Ran Yang, Ph.D

Ran's Lab Presents

THE FUTURE IS NOW: AI, AND AI APPLICATIONS IN MEDICINE, DESIGN, AND THE MUSIC

rxyan2@wm.edu

CONTENTS

THREE MAIN TYPES OF MACHINE LEARNING

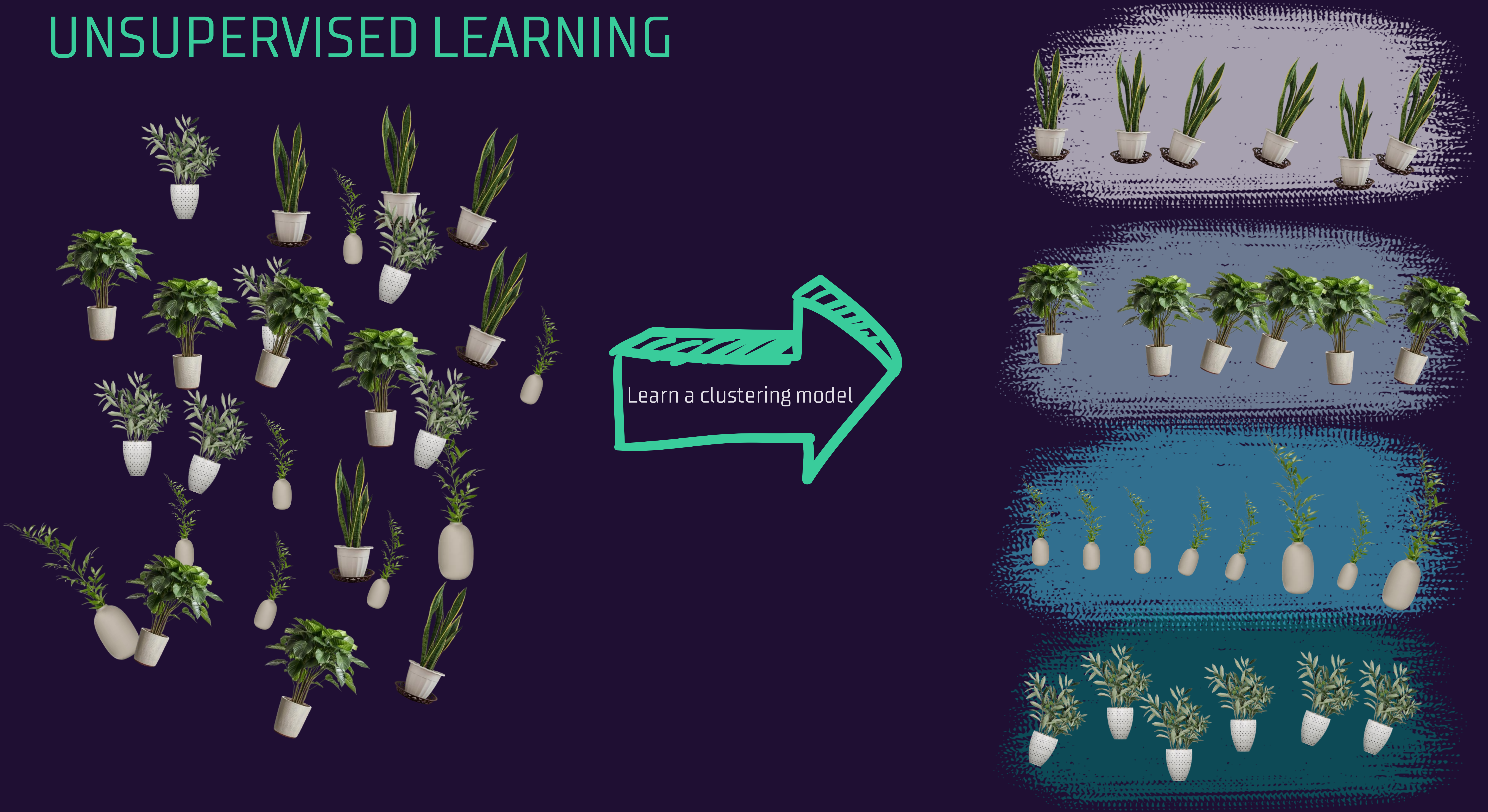
- SUPERVISED LEARNING
- UNSUPERVISED LEARNING
- REINFORCEMENT LEARNING

SUPERVISED LEARNING

Cityscapes Dataset: Example Zurich

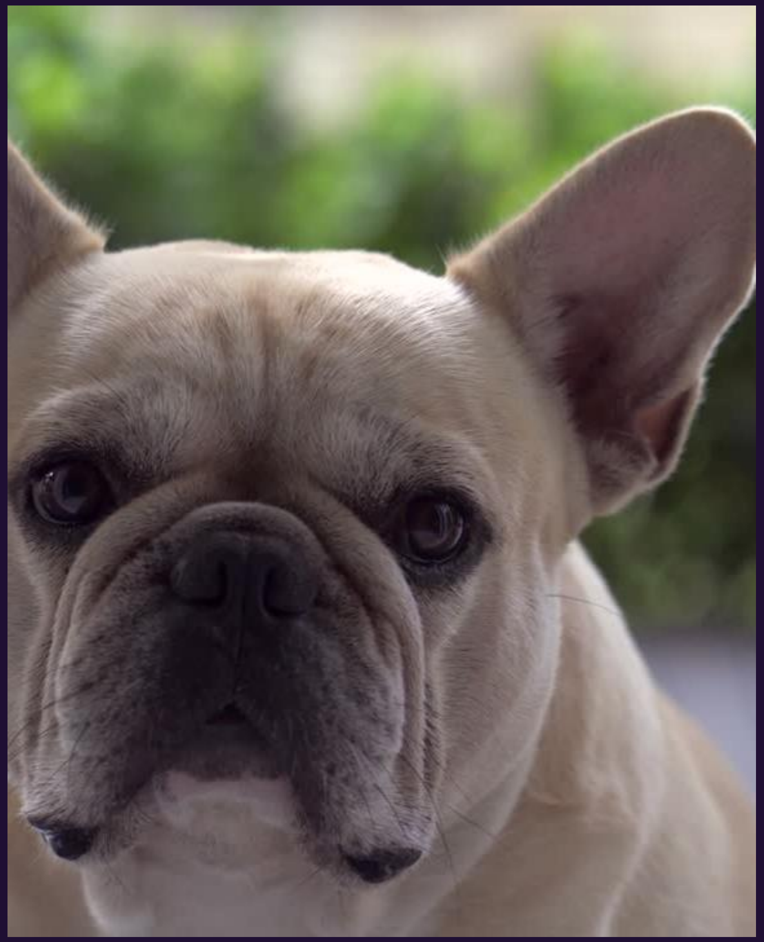
https://www.cityscapes-dataset.com/examples/

