1. Constraint Based Systems

↳ user specifies his initial preference

↳ user is displayed a set of matching items → with reason why an item was recommended

↳ 2 sets of constraints

↳ User

↳ Customer

↳ uses attributes

✓ constraints ↓ objectives ✗

Components of Constraint-based Systems

- ❑ Customer properties — V_C
- ❑ Product properties — V_{PROD}
- ❑ Constraints — C_R (on customer properties)
- ❑ Filter conditions and product — C_F – relationship between customer
- ❑ Products — C_{PROD} – possible instantiations

Q)

V_C =
{
V_{C1} : [vegan, no_dairy, no_requirements] ..... /* dietary requirements */
V_{C2} : [fast_food, traditional, continental, none] ..... /* food preference */
V_{C3} : [high, medium, low] ..... /* spending style */
...
}

V_{PROD} =
{
V_{P1} : [pizza, BBQ, burger, biryani, italian] ..... /* type of food */
V_{P2} : [300, ... , 4000] ..... /* food price */
V_{P3} : [*plaintext*] ..... /* name of food */
...
}

C_R =
{
C_{R1} : v_{C1} = vegan → v_{C2} ≠ fast food
C_{R2} : v_{C2} = continental → v_{C3} ≠ low
...
}

C_F =
{
C_{F1} : v_{C1} = vegan → v_{P1} ≠ BBQ ∧ v_{P1} ≠ burger
C_{F2} : v_{C3} = low → v_{P2} < 1000
...
}

C_{PROD} =
{
C_{PROD1} : v_{P3} = afghani tikka pizza ∧ v_{P2} = 1300
C_{PROD2} : v_{P3} = chicken burger ∧ v_{P2} = 750 → less than 1000
C_{PROD3} : v_{P3} = chicken biryani ∧ v_{P2} = 400 →
C_{PROD4} : v_{P3} = alfredo pasta ∧ v_{P2} = 1200
...
}

if user prefences = No dairy, none, low

recommended items: chicken burger, chicken biryani

CONS

↳ can get 0 recommendations — as customer given req might be such that no product shown

[solution

1. make a default product
2. Relax constraints → till 3 violations recommend

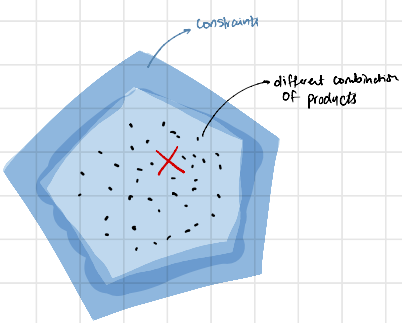
↳ Customer might be unsure about the values of their attributes → customer doesnt know if he is a low/high spender → SOLUTION → suggest resonable default values for attributes

↳ Bussiness rules change frequently → will have to hire for every change → EXPENSIVE

↳ No serendipity

↳ optimization algos are slow

↳ domain expertise needed



↳ Is a subset selection problem

Different techniques to find subsets → to optimise

1. QuickPlan
2. Linear Programming
3. Gradient Desecent → terrible idea

2. Case Based

- can't be sparse → should have all info
- unlike collaborative filtering

↳ uses attributes

↳ items are retrived using similarity measures

$$sim(p, REQ) = \frac{\sum_{r \in REQ} Wr \times sim(p, r)}{\sum_{r \in REQ} Wr}$$

(Wr: weight for requirement)

0) Allowed attributes:

P: {12, 25, 55} (max price)

M: {8, 10, 48, 54, 60}

L: {3, 4, 5, 6}

W: {Yes, No}

Weights attributes:

$W_{price} = 1.5$

$W_{megapixel} = 1$

$W_{LCD} = 1$

* if this data not given get max value from table

↳ Cases

Camera	Price	Megapixel	LCD	waterproof
C1	12	8	3	NO
C2	25	48	4	NO
C3	55	60	6	Yes

NORMALISE

Camera	Price	Megapixel	LCD	waterproof
C1	0.218 (12/55)	0.133 (8/60)	0.428 (3/6)	0 (NO/NO)
C2	0.454	0.8	0.571	0
C3	1	1	0.857	1

↳ User req (user comes with this query)

Price	Megapixel	LCD	waterproof
20	40	5	NO

NORMALISE

Price	Megapixel	LCD	waterproof
0.363 (20/55)	0.666	0.714	0

↳ Find Similarities

$Sim(P_R, P_{C1}) = 1 - |0.363 - 0.218| = 0.855$ (user price, case price)

$Sim(M_R, M_{C1}) = 1 - |0.666 - 0.133| = 0.467$

$Sim(L_R, L_{C1}) = 1 - |0.714 - 0.428| = 0.714$ (L_R: user req, L_{C1}: cases)

↳ Find similarity of user requirements with all

$$Sim(User_{REQ}, C1) = \frac{1.5(0.855) + 1(0.467) + 1(0.714)}{1.5 + 1 + 1} = 0.703$$

(weight attributes)

Ans is highest score

$Sim(User_{REQ}, C2) =$

$Sim(User_{REQ}, C3) =$

PRO

- no 0 recommendations
- has no constraints
  so no domain expert needed → ?
- low cost technique

CON

- attributes arent always independent in reality
  but we assume they are

Knowledge-based Systems: Limitations

- ❑ Cost of knowledge acquisition and domain expertise
- ❑ Attributes (preferences) are not always independent

a) Consider the given table depicting the features of laptops available for purchase.

	HD Display	RAM (*GB*)	Storage (*GB*)	Price (*k*)
L1	Yes	4	512	52
L2	No	8	1020	80
L3	Yes	16	1020	96

Use case-based knowledge driven technique to recommend the most suitable laptop for the following user requirements:

HD Display	RAM (*GB*)	Storage (*GB*)	Price (*k*)
No	16	512	60

Note: Assume all attribute weights to be 1.

b) List down some limitations of constraint-based systems.

Normalise cases

	HD	RAM	STO	PRI
L1	1	0.25	0.502	0.542
L2	0	0.5	1	0.833
L3	1	1	1	1

Normalise User

	HD	RAM	STO	PRI
User	0	1	0.502	0.625

Find Similarities

With L1

HD = 1 − |0 − 1| = 0

RAM = 1 − |1 − 0.25| = 0.25

STO = 1 − |0.502 − 0.502| = 1

PRI = 1 − |0.625 − 0.542| = 0.917

$$\text{Similarity} = \frac{0 + 0.25 + 1 + 0.917}{1 + 1 + 1 + 1} = 0.542$$

With L2

HD = 1 − |0 − 0| = 1

RAM = 1 − |1 − 0.5| = 0.5

STO = 1 − |0.502 − 1| = 0.502

PRI = 1 − |0.625 − 0.833| = 0.792

$$\text{Similarity} = \frac{1 + 0.5 + 0.502 + 0.792}{1 + 1 + 1 + 1} = 0.68$$

→ will be recommended

With L3

HD = 1 − |0 − 1| = 1

RAM = 1 − |1 − 1| = 1

STO = 1 − |0.502 − 1| = 0.502

PRI = 1 − |0.625 − 1| = 0.625

$$\text{Similarity} = \frac{1 + 1 + 0.502 + 0.625}{1 + 1 + 1 + 1} = 0.531$$

b) Limitations of Constraint-based Recommender Systems:

- It may generate zero recommendations in some cases.
- Users are not always sure of what their full requirements are.
- If business rules often change, updates to the knowledge-base and constraints need to be made.
- Domain expertise is required.

