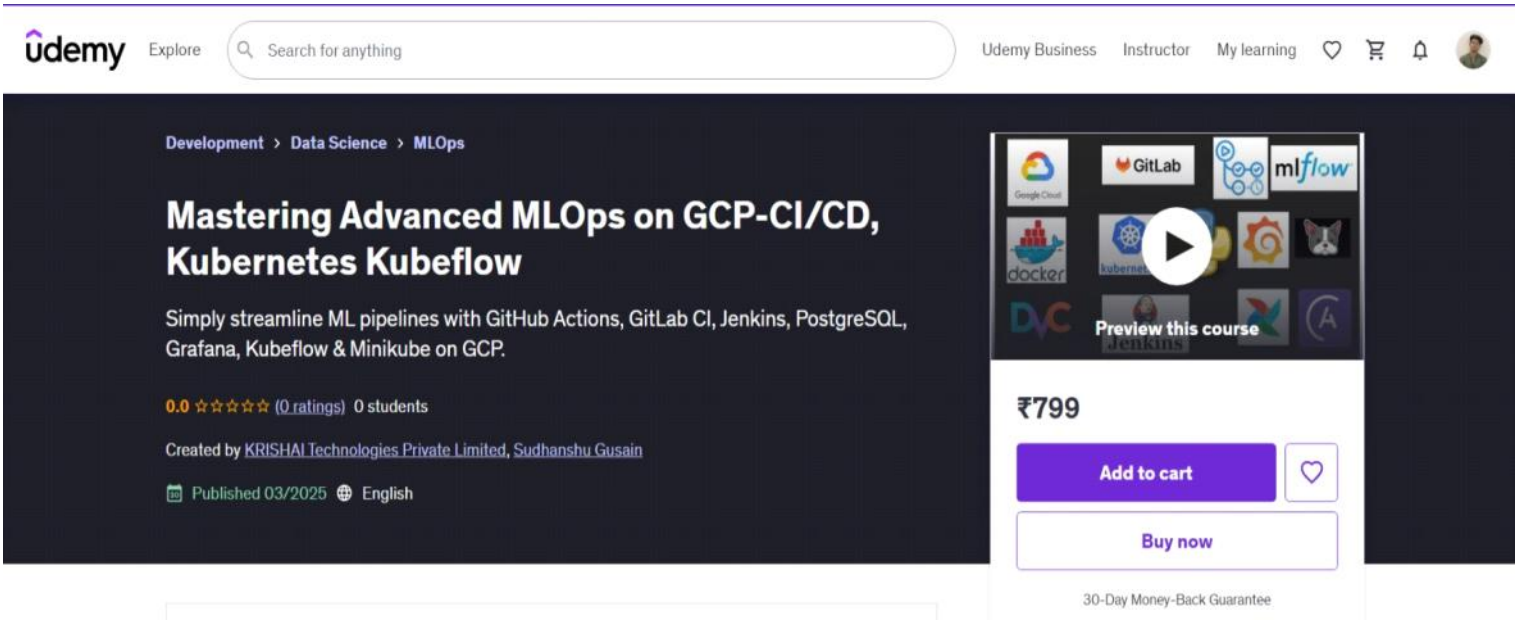
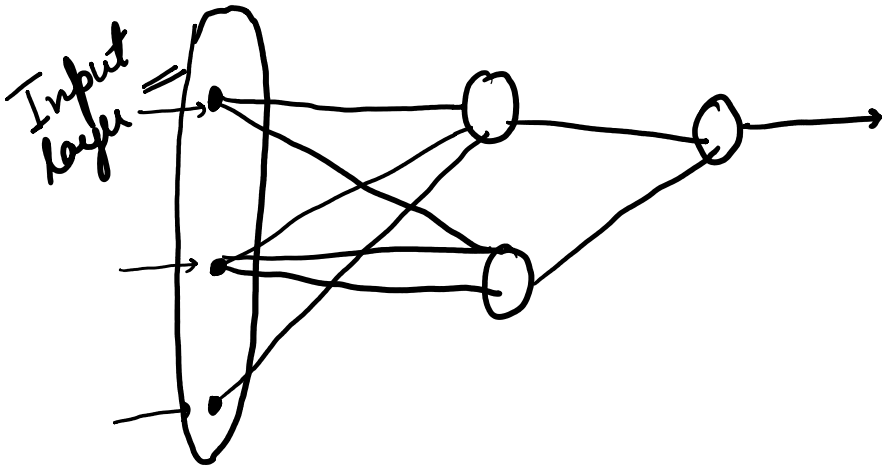
WHAT YOU WILL STUDY IN TODAY VIDEO ?

