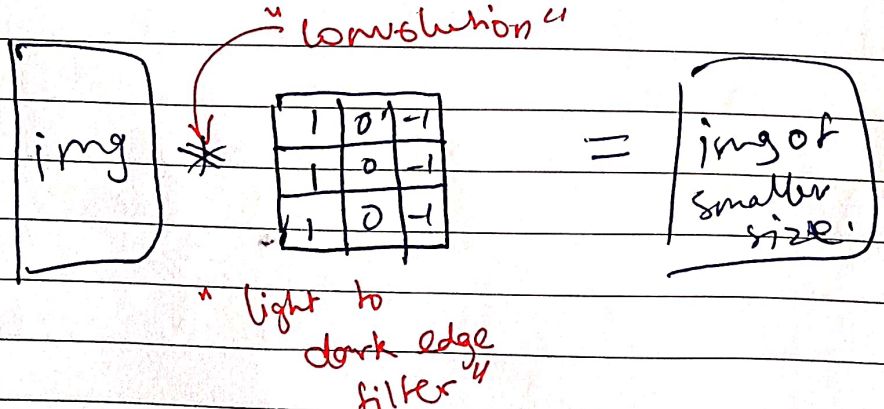
eg)

In first layer: 

In second layer: parts of objects.

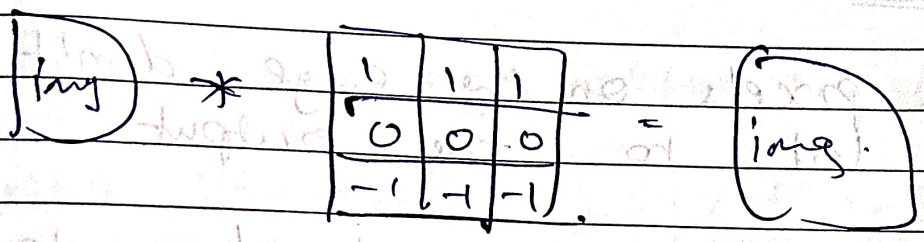
In third layer: objects themselves.

→ Vertical edge detection



2024/10/17

→ Horizontal edge detection



→ Sobel filter

1	0	-1
2	0	-2
1	0	-1

← gives more priority to center pixels.

→ Scharr filter

3	0	-3
10	0	-10
3	0	-3

another filter with same concept taken to extreme.

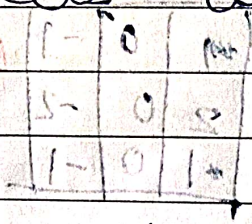
→ Researches used to handpick these kernel weights before but now we implement networks to "learn" the weights of the kernels.

Padding

→ The pixels on the edge don't contribute a lot to the output.

→ So we add a border to the image.

→ This preserves the size of the image & makes edge pixels contribute to the output.



* Valid convolution: no padding

* Same convolution: Pad so that output size is the same as the input size.