Consider the given table depicting the features of processors available for purchase.

	Cores	Price (in thousands)	L3 Cache (in MB)
CPU 1	6	69	32
CPU 2	12	78	32
CPU 3	8	74	64

Use case-based knowledge driven technique to recommend the most suitable processor for the following user requirements.

Feature	Cores	Price (in thousands)	L3 Cache (in MB)
Requirement	12	68	64
Weight	1.25	1.40	1.12

Normalised cases

	12 Cores	78 Price	64 L3 cache
CPU 1	0.5	0.885	0.5
CPU 2	1	1	0.5
CPU 3	0.667	0.949	1

Normalised user

	12 Cores	78 Price	64 L3 cache
Requirement	1	0.872	1
Weight	0.104	0.018	0.0175

sum of weight: 0.1395

Find Similarity

CPU 1 Requirement

$Sim(R_c, C) = 1 - |1 - 0.5| = 0.5$

$Sim(R_p, P) = 1 - |0.872 - 0.885| = 0.981$

$Sim(R_{L3}, L3) = 1 - |1 - 0.5| = 0.5$

$$Sim(CPU1) = \frac{(0.104)(0.5) + (0.018)(0.987) + (0.0175)(0.5)}{0.1395}$$

$$= 0.563$$

CPU 2

$Sim(R_c, C) = 1 - |1 - 1| = 0.872$

$Sim(R_p, P) = 1 - |0.872 - 1| = 0.872$

$Sim(R_{L3}, L3) = 1 - |1 - 0.5| =$

$$Sim(CPU2) = \frac{(0.104)(0.872) + (0.018)(0.872) + (0.0175)(0.5)}{0.1395}$$

$= 0.825$ → will be recommended

CPU 3

$Sim(R_c, C) = 1 - |1 - 0.667| = 0.667$

$Sim(R_p, P) = 1 - |0.872 - 0.949| =$

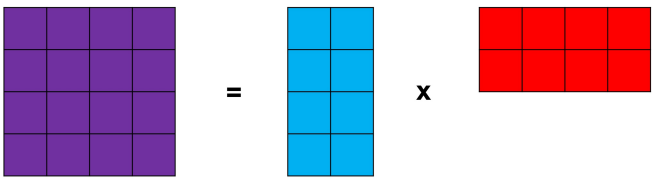
$Sim(R_{L3}, L3) = 1 - |1 - 1| =$

$$Sim(CPU3) = \frac{(0.104)(0.667) + (0.018)(0.923) + (0.0175)(1)}{0.1395}$$

$= 0.742$

Matrix Factorization → focuses on STORAGE

↳ decomposes interaction matrix into a product of 2 or 3 matrices → you can have any number of factors

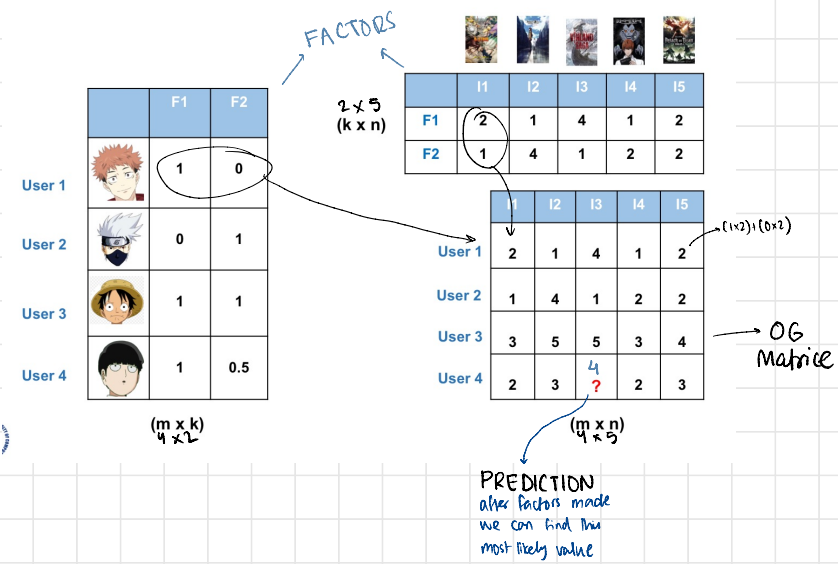


SAVING

4×4 = 16	4×2 = 8	4×2 = 8	= 16	→ 16 − 16 = 0
5×5 = 25	5×2 = 10	2×5 = 10	= 20	→ 25 − 20 = 5
100×100 = 10000	100×5 = 500	x100×5 = 500	= 1000	→ 10,000 − 1000 = 9,000

m = rows
n = columns

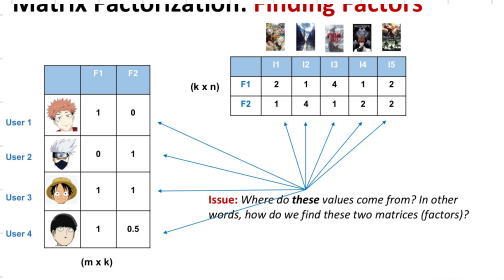
Matrix Factorization



CON
↳ Struggles with sparsity
↳ Suffers from cold start Problem

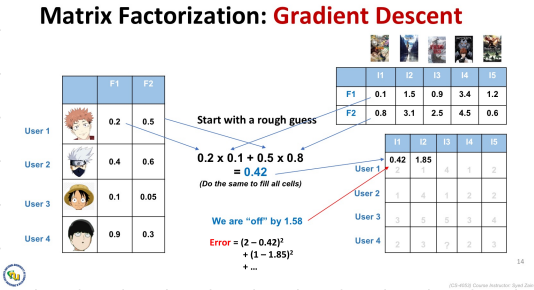
Finding Factors

1. Gradient Descent
2. Genetic Algorithm
3. Linear Programming
4. PSO

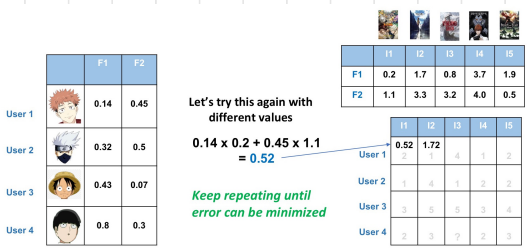


1. Gradient Descent

↳ quadratic



↓ Repeat



Steps

↳ initially put random values

↳ Error function

1. Sum of errors → used in diagram
2. Mean square error

↳ try to make overall error close to 0 → not individual → local optima

↳ ignore anomalies

↳ keep repeating till error minimised

PROS

↳ saves storage

↳ quick and easy predictions

CON

↳ very slow

↳ non deterministic → can't guess how many iterations

↳ expensive

↳ time consuming

↓ solution

use population algos

↳ genetic algo

Consider the given ***m*** x ***n*** interaction matrix:

5	3	5	3	3	1 → user 1
	2		3	3	4
5	3	1		5	
3		2	1		2

a) Which one the following values of ***k*** for the given factors *m* x *k* and *k* x *n,* will be providing the most efficient storage of interactions:

K = 1, 2, 4

b) Populate the factors 4x2 and 2x6 with some arbitrary values and calculate the mean squared error for *User 1* (represented by the first row of the original matrix).