- ► All Types of Layers used in ANN neural networks
- ► Their Use , Mathematical Working and Syntax


udemy
Explore
Search for anything
Udemy Business
Instructor
My learning
Development › Data Science › MLOps
Mastering Advanced MLOps on GCP-CI/CD, Kubernetes Kubeflow
Simply streamline ML pipelines with GitHub Actions, GitLab CI, Jenkins, PostgreSQL, Grafana, Kubeflow & Minikube on GCP.
0.0 ☆☆☆☆☆ (0 ratings) 0 students
Created by KRISHAI Technologies Private Limited, Sudhanshu Gusain
Published 03/2025 English
GitLab
mlflow
docker
Preview this course
₹799
Add to cart
Buy now
30-Day Money-Back Guarantee

① Input

↳ Take inputs for your model



* $X = [x_1, x_2, x_3, x_4 \ldots]$
  Age, Grade, Section

CNN ⇒ pixel values
RNN ⇒ word embedding
} later

CAT → [0 1 0]

(Only one in a neural network) *

most used

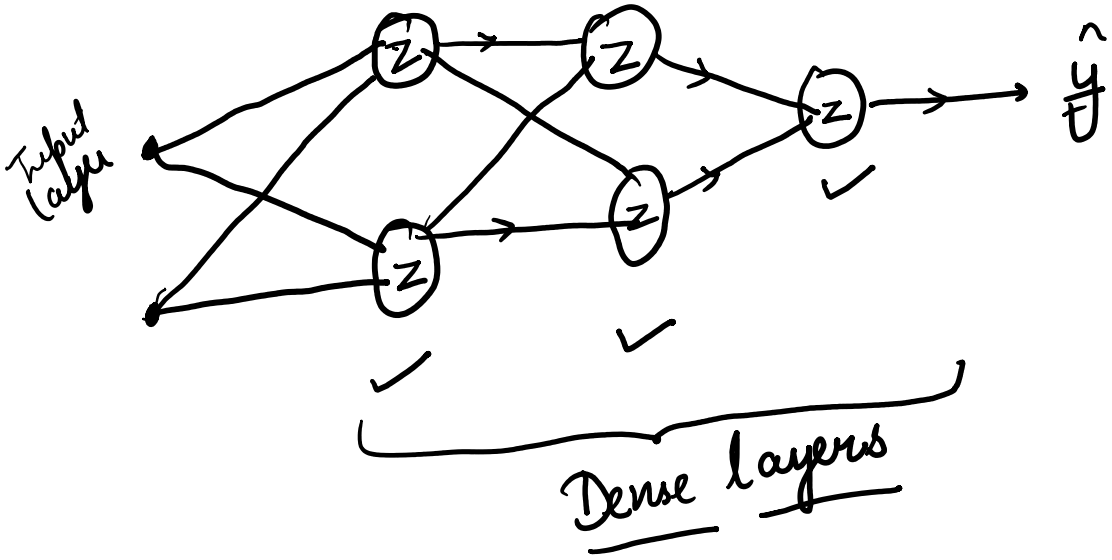
= Dense(64, input_shape = (10,))

= Input(=)

= Input (=)

② Dense

- fully connected layers
- All weights, biases, Z, activation fn works here.



* $Z = WX + b$

weighted sum

$A = \sigma(Z)$

↳ o/p of that neuron

* Common

$\text{Dense}(64, \text{activation} = \text{'relu'})$

↓

no of

no of neurons

③ Activation Layer

Dense (64)
Activation ('relu')