UNSUPERVISED LEARNING



REINFORCEMENT LEARNING

Dog – Agent



chill

State – Action

Dab

Reward

Cookie

Lay down

Penalty

Mean cat

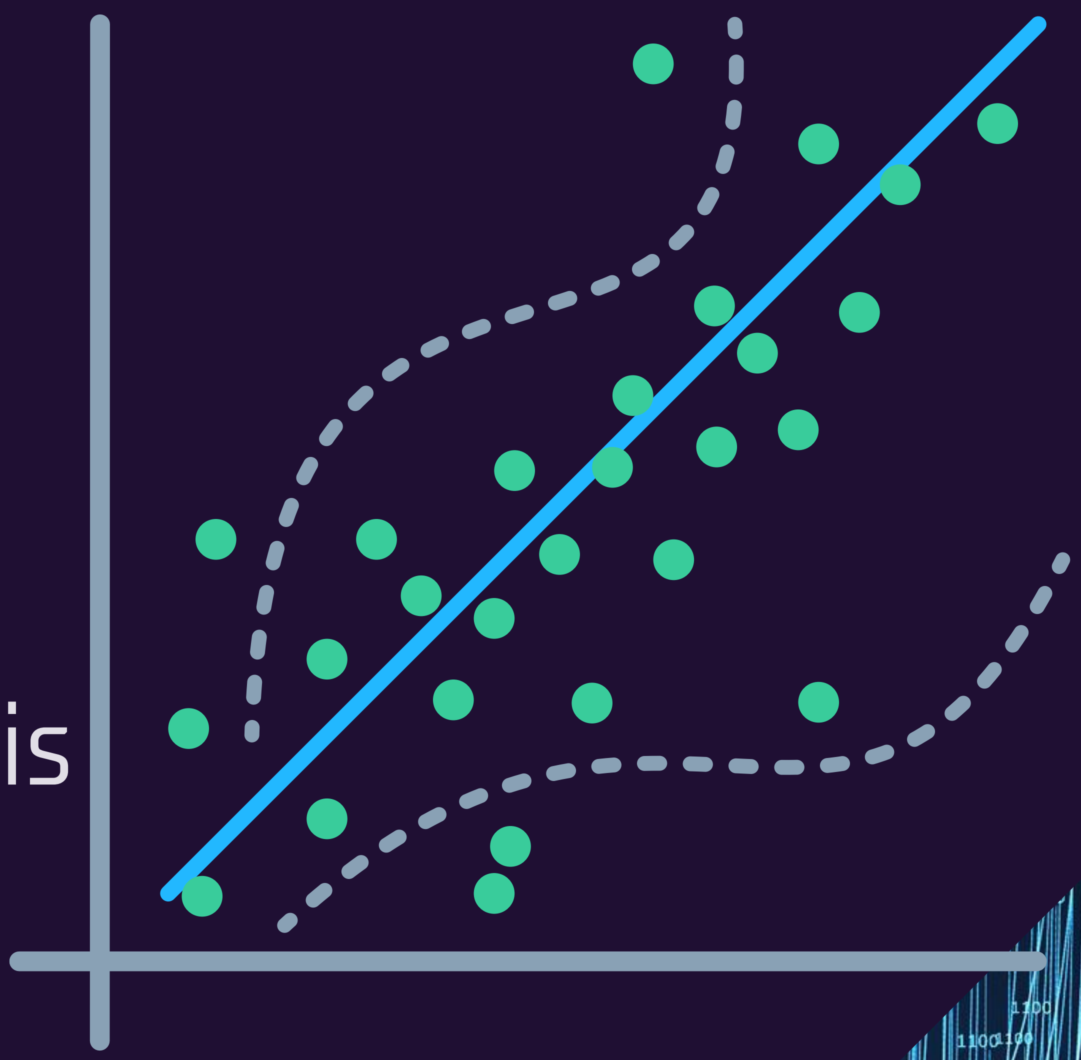
LINEAR REGRESSION

$X = [x_1, x_2, \ldots, x_n]$,

$y = [y_1, y_2, \ldots, y_n]$,

A linear regression model with m features is

$\hat{y}_i = \beta_0 + \beta_1 x_{1i} + \beta_2 x_{2i} + \ldots + \beta_m x_{mi}$



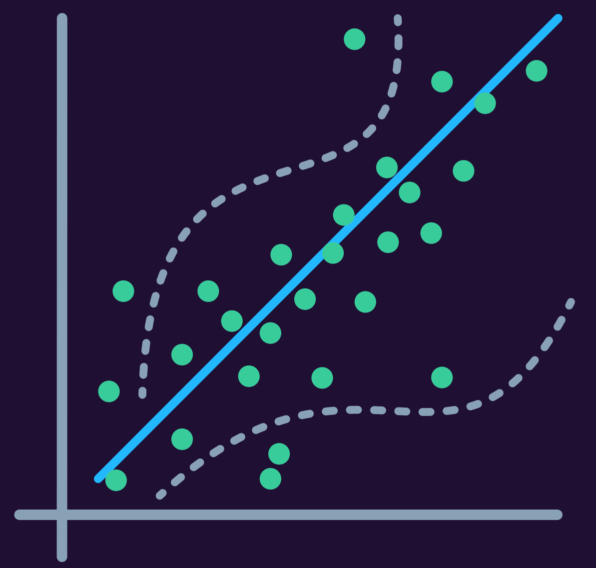
Loss – Mean Squared Error :