$$p = \frac{f-1}{2}$$

p → Padding size

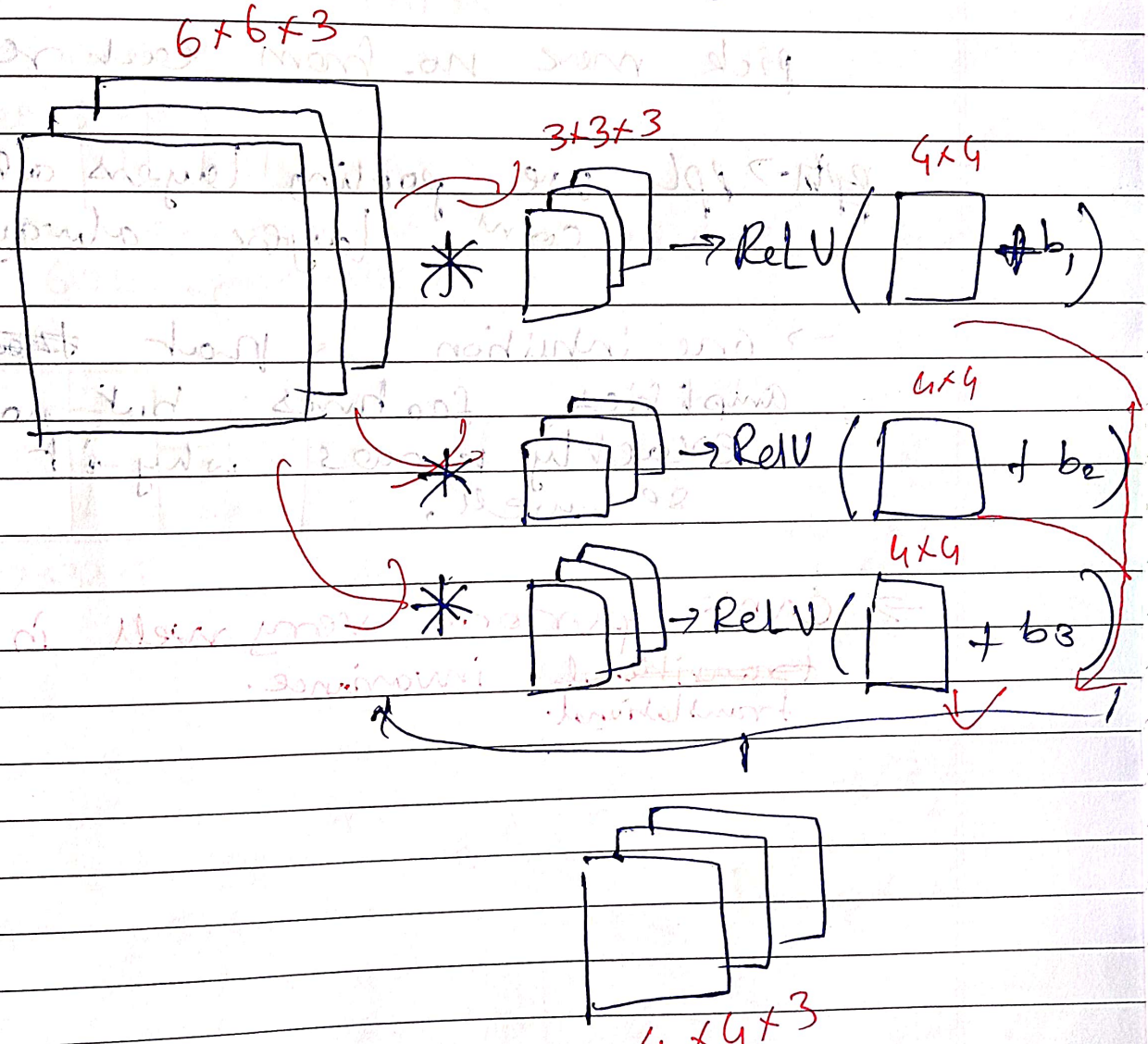
→ Size of kernel.

* Always use odd size filters
1×1, 3×3, 5×5, 7×7.

Stride

Size of image after stride-

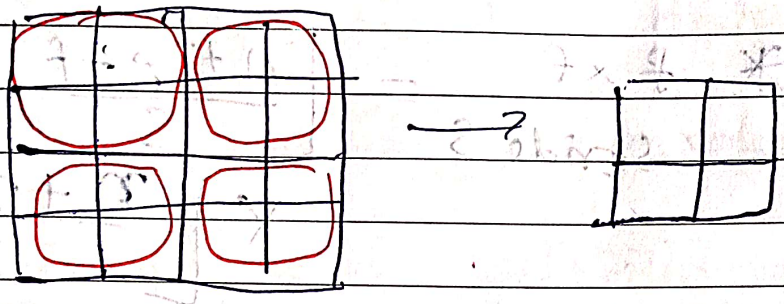
nan $*$ $f \times f$ Padding p stride s $= \left\lfloor \frac{n+2p-f}{s} + 1 \right\rfloor \times \left\lfloor \frac{n+2p-f}{s} + 1 \right\rfloor$



Date
Page

Pooling :

Max Pool

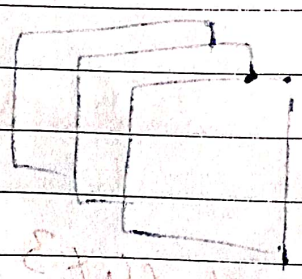


pick max no. from each region.

→ ppl use pooling layers after a conv. layer always.

→ one intuition is that it amplifies features but no one exactly knows why it works so well.