④ Dropout Layer

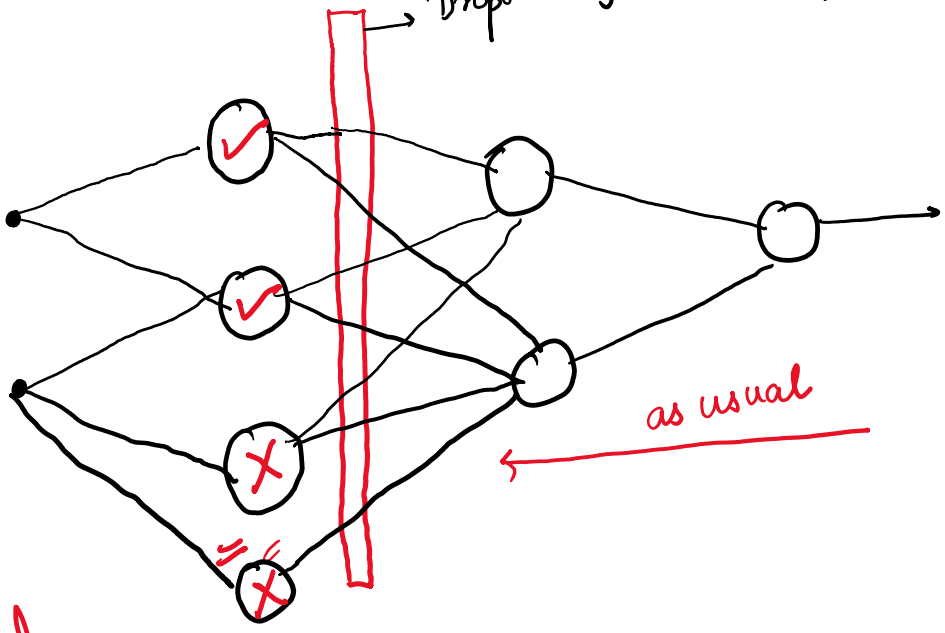
→ drops neurons randomly

↓

Prevent overfitting

During Backward propagation =

Dropout layer → dropout rate → 0.30



Neuron = 4
dr = 50%
= 2.

Mathematical

① Before Dropout → Single layer network 4 neurons

relu = max(0, z)

Neuron	(X) Input	(W) Weight	(b) Bias	Z	Output
1	2	0.5	1	2	2
2	3	-0.3	1	0.1	0.1
3	1	0.8	-1	-0.2	0
4	4	-0.5	2	0	0

Output 1 = [2, 0.1, 0, 0]

dropout rate = 0.5 (50%)

② With Dropout layer

Keep (1) or Drop (0)

Neuron	Activated O/P	Dropout layer	final O/P
1	2	1 (keep)	2 active
2	0.1	0 (Drop)	0 non-active
3	0	1 (keep)	0 active
4	0	0 (Drop)	0 non-active

Output 2 = [2, 0, 0, 0]

After dropout

* Compensating for non-active neurons

$$\text{formula} = \frac{1}{1-p}$$

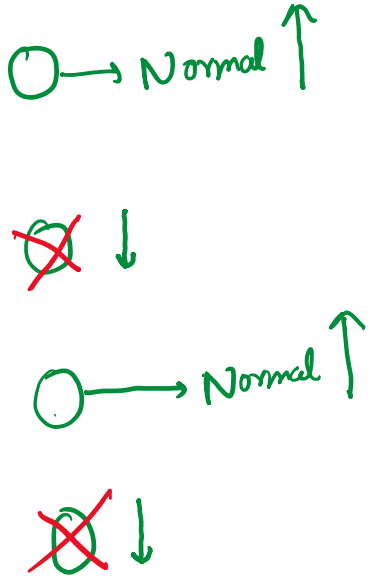
p → dropout rate (0.5)

$$\Rightarrow \frac{1}{1-0.5} = 2.$$

$\Rightarrow \frac{1}{1-0.5}$

* Scale active outputs

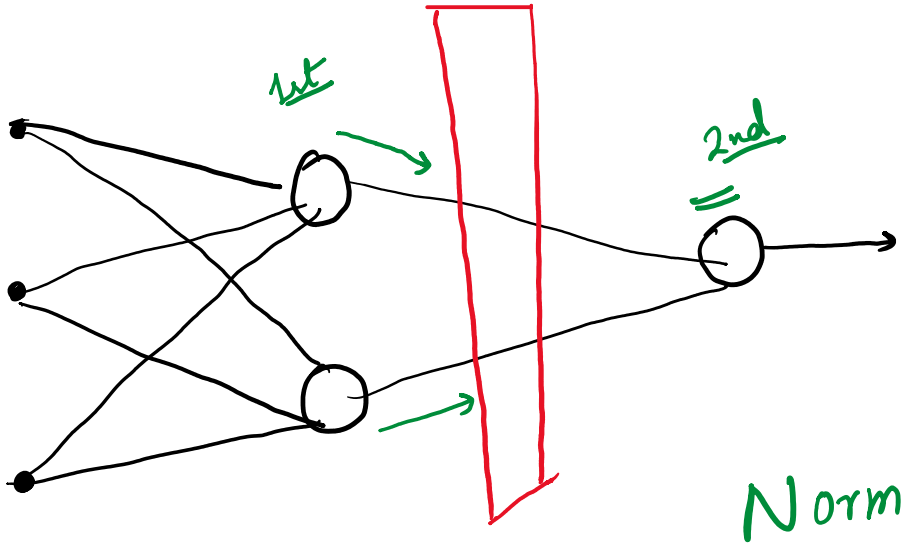
Neuron	Output	Scaled O/P
1	2	2 × 2 = 4
2	0	0 × 2 = 0
3	0	0 × 2 = 0
4	0	0 × 2 = 0