$MSE = (1/n) * \Sigma(\hat{y}_i - y_i)^2$, where $i = 1$ to n

Optimizer: Gradient Descent

$\beta_j := \beta_j - \alpha * \partial MSE/\partial \beta_j$, where $j = 0$ to m

OPTIMIZATION



Optimizer: Gradient Descent

$\beta_j := \beta_j - \alpha * \partial MSE/\partial\beta_j$, where j = 0 to m, α is the learning rate,

Partial Derivatives:

$\partial MSE/\partial\beta_0 = (2/n) * \Sigma(\hat{y}_i - y_i)$, where i = 1 to n $\partial MSE/\partial\beta_j = (2/n) * \Sigma(\hat{y}_i - y_i) * x_{ji}$, where i = 1 to n and j = 1 to m

Update Equations:

$\beta_0 := \beta_0 - \alpha * (2/n) * \Sigma(\hat{y}_i - y_i)$

$\beta_j := \beta_j - \alpha * (2/n) * \Sigma(\hat{y}_i - y_i) * x_{ji}$, where j = 1 to m

Once the coefficients are learned, the trained linear regression model can be used to make predictions on new, unseen data

$(\hat{y} = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + ... + \beta_m x_m)$.

AN EXAMPLE

- HOUSE
- HORCE

How an American gambler unlocked the secret to Hong Kong horse racing, winning almost US$1 billion

In the 1980s and 90s, computer nerd Bill Benter did the impossible: he wrote an algorithm that beat the unpredictability of the racetrack, winning big in the process

Listen to this article

Kit Chellel + FOLLOW

Published: 9:47am, 17 Jun 2018

Why you can trust SCMP

MATRIX (IS AMAZING)

$X = [x_1, x_2, \ldots, x_n]^T$

$y = [y_1, y_2, \ldots, y_n]^T$

$\hat{y} = X\beta$

$MSE = (1/n) * (y - \hat{y})^T(y - \hat{y})$

y is the vector of actual target values, of shape $(n, 1)$.

$\hat{y}$ is the vector of predicted values, of shape $(n, 1)$.

Gradient Descent: $\beta := \beta - \alpha * \nabla MSE$

α is the learning rate, which determines the step size of the updates.

∇MSE is the gradient of the MSE with respect to the coefficients β.

$\nabla MSE = (2/n) * X^T(X\beta - y)$ feature matrix X of shape $(m+1, n)$.

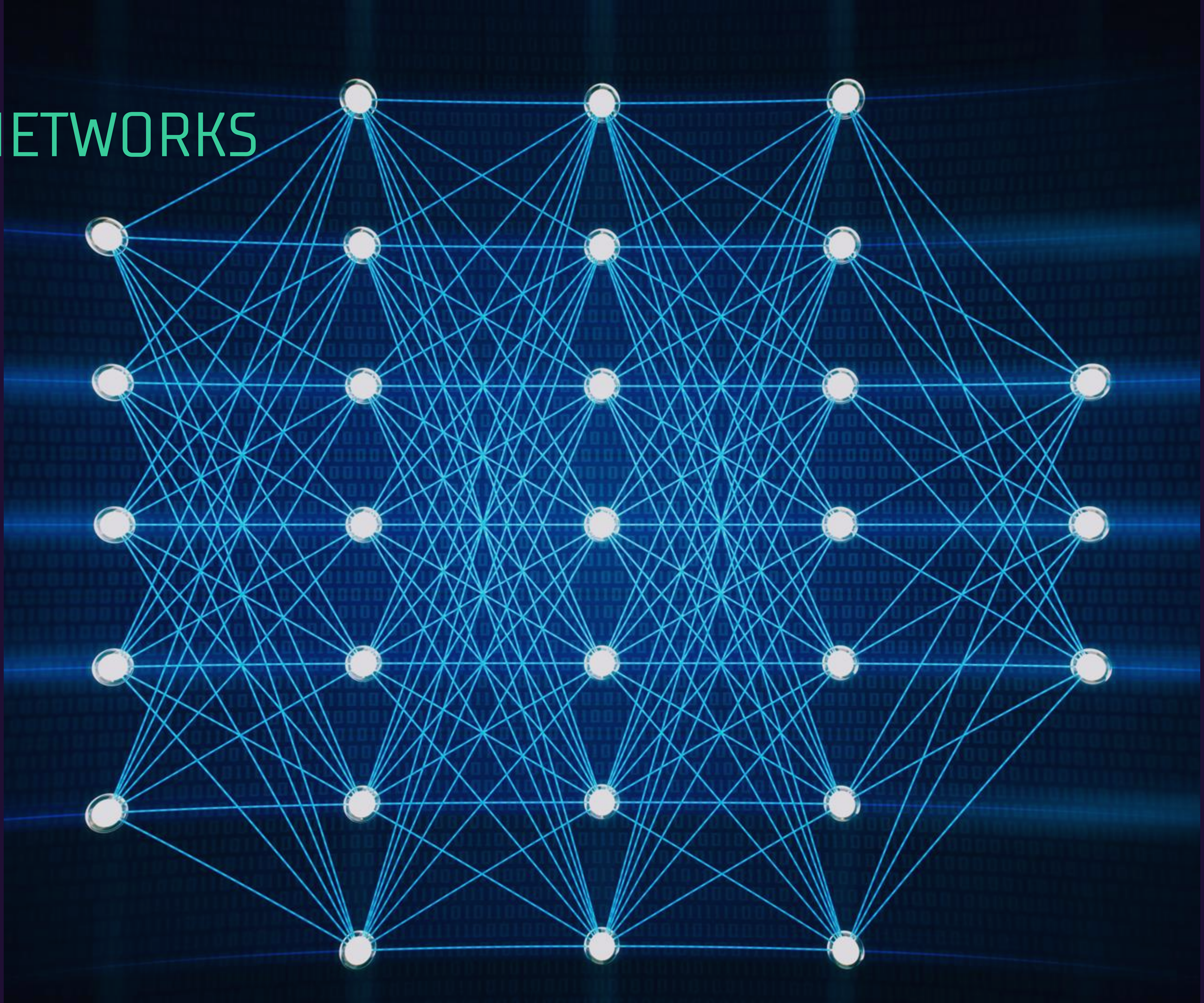
$\beta := \beta - \alpha * (2/n) * X^T(X\beta - y)$