→ CNNs perform very well in ~~translational~~ translational invariance.



Date
Page

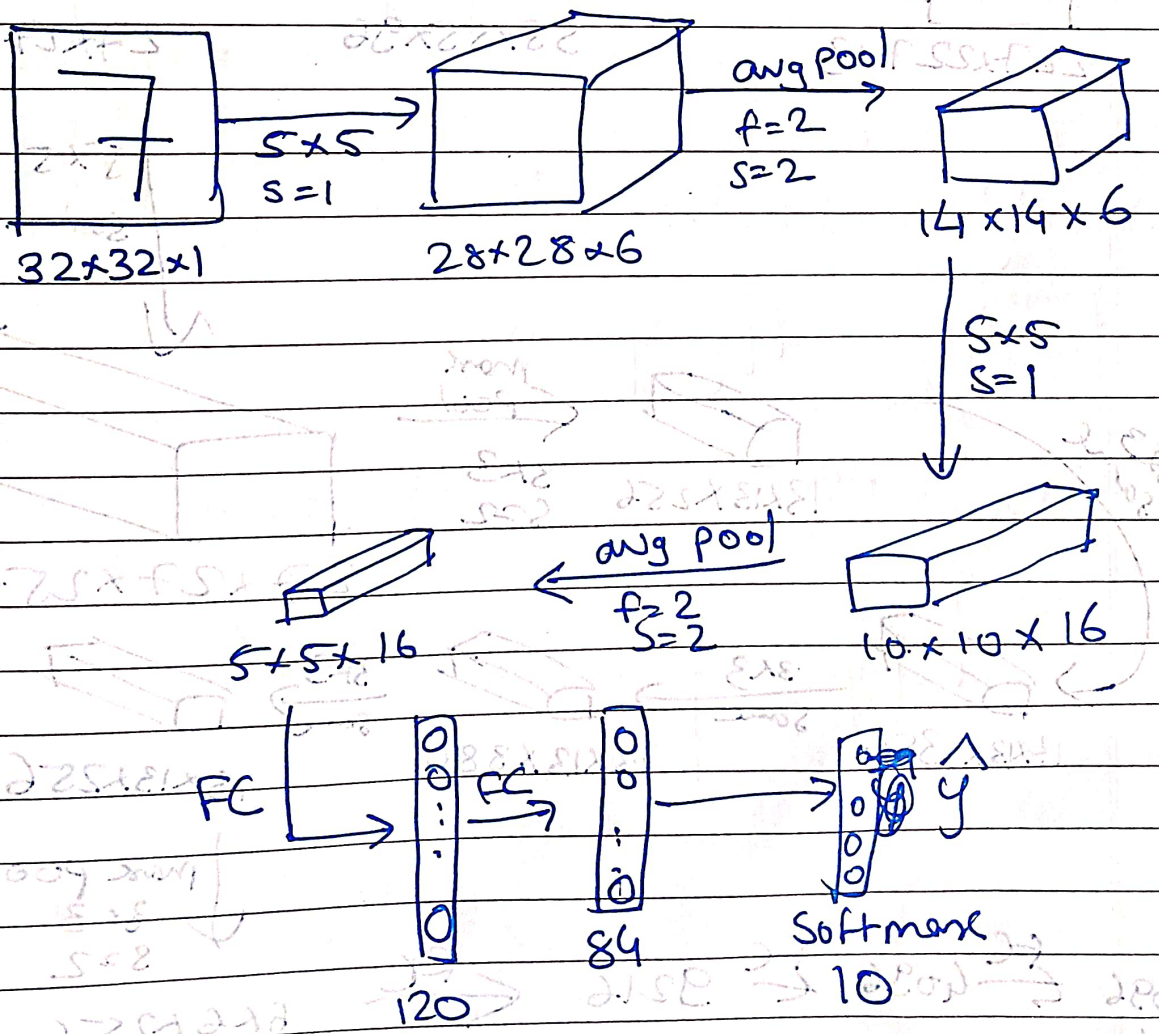
Course 4 week 2
Deep convolutional models: Case Studies.