a) $m=4, n=6 \rightarrow 4\times6=24$

$k=1 \rightarrow 4\times1+1\times6=10 \rightarrow 24-10=14$ → ANS → smaller k lesser complexity

$k=2 \rightarrow 4\times2+2\times6=20 \rightarrow 24-20=4$

$k=4 \rightarrow 4\times4+4\times6=40 \rightarrow 24-40=16$

↓ asked most efficient not which saves most storage

b)

1	0
3	2
1	3
2	0

0	2	4	1	0	0
1	1	3	0	2	0

0	2	4	1	0	0 → user 1
2	8	18	3	4	0
3	5	13	1	6	0
0	4	8	2	0	0

Sum of squared error = $(5-0)^2+(3-2)^2+(5-4)^2+(3-1)^2+(3-0)^2+(1-0)^2+$

= khud plus karo

i am not going ☺

For the following **3 x 5** interaction matrix, answer the questions given below:

	Item 1	Item 2	Item 3	Item 4	Item 5
User 1	3	1	1	3	1
User 2	1	2	?	1	3
User 3	4	3	5	4	4

a) Do these two matrix factors correctly represent the original interaction matrix? Provide a reason.

	F1	F2
U1	1	0
U2	0	1
U3	0	1

	I1	I2	I3	I4	I5
F1	3	1	1	3	1
F2	1	2	4	1	3

b) Use the matrix factors provided in Part (a) to predict missing rating ***R*(U2, I3)**.

c) How much storage space does the original interaction matrix require? Compare it with the total storage space required by the two given factors and write your comments about it.

d) Re-initialize the matrix factors in Part (a) with different arbitrary values in range [0.1—0.9] and calculate the sum of all squared errors.

a) **Answer:** No. The factor for User 3 does not truly represent the original interaction matrix e.g., for Item 1, the dot product U3 . I1 gives us (0x3 + 1 x1) = 1 instead of the original rating 4.

b) **Answer:** (0x1 + 1x4) = 4.

c) **Answer:** The original interaction matrix would take 3x5 = 15 units of storage, while the two factors would take (3x2) + (2x5) = 16 units of storage. Although, for this particular example the original interaction matrix is taking up less storage than its factors, when the size of the matrix is large, there will be a significant decrease in the storage requirements for the factored matrices compared to the original matrix.

d) Initialized matrix:

	Item 1	Item 2	Item 3	Item 4	Item 5
User 1	0.9	0.7	0.4	0.8	0.3
User 2	0.5	0.6	0.2	0.1	0.7
User 3	0.5	0.6	0.2	0.1	0.7

Sum of Squared Error:

$(3-0.9)^2+(1.0.7)^2+(1-0.4)^2+(3-0.8)^2+(1-0.3)^2+(1-0.5)^2+(2-0.6)^2+(1-0.1)^2+(3-0.7)^2+(4-0.5)^2+(3-0.6)^2+(5-0.2)^2+(4-0.1)^2+(4-0.7)^2 = 85.56$

[Estimated Time: 25 mins, 20 points (5+5+5+5)]

Question 3: Consider the given 7x6 interaction matrix ***R*** and its factors ***U*** and ***V***.

5	5	5	0	0	0
5	4	5	0	0	0
1	5	2	0	0	0
5	5	3	5	5	5
-1	-4	-5	1	1	1
-2	-1	-1	1	1	1
-3	-5	-1	1	1	1

5	1
-2	1
-4	0
1	0
0	1
0	0
0	0

1	-3	-2	-5	-1	-5
3	-2	0	1	0	-6

Answer the following questions:

a) If ***r*** and ***c*** represent rows and columns, do the factors contain valid values for original interactions ***R* (r1, c2)** and ***R* (r3, c3)**?
b) The given system allows negative ratings *[-5 to -1]*, no interaction *[0]* and positive ratings *[1 to 5]*. What is the major redundancy in this approach?
c) Will using gradient descent algorithm with a learning rate of 0.0005 for optimizing the given factors be a suitable approach? Justify your answer with a reason.
d) Name one major disadvantage of Matrix Factorization technique.

a) $R(r1, c2) = 5 \rightarrow (5,1), (-3,-2)$
$= (5)(-3) + (1)(-2)$
$= -17$
False

$R(r3, c3) = 2 \rightarrow (-4,0)(-2,0)$
$= (-4)(-2) + (0)(0)$
$= -8$
False

b) Major redundancy is attempting to capture all interaction types with only 2 factors, potentially overfitting

c) a learning rate of 0.0005 is very small, so slow convergence but it also helps avoid large updates, that leads to instabolity so suitable approach

d) struggles with sparsity when there are few observed interactions

Question 4: Matrix Factorization (CLO: 3) **10 points** (2.5 x 4)

Consider the following matrix for anime recommendation where 1 and -1 represent *like* and *dislike*, respectively. Answer the given questions:

	Bleach	Berserk	Erased	Akira
User 1	1	?	1	-1
User 2	-1	1	1	1
User 3	?	?	-1	-1
User 4	-1	1	1	?

a) Is it possible to make more than three factors of this matrix? Defend your answer with a proof.
b) For what values of *m, k, l* and *n*, will the three factors (*m* x *k*), (*k* x *l*) and (*l* x *n*) consume less memory in total than the original interaction matrix?
c) We used interactions instead of ratings in the given matrix. Can this approach help gradient descent converge quicker?
d) Does matrix factorization suffer from cold-start problem as severely as Collaborative Filtering? Justify your answer.

a) yes, as we can make any amount of factors

b) $(m \times k)$ $(k \times l)$ $(l \times n)$
(4×2) (2×1) (1×4)

og matrix = $4 \times 4 = 16$
factors = $4 \times 2 + 2 \times 1 + 1 \times 4 = 14$

c) yes, as interactions are less complex than a range of ratings

d) yes, as if there's no training data no factors can be generated

NEURAL NETWORKS

Derivatives

↳ rate of change w.r.t to something

eg

$f(x) = 2x$

$f'(x) = 2$

Partial Derivatives

↳ everything else is constant

↳ one thing changes

eg

$f(x, y, z) = 2x + 3y + 5z$

$f'(x) = 2$

$f'(y) = 3$

$f'(z) = 5$

Constraints

$z < 5, \quad x \leq 2, \quad y \leq 6$

easy eq so done manually

$f(x, y, z) = 2(2) + 3(6) + 5(4)$

$= 42$

for harder eq use gradient descent

gradient

↳ vector of partial derivatives

↳ overall change w.r.t every variable

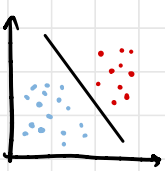
$y = mx + b \longrightarrow$ Linear eq

Linearly seperable data

Problem →

↳ lines can seperate data

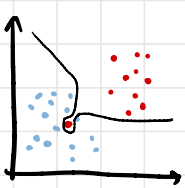
↳ not possible in real life



Non Linearity

→ Activation

↳ converts line into any shape



Large Regressor models

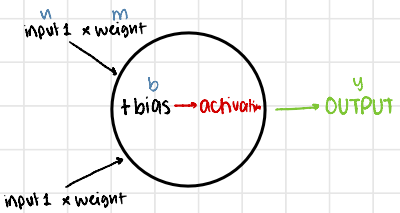
↳ neural networks that have

↳ they are not used for classification

Neuron

↳ takes no. of inputs and gives an output

$f(mn + b)$ ↓ if we ignore activation



Neural Network

↳ has layers of neurons

↳ output of 1 neuron is input to another → virtual connections

Activation

↳ adds non linearity to the output of a neuron

↳ helps decide wether neuron should be activated or not

Neural Network: Why is it called a Network?