NEURAL NETWORKS

INPUT LAYER

HIDDEN LAYERS

OUTPUT LAYER



NEURAL NETWORKS

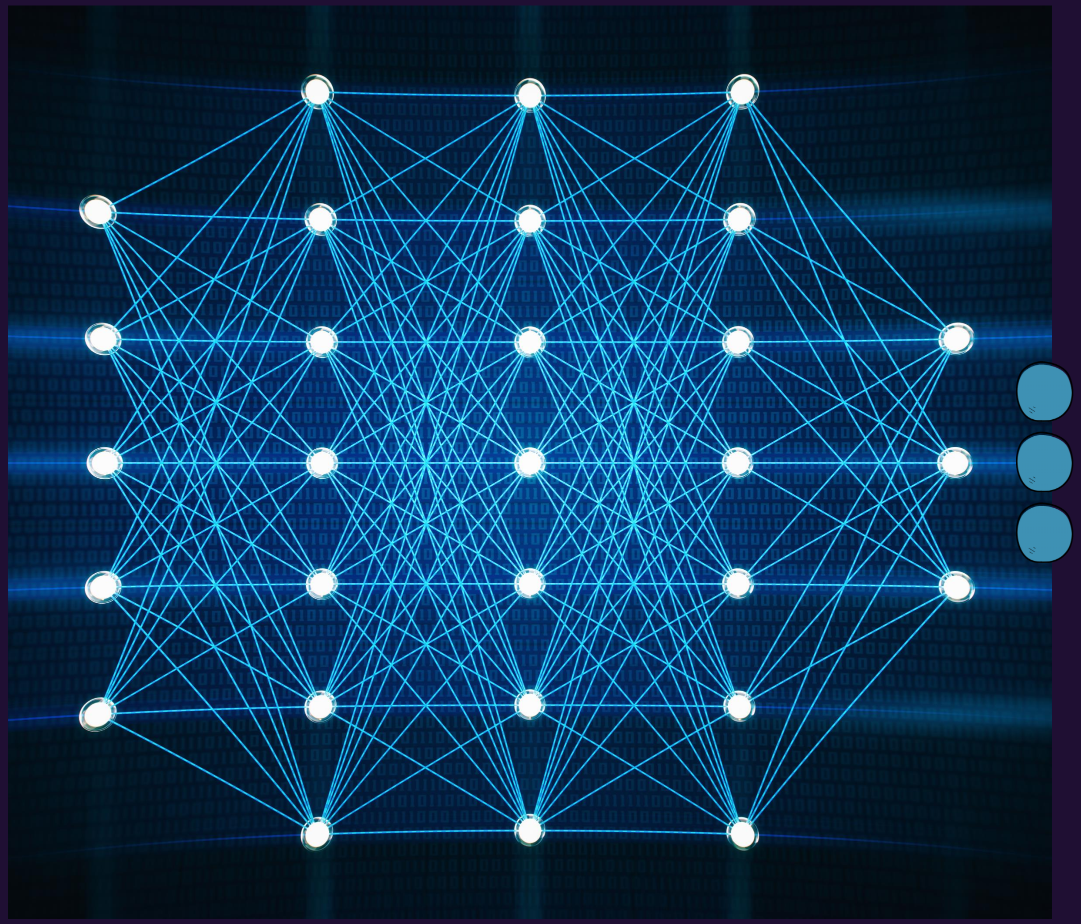
Input features: x_1, x_2, x_3, x_4, x_5

Hidden layers: h_1, h_2, h_3

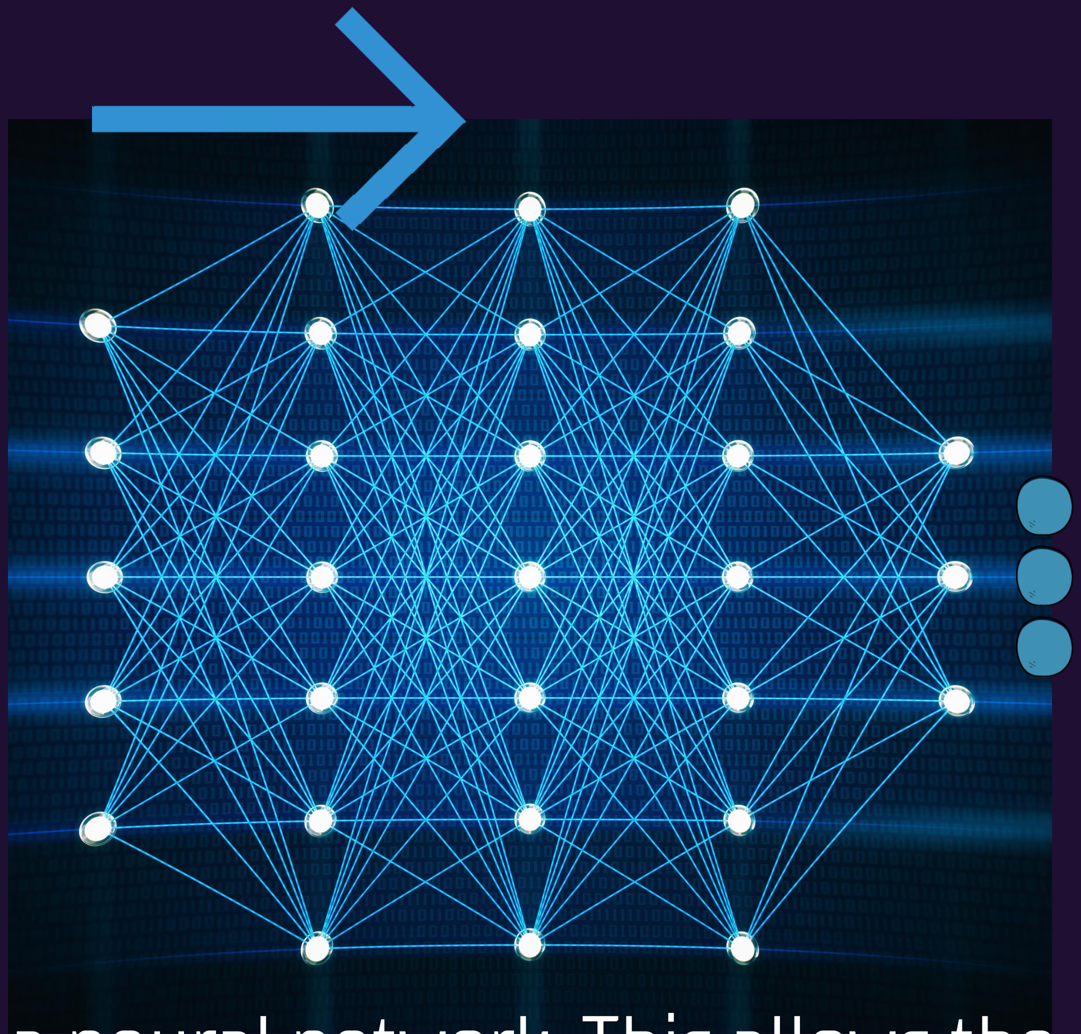
Output values: y_1, y_2

Weights and biases:

- W_1: weight matrix connecting the input layer to the first hidden layer, of shape (5, 7)
- b_1: bias vector for the first hidden layer, of shape (7, 1)
- W_2: (7, 7)
- b_2: (7, 1)
- W_3: (7, 7)
- b_3: (7, 1)
- W_4: (7, 3)
- b_4: (3, 1)



FORWARD PROPAGATION