Outline: LeNet-5
AlexNet
VGG Net
ResNet (152 layer)
Inception

* LeNet-5

(1998)

→ It was trained on grayscale images.
→ 60k parameters



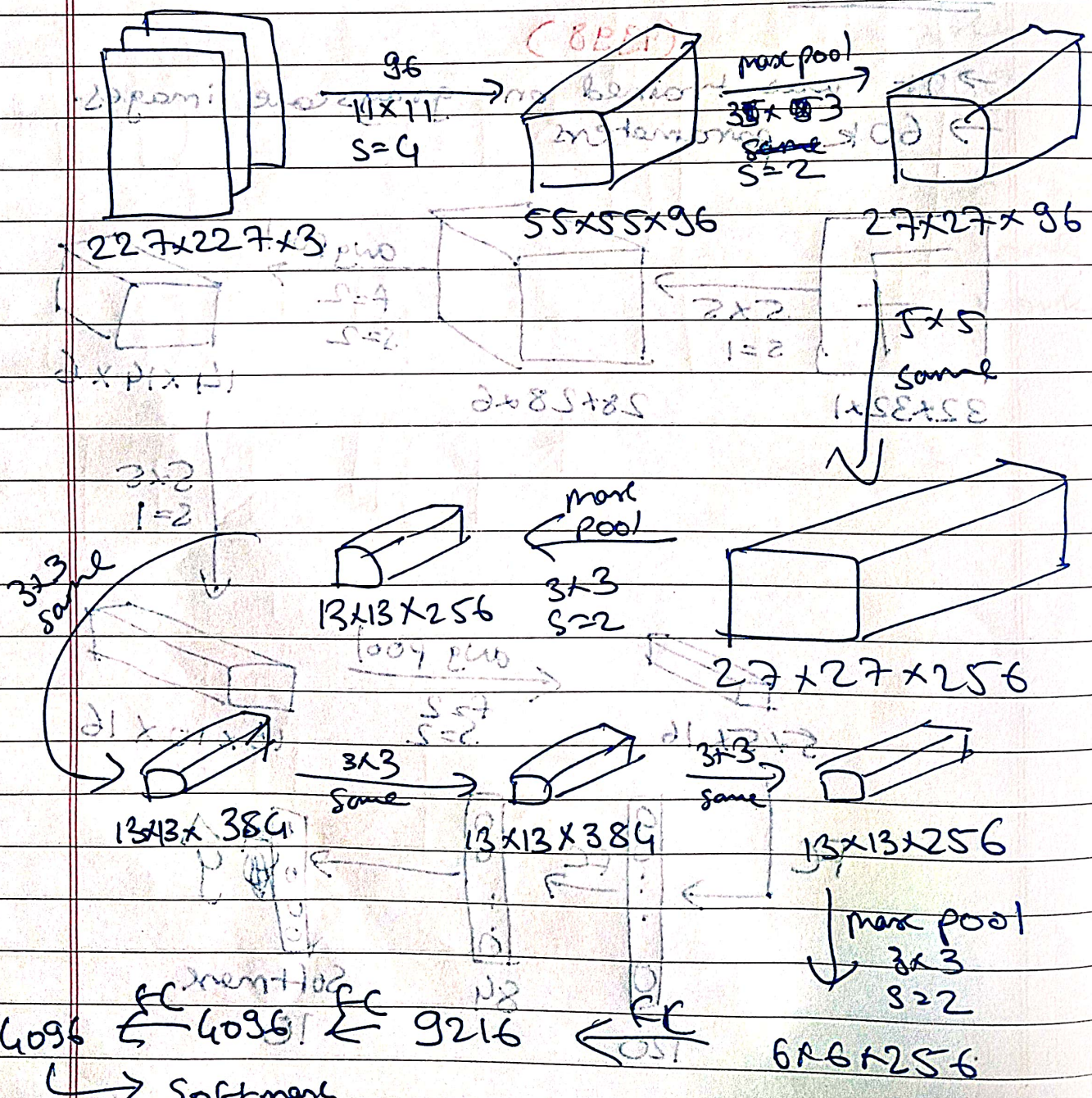
2024/10/17 12:

n_H & n_W ↓, & n_C ↑ as you go forward in NN.