- A single neuron doesn't really help us find complex patterns *(and is not cool enough!)*
- We need a network of inter-connected neurons to make complex decisions
- The output of one neuron becomes the input of another neuron

Neural Network Architecture

↳ Its composed of layers of neurons

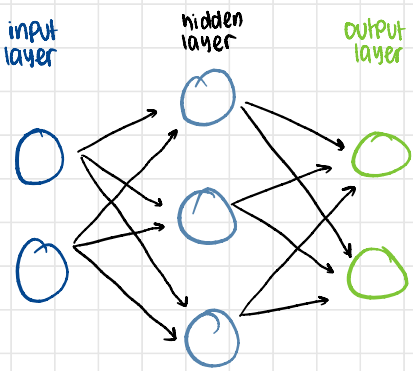
↳ the more hidden layers the better

↳ more expensive ~~cons~~

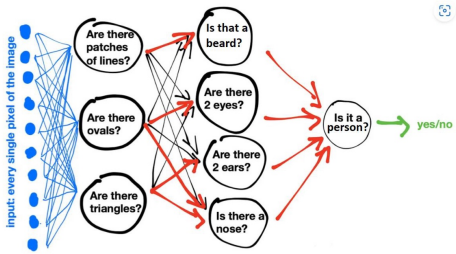
↳ can sometimes bring down accuracy

↳ It is a fully connected graph

↳ They are unidirection → only go one way



Neural Network: How does it work?

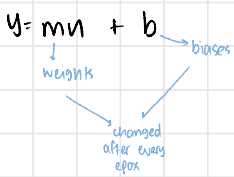


28

- features are abstract
- the layer closer to hidden layer has more detailed questions

Training

1. Feed raw input (features) to the input layer
2. Initialize all the weights and biases for hidden layers with random values
3. Test if the network can accurately produce the output
   I. If it does not produce accurate results then adjust the weights and biases. It simply means we want to use optimization (e.g. gradient descent) to minimize the value of our loss function. Repeat **Step 3**
   II. If it does produce accurate results then terminate training



↳ train till you
1. reach a certain accuracy
2. reach a certain epox

*To "adjust" the parameters (weights and biases) we use **backpropagation***

- ❑ When the neural network outputs the wrong answer, you find the slopes (derivative) of the output layer first because it was the direct cause of the incorrect answer
- ❑ Since the output layer depends on the hidden layer, you'll have to fix that too by finding the slopes and using gradient descent
- ❑ Eventually you'll work your way back (backpropagate) to the first hidden layer

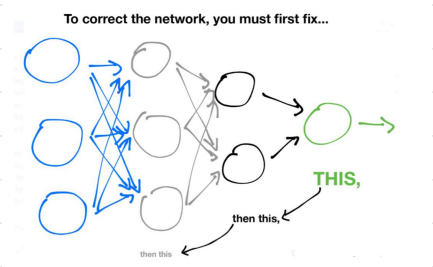
32

Backpropogation

↳ To correct the network, you must fix the error towards the input layer → till the 1st hidden layer

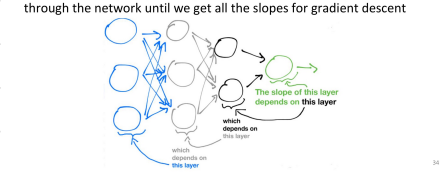
↳ hidden layers never have any error

↳ start weights and biases with random values



$$\text{Error} = \frac{1}{2}(\text{prediction} - \text{actual})^2$$

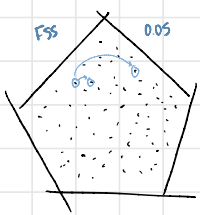
❑ We calculate slopes by starting from the back and moving backwards through the network until we get all the slopes for gradient descent



Exploration aka learning curve

↳ jumps to other region

con
↳ if high exploration might jump and skip solution



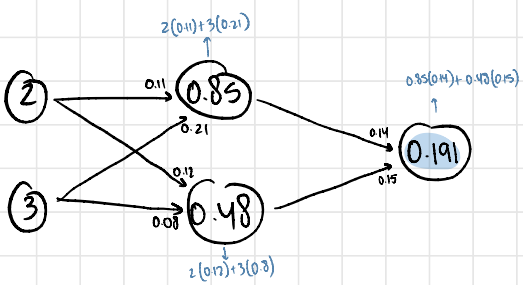
Exploitation

↳ trying near regions / neighbourhood

8) w1: 0.11, w2: 0.21, w3: 0.12, w4: 0.08, w5: 0.14, w6: 0.15

Our target output is 1

↳ FORWARD PASS



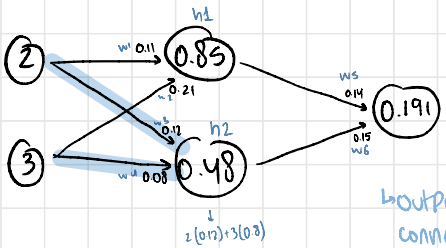
error is always +ve because of this square

$$\text{Error} = \frac{1}{2}(0.191 - 1)^2 = 0.327$$

↳ BACKWARDS PASS → to reduce the error

Gradient Descent

↳ Update the value of weights



↳ output only knows connected weights

Prediction: $\underbrace{(i_1 w_1 + i_2 w_2)}_{h1} w_5 + \underbrace{(i_1 w_3 + i_2 w_4)}_{h2} w_6$

Error: $\frac{1}{2}(\text{Prediction} - \text{actual})^2$

Δ: Prediction − actual → delta

❑ The derivative of our loss can be evaluated by using Chain rule:

$$\frac{\partial Error}{\partial W_6} = \frac{\partial Error}{\partial prediction} * \frac{\partial prediction}{\partial W_6}$$ ← chain rule

Error = $\frac{1}{2}(prediction - actual)^2$

prediction = $(i_1 w_1 + i_2 w_2) w_5 + (i_1 w_3 + i_2 w_4) w_6$ (h1, h2)

$$\frac{\partial Error}{\partial W_6} = \frac{\frac{1}{2}(predictoin - actula)^2}{\partial prediciton} * \frac{\partial (i_1 w_1 + i_2 w_2) w_5 + (i_1 w_3 + i_2 w_4) w_6}{\partial W_6}$$

$$\frac{\partial Error}{\partial W_6} = 2 * \frac{1}{2}(predictoin - actula) \frac{\partial (predictoin - actula)}{\partial prediciton} * (i_1 w_3 + i_2 w_4)$$ ← $h_2 = i_1 w_3 + i_2 w_4$

$$\frac{\partial Error}{\partial W_6} = (predictoin - actula) * (h_2)$$

Δ = prediction − actual ← delta

$$\frac{\partial Error}{\partial W_6} = \Delta h_2$$

↳ $\Delta = 0.191 - 1 = -0.809$

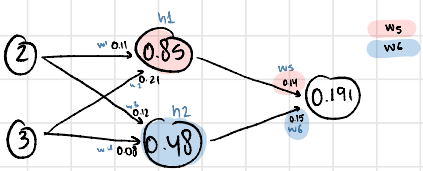
$\alpha = 0.05$ → learning rate → guess it