Input layer to the first hidden layer: $z_1 = W_1 \cdot x + b_1$ $h_1 = a(z_1)$

First hidden layer to the second hidden layer: $z_2 = W_2 \cdot h_1 + b_2$ $h_2 = a(z_2)$

Second hidden layer to the third hidden layer: $z_3 = W_3 \cdot h_2 + b_3$ $h_3 = a(z_3)$

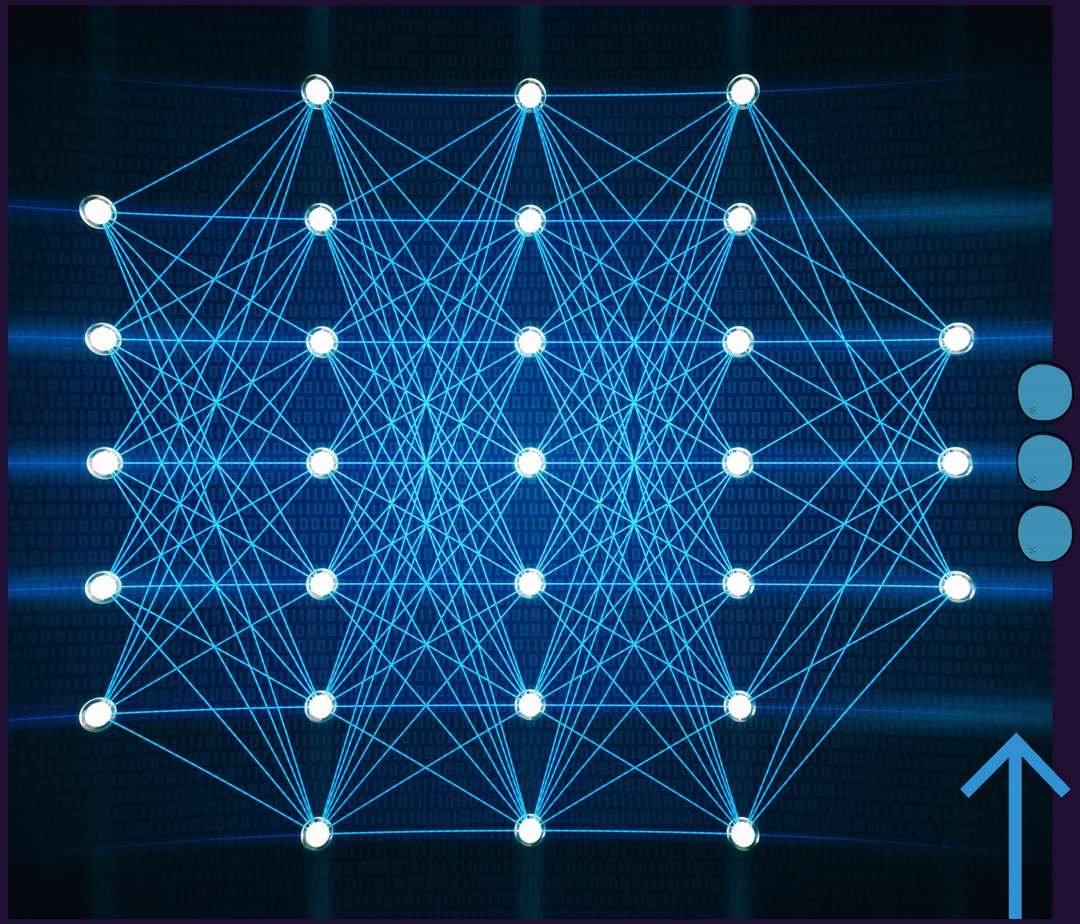
Third hidden layer to the output layer: $z_4 = W_4 \cdot h_3 + b_4$ $y = a(z_4)$

Activation function: a

A differentiable nonlinear activation function is used in the hidden layers of a neural network. This allows the model to learn more complex functions than a network trained using a linear activation function.