* Dense(64, activation = 'relu')
Dropout(0.5)
= ↳ drop out rate

⑤ Batch Normalization

- Normalizes the O/P of neuron
- Model training speed ↑
  Stable training
  faster convergence



100 - 1000

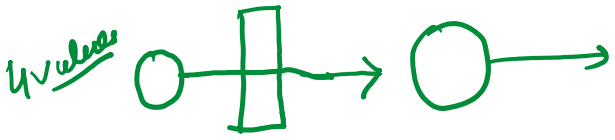
↓

0 - 10.

Normalization?

- Mean ≈ 0
- Variance ≈ 1

Mathematical



$X = [5, 10, 15, 20]$

→ values are a bit large

① Compute mean

$$\mu = \frac{5+10+15+20}{4} = 12.5$$

$\mu = \frac{}{4}$

② Compute variance.

$$\sigma^2 = \frac{(5-12.5)^2 + (10-12.5)^2 + (15-12.5)^2 + (20-12.5)^2}{4}$$

$$\sigma^2 = 31.25.$$

③ Normalize each input

$$X_{norm} = \frac{X - \mu}{\sqrt{\sigma^2 + \epsilon}} \quad \epsilon = 10^{-5}$$

$$X_{norm} = \frac{[5, 10, 15, 20] - 12.5}{\sqrt{31.25 + 10^{-5}}}$$

$X_{norm} = [-1.34, -0.45, 0.45, 1.34]$ ← {Mean = 0, Variance = 1}

= Normalized values.

④ Scale & Shift.

Scale factor (γ)

Shift factor (β)

factor (γ)

ν (β)

Let, take,

$$\gamma = 2$$
$$\beta = 3$$

$$X_{new} = \gamma \times X_{norm} + \beta$$

$$X_{new} = 2 \times [-1.34, -0.45, 0.45, 1.34] + 3$$

$$= [0.32, 2.1, 3.9, 5.68]$$

These values will be passed

* Dense (64, activation = 'relu')
  Batch Normalization ()
  Dense (64, activation = 'relu')

⑥ Flatten layer

- Convert multi-dimensional to 1-D input
- Mostly used in CNN

* **Before.**

2-D matrix

$$X = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

Shape = (2, 3)

2-D array

Convert →

After.

$$X = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{bmatrix}$$

Shape = (6, 1)

1-D array

*

High dim input

↓

flatten()

↓

1-D output

↓

Dense(- -)

Denise L

⑦ Reshape Layer

→ Changes shape of a tensor / array

Dense layer (64, 1) → Reshape layer (8, 8) → Dense (8, 8)

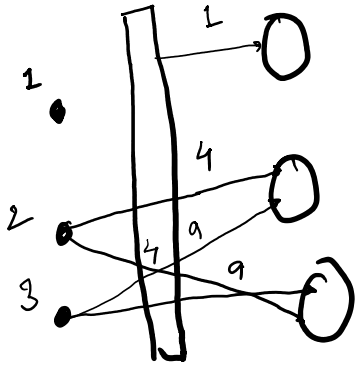
* Dense (64, activation = 'relu')
  Reshape (8, 8) → shape

  (8, 8) tensor

* One more layer → Regularization layers → Overfitting prevent.

  = { Lasso Reg, Ridge Reg, Elastic Net }

⑧ Lambda Layer. ⟶ Custom computation within model



Math

$X = [1, 2, 3, 4]$

Lambda layer — Custom logic $= x^2$ =

$X_{new} = [1, 4, 9, 16]$

Syntax

```
def squared_x(x):
    return x**2

Dense(64, activation='relu')
Lambda(squared_x)
```

Lambda (squared_x)