↳ Updating w_6

* $w_6 = w_6 - \alpha \Delta h_2$
  - $0.15 - 0.05(-0.809)(0.48)$
  - $0.169 \simeq 0.17$

↳ Updating w_5

* $w_5 = w_5 - \alpha \Delta h_1$
  - $0.14 - 0.05(-0.809)(0.85)$
  - $0.174 \simeq 0.17$



❑ In order to update **w1** (existing between input layer and hidden layer):

$$\frac{\partial Error}{\partial W_1} = \frac{\partial Error}{\partial prediction} * \frac{\partial prediction}{\partial h_1} * \frac{\partial h_1}{\partial W_1} \quad \leftarrow \text{chain rule}$$

$$\text{Error} = \frac{1}{2}(\text{prediction} - \text{actual})^2$$

$$\text{prediction} = (h_1)\, w_5 + (h_2)\, w_6$$

$$h_1 = i_1 w_1 + i_2 w_2$$

$$\frac{\partial Error}{\partial W_1} = \frac{\partial \frac{1}{2}(predictoin - actula)^2}{\partial prediciton} * \frac{\partial\,(h_1)\, w_5 + (h_2)\, w_6}{\partial h_1} * \frac{\partial\, i_1 w_1 + i_2 w_2}{\partial w_1}$$

$$\frac{\partial Error}{\partial W_1} = 2 * \frac{1}{2}(\text{predictoin} - \text{actula}) \frac{\partial (predictoin - actula)}{\partial prediciton} * (w_5) * (i_1)$$

$$\frac{\partial Error}{\partial W_1} = (\text{predictoin} - \text{actula}) * (w_5 i_1)$$

$$\Delta = \text{prediction} - \text{actual} \quad \leftarrow \text{delta}$$

$$\frac{\partial Error}{\partial W_1} = \Delta\, w_5 i_1$$

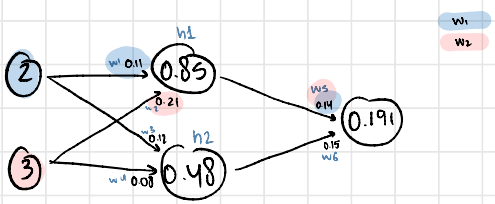
4

Updating W1

* $W_1 = W_1 - \alpha\, \Delta (W_5 \cdot i_1)$
  $= 0.11 - 0.05(-0.809)(0.14)(2)$
  $= 0.121$

Updating W_2

* $W_2 = W_2 - \alpha\, \Delta (W_5 \cdot i_1)$
  $= 0.21 - 0.05(-0.809)(0.14)(3)$
  $= 0.227 \simeq 0.23$

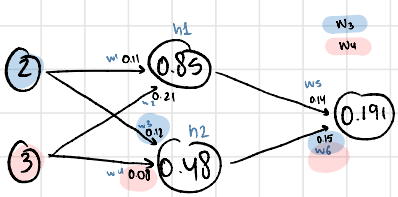


Updating W_3

* $W_3 = W_3 - \alpha\, \Delta (W_6 \cdot i_1)$
  $= 0.12 - 0.05(-0.809)(0.15)(2)$
  $= 0.132 \simeq 0.13$

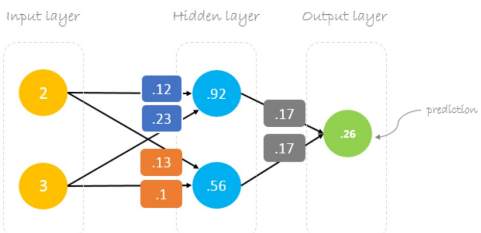
Updating W_4

* $W_4 = W_4 - \alpha \cdot \Delta (W_6 \cdot i_2)$
  $= 0.08 - 0.05(-0.809)(0.15)(3)$
  $= 0.098 \simeq 0.10$



- All the updated weights *(rounded off to 2 digits)* are as follows:
  - **w1 = 0.12**
  - **w2 = 0.23**
  - **w3 = 0.13**
  - **w4 = 0.10**
  - **w5 = 0.17**
  - **w6 = 0.17**
- **Task:** Re-calculate and verify the updated values of these weights

- Now, using the new weights we will repeat the forward pass



- Notice that our prediction is slightly better but still not perfect. Therefore, keep updating the weights and performing backward-forward passes until the error is minimized

Are we still missing something?

- Bias
- Activation
  - Sigmoid *(for binary classification)*
  - ReLU *(suffers from vanishing gradient problem)*
  - LeakyReLU *(improved ReLU)*
  - Softmax *(for multi-class classification)*
- Regularization
- Dropout
- Optimization
  - Stochastic Gradient Descent
  - Batch and Mini-batch Gradient Descent
  - Momentum
  - Adam

Question 6: Neural Recommendations (CLO: 4) **10 points** (5 + 5)

a) Explain why the following ratings data cannot be properly split into training and testing samples for neural recommendations:

	I1	*I2*	*I3*	*I4*	*I5*	*I6*	*I7*	*I8*	*I9*	*I10*
U1	2	1		3	5	4		2	1	2
U2										
U3	4	5		3	2	4			5	1
U3		1	4	3	4	2		1	1	5
U5	4	2		1	3	2		1	2	3
U6	3			1		2		3	2	2

Note: *Each row represents a unique user and each column is a unique item.*

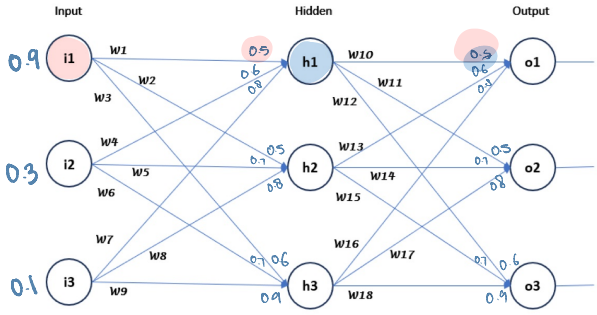
b) Explain why we are usually interested in interactions more than ratings when working with recommendations generated by a GAN.

a) as it is a sparse matrix
- ↳ user 2 didn't rate any item
- ↳ item 7 has no ratings by any user

So we cant split such data

b) GANs are extremelly sensitive
providing ratings is much more complex to converge
hence we use interactions (0 or 1)

Consider the neural network given below:



W10
W1

Item features:	0.9	0.3	0.1

Initial weights are **w1** = **w2** = **w10** = **w11** = 0.5; **w3** = **w4** = **w12** = **w13** = 0.6; **w5** = **w6** = **w14** = **w15** = 0.7; **w7** = **w8** = **w16** = **w17** = 0.8; and **w9** = **w18** = 0.9. Assume that the true output of **o1** was **0.7**. Find the error at **o1** and run a backward pass to update w10.