- Sigmoid: $f(z) = 1 / (1 + \exp(-z))$, $f'(z) = f(z) * (1 - f(z))$
- Hyperbolic Tangent (tanh): $f(z) = (\exp(z) - \exp(-z)) / (\exp(z) + \exp(-z))$, $f'(z) = 1 - f(z)^2$
- Rectified Linear Unit (ReLU): $f(z) = \max(0, z)$, $f'(z) = 1$ if $z > 0$ else 0
- Leaky ReLU: $f(z) = \max(\alpha z, z)$, where α is a small positive constant, $f'(z) = 1$ if $z > 0$ else α
- Softmax (for output layer in multi-class classification): $f(z)_i = \exp(z_i) / \sum_j \exp(z_j)$, $f'(z)_i = f(z)_i * (1 - f(z)_i)$

LOSS



y_true: the true output values, of shape (3, 1)

L: the loss function, which measures the difference between the predicted output (y) and the true output (y_true)

Loss Functions:

Mean Squared Error (MSE) for regression problems: $L = (1/2) \cdot \Sigma(y - y\_true)^2$

Binary Cross-Entropy for binary classification problems:

$L = -[y\_true \cdot \log(y) + (1 - y\_true) \cdot \log(1 - y)]$

Categorical Cross-Entropy for multi-class classification problems: $L = -\Sigma(y\_true \cdot \log(y))$

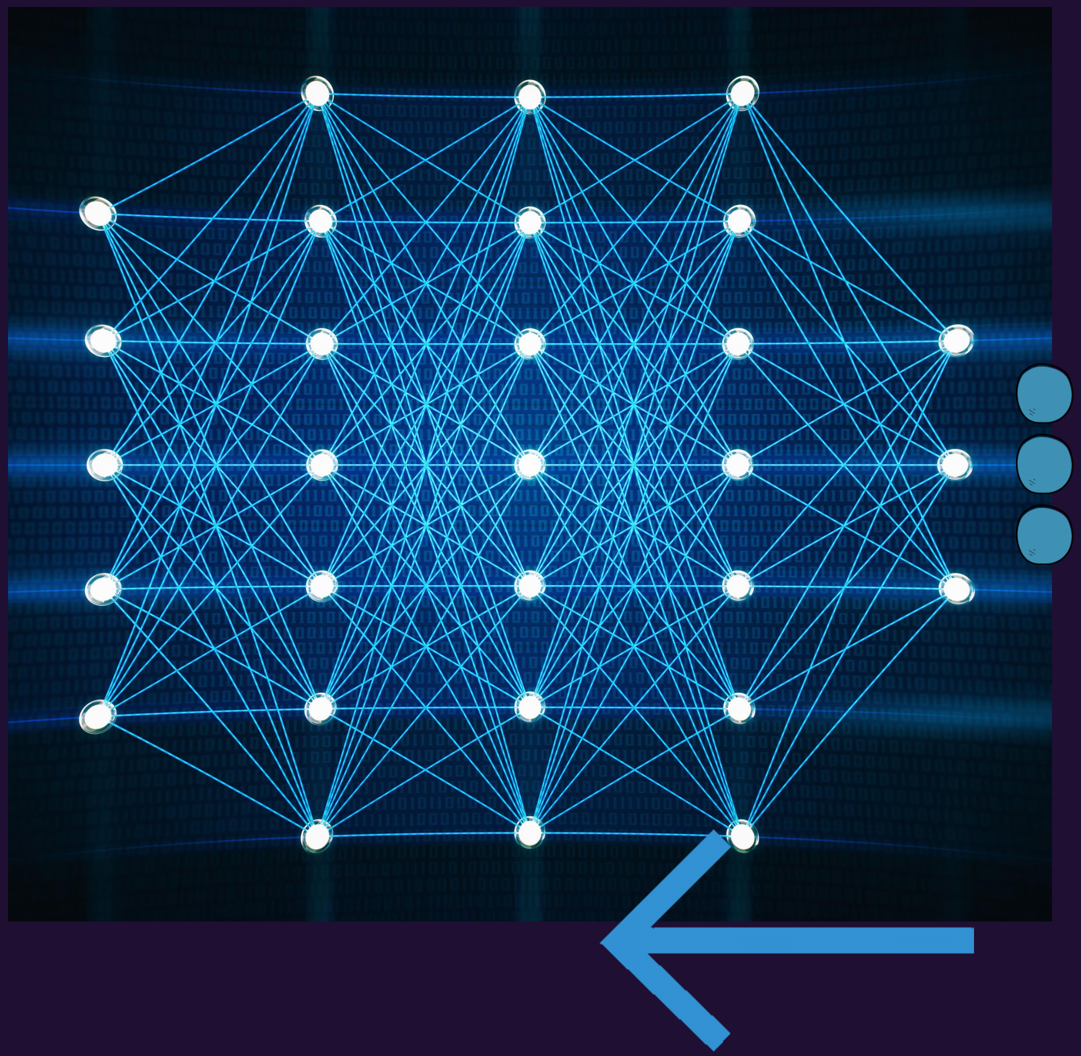
BACKWARD PROPAGATION

Output layer: $\delta_4 = \partial L/\partial y \odot a'(z_4)$ $\partial L/\partial W_4 = \delta_4 \cdot h_3^T$ $\partial L/\partial b_4 = \delta_4$

h3: $\delta_3 = (W_4^T \cdot \delta_4) \odot a'(z_3)$ $\partial L/\partial W_3 = \delta_3 \cdot h_2^T$ $\partial L/\partial b_3 = \delta_3$

h2: $\delta_2 = (W_3^T \cdot \delta_3) \odot a'(z_2)$ $\partial L/\partial W_2 = \delta_2 \cdot h_1^T$ $\partial L/\partial b_2 = \delta_2$

h1: $\delta_1 = (W_2^T \cdot \delta_2) \odot a'(z_1)$ $\partial L/\partial W_1 = \delta_1 \cdot x^T$ $\partial L/\partial b_1 = \delta_1$