→ At that time ppl were using Sigmoid / tanh activation more other than ReLU

* ~~Alex~~ AlexNet (2012)
~60M parameters

→ named after first author Alex Krizhevsky

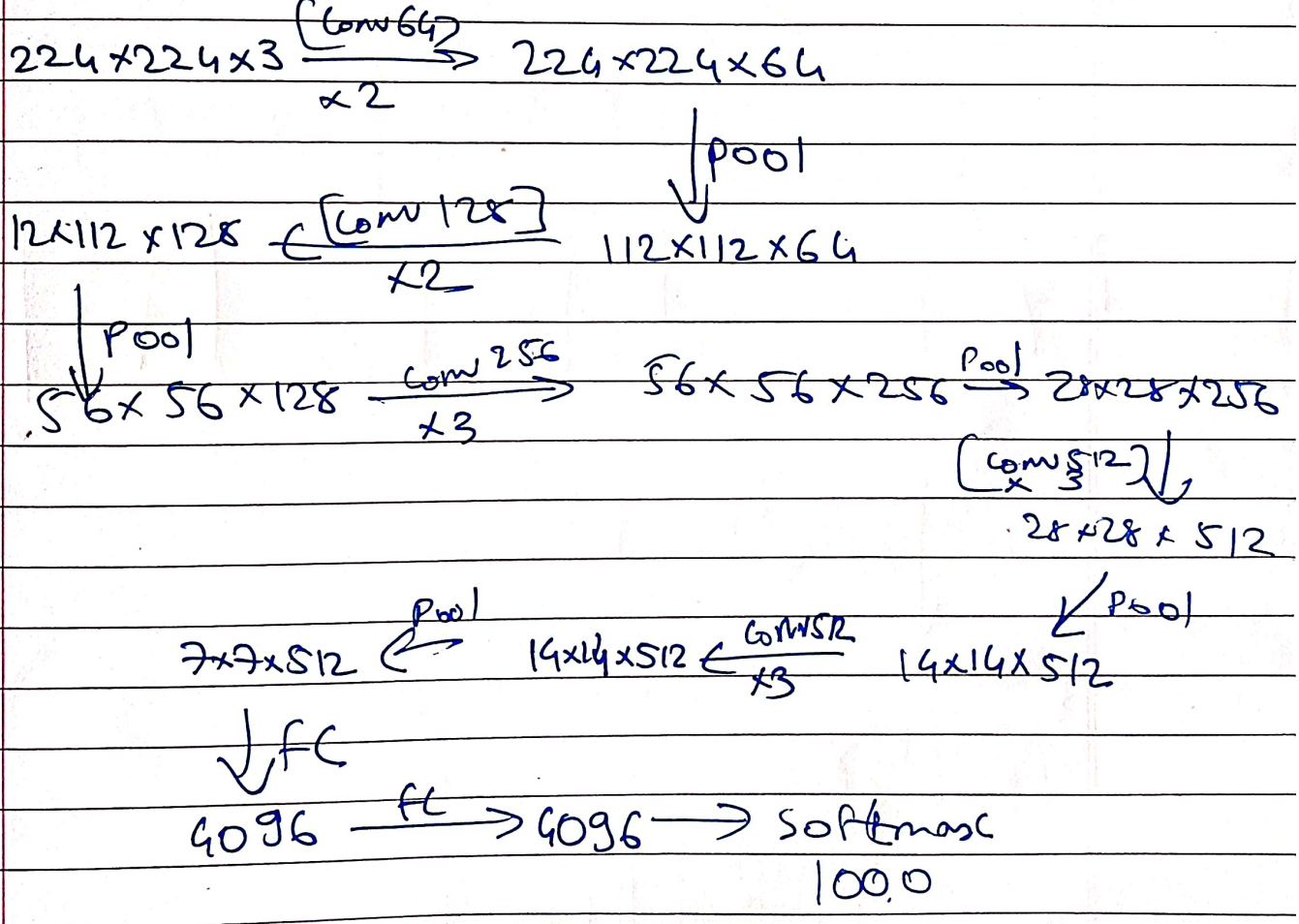


2024/10/17

→ made use of ReLU
→ Multiple GPUs
→ New layer: Local response Normalisation (LRN)
→ Complicated & a lot of hyper parameters.

* VGG-16 (2015)
~ 138 M parameters.

→ simpler
Conv = 3×3 filter
S = 1, same
Max pool = 2×2, S = 2



→ No of filters
64 → 128 → 256 → 512.

→ VGG-19 is an even bigger model but performs almost as well as VGG-16