$h1 = (0.9)(0.5) + (0.3)(0.6) + (0.1)(0.8) = 0.71$

$h2 = (0.9)(0.5) + (0.3)(0.7) + (0.1)(0.8) = 0.74$

$h3 = (0.9)(0.6) + (0.3)(0.7) + (0.1)(0.9) = 0.84$

$O1 = (0.71)(0.5) + (0.74)(0.6) + (0.84)(0.8) = 1.471$

$error = \frac{1}{2}(1.471 - 0.7)^2 = 0.297$

taking $\alpha = 0.05$

$\Delta = 1.471 - 0.7 = 0.77$

updating w_{10}

$*w_{10} = w_{10} - \alpha \Delta h_1$

$= 0.5 - 0.05(0.77)(0.71)$

$= 0.47$

updating w_1

$*w_1 = w_1 - \alpha(\Delta \cdot w_{10} \cdot i_1)$

$= 0.5 - 0.05(0.77)(0.5)(0.9)$

$= 0.483$

Question 5: Consider the given inputs and parametric values for a simple recommendation neural network. The following tables show data for three training samples. Each sample has distinct user and item features as the input. We can ignore the activations and hidden layer values to denote them simply as *a, b, c* and *d*. The output of the network is either 0 or 1 representing the predicted interaction of the particular user with the given item.

	I	H1	H2	O
User Features	1	a	c	1
Item Features	0.01	b	d	

I	H1	H2	O
0.99	a	c	1
0.005	b	d	

I	H1	H2	O
0.99	a	b	1
0.8	c	d	

Answer the following questions:

a) Based on the observed values for input features. Will it be a good idea to omit user features from the input and use only item features for classification? Justify your answer with a reason.
b) Given the variance in item features, is the model exhibiting over-fit or under-fit behavior?
c) What should be the ideal activation function in the output layer of this network?
d) For 1000 users, if we assume the variance in their features to be 0.08. What does it explain about the users present on the website/system?

a) both u and i features vary b/w samples
u features contributes unique info about users preference
which can enhance recommendations accuracy
so omitting u features is not a good idea as
it will lead a loss to critical info

b) all '0' values are 1 → consistent output
item features have low variance → low variance features
this pattern suggest overfitting to a specific item

c) since output predicts 0 and 1
a sigmoid activation function is recommended

d) It implies low diversity in user feature values → as low variance of 0.08
meaning users on the website mostly have same preferences

Consider the case where the data points are linearly separable. What will be the role of activations in a neural network architecture with an input layer, several hidden layers and an output layer?

If the data points in a given training set are already linearly separable, there isn't any need to induce non-linearity, hence there should be no role of activations in that case and a large regressor would suffice.

chp 9

knowledge Graphs based Systems

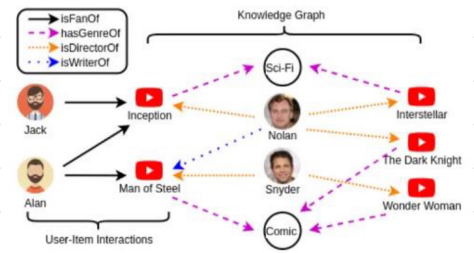
↳ similar to content based filtering

↳ uses graphs to represent users, items and their relationships

↳ uses a network of interconnected data

↳ nodes ⟶ reliance

↳ how entities connected in social network



* diff node represent diff things
  users, genres, directors, movies

* diff edges represent diff relations
  ⟶ ⇢ ⟶ ⇢

↳ Diversity

↳ KGs represents semantically rich relations b/w entities

↳ different aspects of user preference → director, → genre

↳ Scalability

↳ doesn't need sparse matrix to predict similarity

↳ can be vector based → as can define dimensions

↳ users preference are accumulated and stored in the graph structure

users → fav movie → fav director, fave genre

↳ Algos we can use for KG based recommendation

1. Embedding based → Neural Networks
2. Path based → Data Structures
3. RippleNet

knowledge Graph

1. Embedding based KG Recommendation

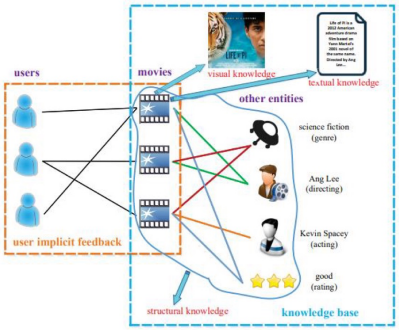
my movie is closer to which movie in KG

↳ movie synopsis
↳ actors

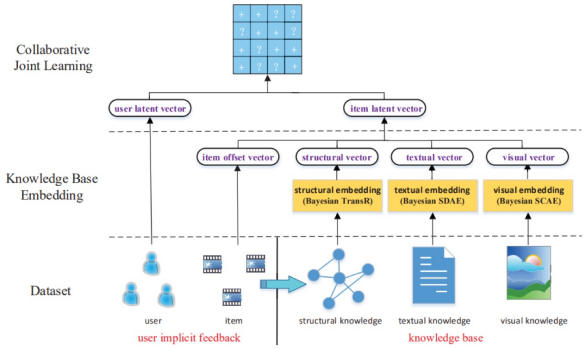
movie posters

- Pre-process KG using knowledge graph embedding algorithms
  - For *structural* knowledge: TransR embedding
  - For *textual* knowledge: Word2Vec, Stacked Denoising Autoencoder
  - For *visual* knowledge: Stacked Convolutional Autoencoder
- The generated embeddings become input to joint collaborative (content) filtering
- Use cosine similarity (or some other measure) for finding distance of **u** to $\mathbf{v}_i$ *(i=1, 2, …, n)*

Embedding-based KG Recommendation



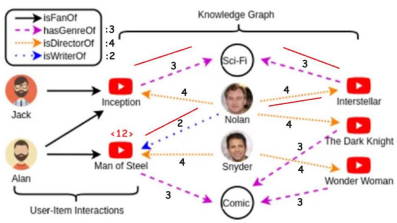
Embedding-based KG Recommendation



2. Pam based KG Recommendation

- For an active user, traverse a path from each of the *k* top-rated items of the user
- Each edge is assigned a feature score
- The path follows edges representing knowledge about the item
- Each edge is visit once and its feature score is added to total path score
- For each path of an unrated item:
  - It ends on an unrated item and its path score is recorded
  - The maximum path score across all paths become the item utility value
- The top-*k* items with maximum utility values are recommended

Path-based KG Recommendation



- Domain knowledge is used to assign feature scores
- The path score in this example is 12

Inception → Scifi → Interstellar → Nolan → Man of Steel

3 + 3 + 4 + 2 = 12

↳ each edge is visited once

item utility value

↳ for each pam of an unrated item

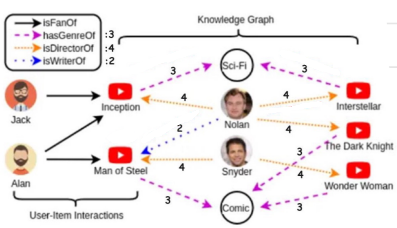
↳ ends on unrated item

↳ max pam score

↓

top k items with max utility is RECOMMENDED

Path-based KG Recommendation



inception → Nolan → wonder woman ← snyder ← man of steel

↓

recommended

Pam

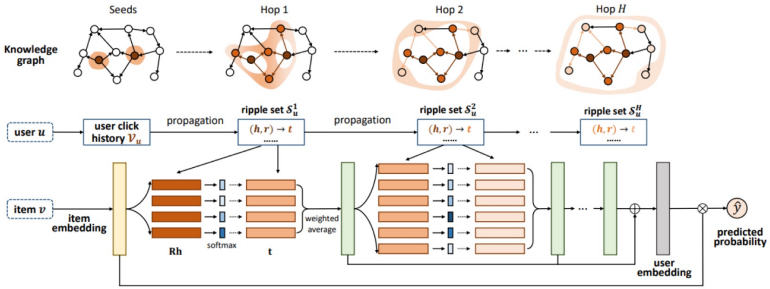
?