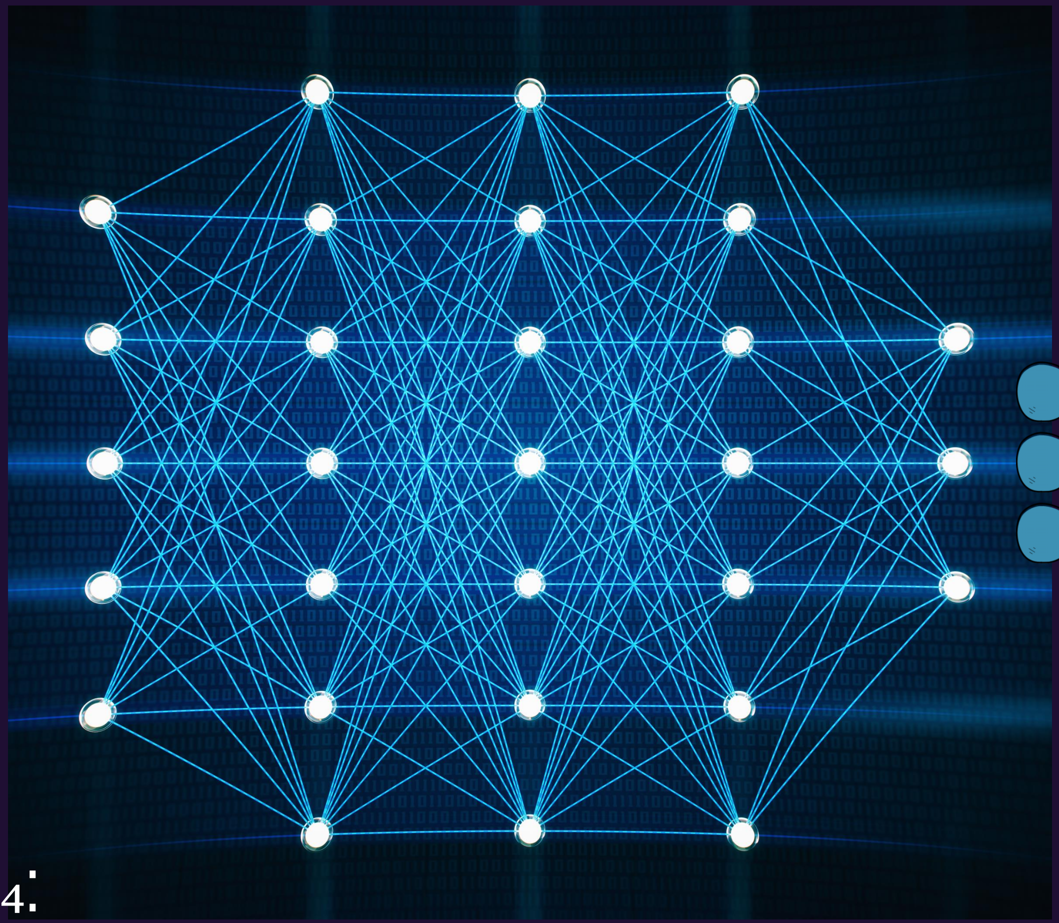


The gradients: $(\partial L/\partial W_1, \partial L/\partial b_1, \partial L/\partial W_2, \partial L/\partial b_2, \partial L/\partial W_3, \partial L/\partial b_3, \partial L/\partial W_4, \partial L/\partial b_4)$ are used to update the weights and biases optimization algorithm, such as gradient descent.

α is learning rate.

$$
\begin{aligned}
W_1 &:= W_1 - \alpha \cdot \partial L/\partial W_1 \\
b_1 &:= b_1 - \alpha \cdot \partial L/\partial b_1 \\
W_2 &:= W_2 - \alpha \cdot \partial L/\partial W_2 \\
b_2 &:= b_2 - \alpha \cdot \partial L/\partial b_2 \\
W_3 &:= W_3 - \alpha \cdot \partial L/\partial W_3 \\
b_3 &:= b_3 - \alpha \cdot \partial L/\partial b_3 \\
W_4 &:= W_4 - \alpha \cdot \partial L/\partial W_4 \\
b_4 &:= b_4 - \alpha \cdot \partial L/\partial b_4
\end{aligned}
$$

BACKWARD PROPAGATION – IN CASE YOU'RE CURIOUS



$L = L(y(z_4(W_4, h_3, b_4)))$

- L is the loss function
- $y = a(z_4)$ is the output of the network, which is the activation function a applied to the weighted sum z_4
- $z_4 = W_4 \cdot h_3 + b_4$

Find the gradients $\partial L/\partial W_4$ and $\partial L/\partial b_4$ using the chain rule.

For the weights W_4:

$\partial L/\partial W_4$

$= \partial L/\partial y \cdot \partial y/\partial z_4 \cdot \partial z_4/\partial W_4$

$= (\partial L/\partial y \odot a'(z_4)) \cdot (h_3^T)$ (Using the chain rule)

$= \delta_4 \cdot h_3^T$

For the biases b_4:

$\partial L/\partial b_4$

$= \partial L/\partial y \cdot \partial y/\partial z_4 \cdot \partial z_4/\partial b_4$

$= (\partial L/\partial y \odot a'(z_4)) \cdot (1)$ (Since $\partial z_4/\partial b_4 = 1$)