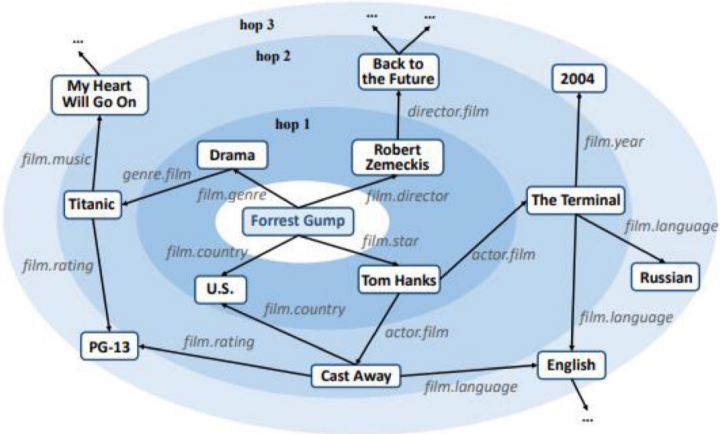
3. Ripple Net

- inputs
  - user u
  - item v
- outputs: predicted probability user u will click item v

- For the input user ***u***, his historical set of interests V_u is treated as seeds in the KG, then extended along links to form multiple ripple sets
- A ripple set is the set of knowledge triples that are k-hop(s) away from the seed set V_u. These ripple sets are used to interact with the item embedding iteratively for obtaining the responses of user ***u*** with respect to item ***v***, which are then combined to form the final user embedding
- Lastly, we use the embeddings of user ***u*** and item ***v*** together to compute the predicted probability

RippleNet





CHP 10

Generative Adversarial Networks (GANs)

↳ an unsupervised architecture

↳ uses 2 neural networks, competing against eachother → thrive the adversial

in order to generate new, synthetic instances of data → that can pass for real data

↳ They can learn to mimic any distribution of data

↳ can be used for

- ↳ image generation
- ↳ voice generation
- ↳ video generation

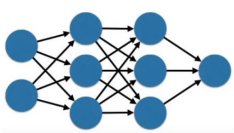
Image Generation

- This Person Does Not Exist - Random Face Generator (this-person-does-not-exist.com)

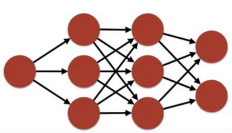
GAN architecture

↳ has 2 networks

- ↳ generator
- ↳ discriminator



Generator



Discriminator

Generator

Discriminator

Adversial Learning

↳ Its a 2 player game b/w the generator and discriminator

The Generator

↳ starts with a random noise (reasonable random noise)

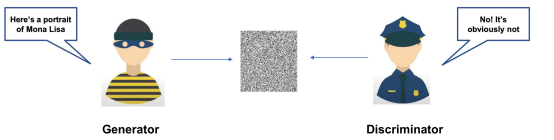
↳ tries to mimic the target distribution

e.g. • image of digit
• image of mona lisa

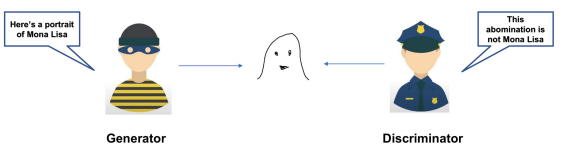
The Discriminator

↳ has already seen the training data so knows the target distribution

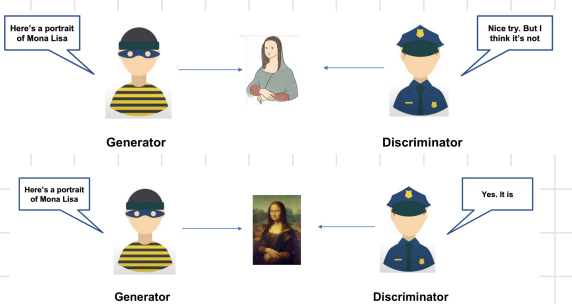
↳ so acts like a trainer and judger

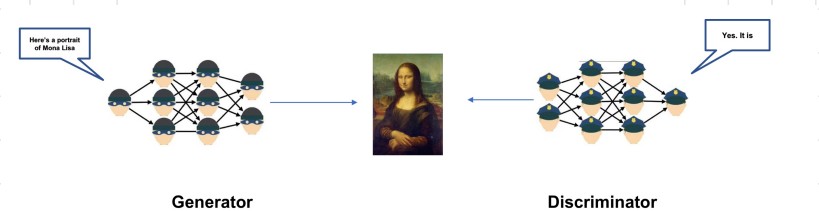


↳ It repeatedly tries to fool the discriminator that the generated distribution (image) is a real one



↳ based on feedback (loss) of the discriminator the quality of the generated output is gradually improved





STEPS TAKEN BY GANS

↳ The generator takes in random numbers and returns an image

↳ The discriminator is fed
  ↳ this generated image
  ↳ a stream of images ← taken from the actual ground truth dataset

↳ It takes both real and fake images and returns probabilities representing ← a no b/w 0 and 1
  ↳ 1 → a prediction of authenticity
  ↳ 0 → fake

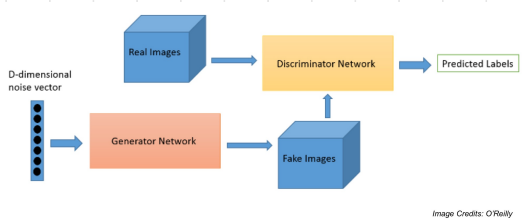


Image Credits: O'Reilly

$$\min_G \max_D V(D,G)$$

$$V(D,G) = \mathbb{E}_{x \sim p_{data}(x)}[\log D(x)] + \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z))]$$

Example: Generating MNIST digits

Generator

↳ an inverse CNN

↳ input → vector of random noise

↳ output → upsamples it to an image

Discriminator

↳ a standard CN

↳ categorises images fed to it

↳ always a binary classifier, labels images
  ↳ real
  ↳ fake

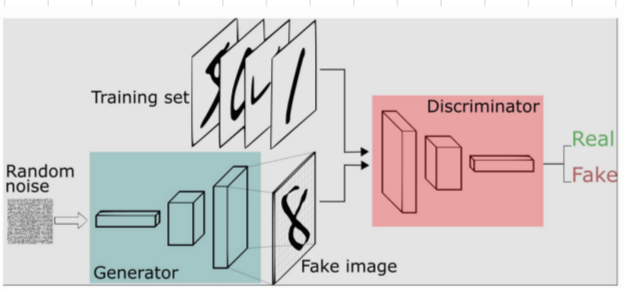
Steps

1. show MNIST dataset to discriminator
   ↳ never train discriminator full dataset
   ↳ on 30 to 35% dataset shown
2. Train discriminator as a classifier
3. Generator will take noise ← reasonable random noise and produce image
   It will have some loss that discriminator will use

MNIST dataset

↳ handwritten dataset of digits

0 – 9



How to train a GAN

- ❑ Build your generator and discriminator models
- ❑ Sample data from true distribution and train discriminator on this data
  - ❑ Discriminator works as a binary classifier
- ❑ Create input data from latent space (noise) and feed it to the generator
- ❑ The generator output serves as input to the discriminator
- ❑ The discriminator uses Sigmoid activation to classify the generated output as real or fake
- ❑ The discriminator loss is fed-back to the generator
- ❑ The generator is trained until it reaches convergence

CON

1. GANs are hard to train → as convergence hard to achive
2. Hard to meet convergence criteria → how realistic do I want it to be
3. Vanilla GANs suffer from vanishing gradient Problem (doesn't take cross loss) → gradients start becoming 0 so then training can't be done → as it uses feedback of loss
4. Overwhelming adversary → discriminator shouldn't be over powered → unless very realistic wont accept it / discriminator shouldn't be weak → unrealistic images accepted
   - ↳ one network should not overpower another
5. Can not count objects → cant say make 5 Mona Lisa Paintings → made why cant draw fingers properly
6. Sensitive to choice of hyperparameters → no optimal hyperparameter
7. ... and many other challanges

Guidelines for training

- ↳ use tanh activation for generator
- ↳ use sigmoid for discriminator → as 0 fake 1 real
- ↳ freeze generator for every other training iteration of discriminator
- ↳ use LeakyReLU in dense layers
- ↳ Use a Functional API → for keras
- ↳ use drop out
  - ↘ randomly switches off some neurons during training → so no overfitting

Types of GANs

- Vanilla GANs → OG
- CGAN → Conditional GAN → uses class label as input
- Wasserstein GAN
- Style GAN → input product, output related product → Pink shirt → Pink hat
- Info GAN
- DCGAN
- Dual GAN
- KDGAN
- ⋮

GANs in RS

- used in RS to generate ratings
- solves cold start problem
- slow training but prediction are fast
- CGAN
- Style GAN

CHP 11

Basic Design Principles

↳ for recommendation output is always a 1D vector

↳ GANs pick one

- ❑ We never work on **ratings** *i.e.*, these ratings are to be transformed to **interactions**
- ❑ We always use a supervised variant of GAN for recommender system *e.g.*, cGAN, cWGAN, IRGAN
- ❑ We can do either user-to-item generation or item-to-user generation
- ❑ Always watch out for a trivial solution *(network learning to output all zeros or ones)*

Example: MIND News Recommendation

↳ a news dataset with

↳ impression logs ← +ve response → 1, -ve response → 0

↳ historical choices for users and news items

	ImpressionID	UserID	Time	History	Impressions
0	1	U13740	11/11/2019 9:05:58 AM	N55189 N42782 N34694 N45794 N18445 N63302 N104...	N55689-1 N35729-0
1	2	U91836	11/12/2019 6:11:30 PM	N31739 N6072 N63045 N23979 N35656 N43353 N8129...	N20678-0 N39317-0 N58114-0 N20495-0 N42977-0 N...
2	3	U73700	11/14/2019 7:01:48 AM	N10732 N25792 N7563 N21087 N41087 N5445 N60384...	N50014-0 N23877-0 N35389-0 N49712-0 N16844-0 N...
3	4	U34670	11/11/2019 5:28:05 AM	N45729 N2203 N871 N53880 N41375 N43142 N33013 ...	N35729-0 N33632-0 N49685-1 N27581-0
4	5	U8125	11/12/2019 4:11:21 PM	N10078 N56514 N14904 N33740	N39985-0 N36050-0 N16096-0 N8400-1 N22407-0 N6...

N27581-0 → User saw but didn't click → ignored

N6... → User saw and clicked → historical

Construct Training data

↳ create training data by treating

↳ historical choices → positive interaction → 1

↳ ignored new → negative interaction → −1 ← change 0 to −1

	UserID	NewsID	Interaction
78833	U46778	N17157	1
78833	U46778	N28081	1
78833	U46778	N58641	1
78833	U46778	N3926	1
78833	U46778	N16290	1

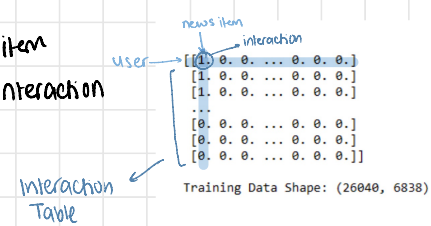
	UserID	NewsID	Interaction
78833	U46778	N60456	-1
78833	U46778	N19592	-1
78833	U46778	N35738	1
78833	U46778	N3123	-1
78833	U46778	N12029	-1

→ change data so it's suitable for GANs

↳ row → user

↳ colum → news item

↳ intersection → interaction



Interaction Table

-1 → saw, not intrested
0 → didn't see
1 → saw, intrested

NO labels

since user features are not given and we need class labels

So do clustering on interest of user → news features → what type of news does user like

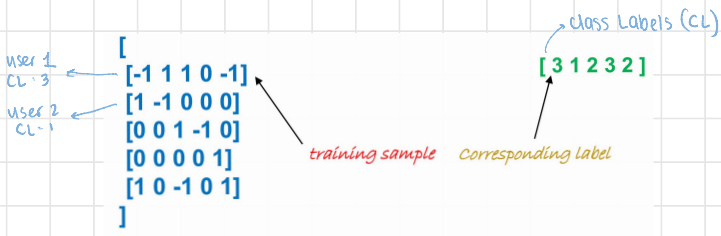
↳ every cluster is a class label

We often do not have any labelled recommendation data hence we can use clustering or some other annotation technique to create our label set

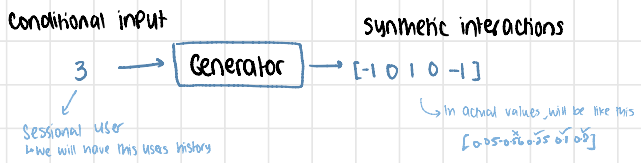
	NewsID	Category	Subcategory	Title	Abstract	Processed_Text	Embeddings	Label
2	N61837	news	newsworld	The Cost of Trump's Aid Freeze in the Trenches...	Lt. Ivan Molchanets peeked over a parapet of s...	news newsworld cost trump aid freeze trench uk...	[-0.05749018, -0.05521024, -0.016900085, 0.030...	5
3	N53526	health	voices	I Was An NBA Wife. Here's How It Affected My M...	I felt like I was a fraud, and being an NBA wi...	health voice nba wife here affected mental hea...	[-0.03127052, -0.029868228, -0.028364867, 0.05...	2
4	N38324	health	medical	How to Get Rid of Skin Tags, According to a De...	They seem harmless, but there's a very good re...	health medical get rid skin tag according derm...	[-0.07352022, -0.0478042, -0.007855178, 0.0447...	1
5	N2073	sports	football_nfl	Should NFL be able to fine players for critici...	Several fines came down against NFL players fo...	sport footballnfl nfl able fine player critici...	[0.018641314, -0.004955198, -0.01916305, 0.091...	3
10	N9721	health	nutrition	50 Foods You Should Never Eat, According to He...	This is so depressing.	health nutrition food never eat according heal...	[-0.05535877, -0.080809906, -0.044505723, 0.01...	9

GAN Training

↳ we can train it in the following manner



Example: Generation Recommendation



* if cold start user
  ↳ forward it to news predictor

ishma hafeez notes