$= \delta_4$

$\delta_4 = \partial L/\partial y \odot a'(z_4)$

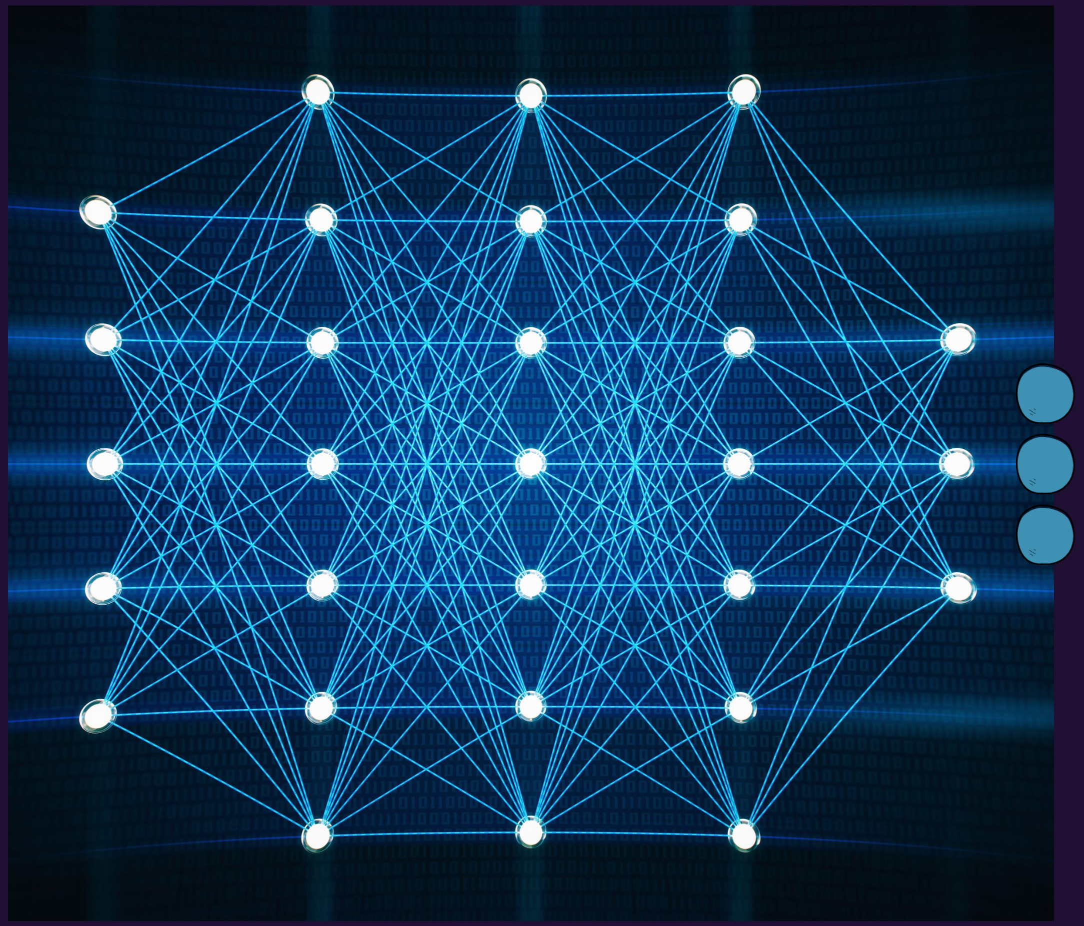
$\partial L/\partial W_4 = \delta_4 \cdot h_3^T$

$\partial L/\partial b_4 = \delta_4$

BACKWARD PROPAGATION – IN CASE YOU'RE CURIOUS MORE

The hidden layers have dimensions n_1, n_2, and n_3, the number of parameters is:

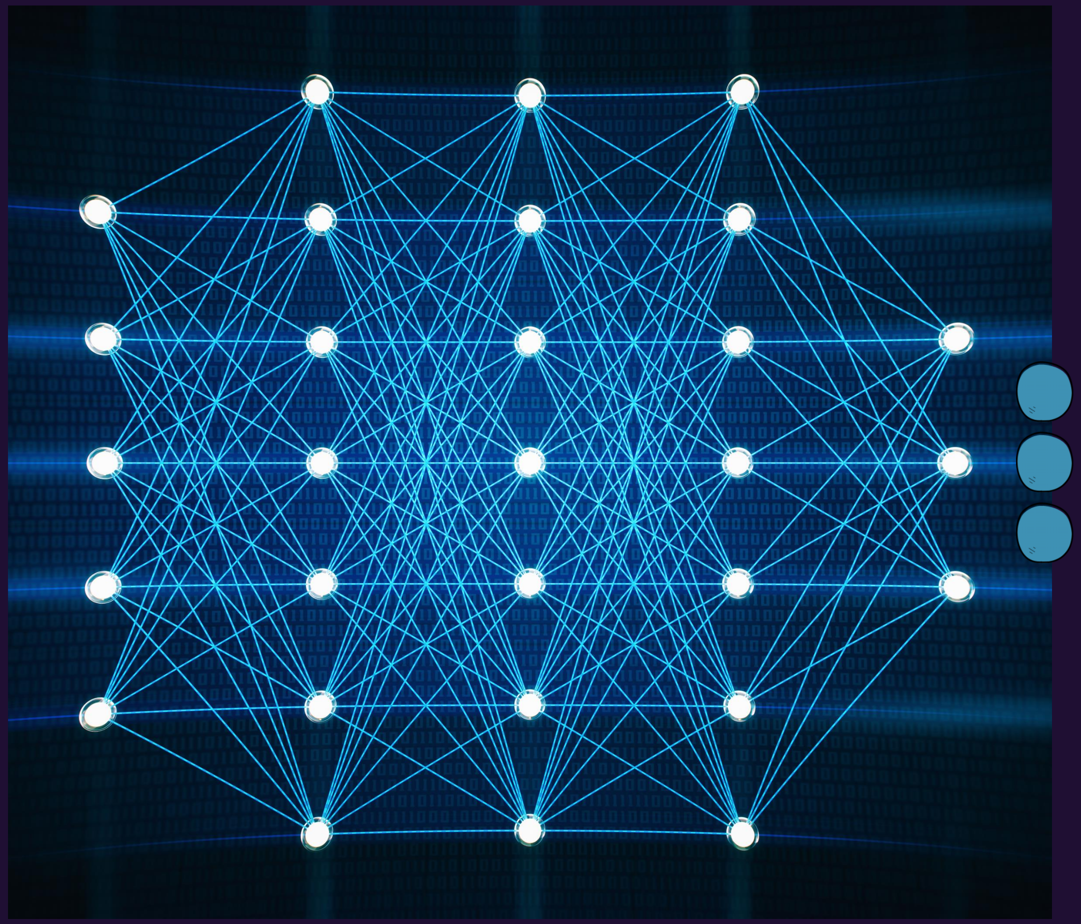
- W_1: $5 \times n_1$ parameters
- b_1: n_1 parameters
- W_2: $n_1 \times n_2$ parameters
- b_2: n_2 parameters
- W_3: $n_2 \times n_3$ parameters
- b_3: n_3 parameters
- W_4: $n_3 \times 3$ parameters
- b_4: 3 parameters



Our example n=7

Parameters: 35 + 7 + 49 + 7 + 49 + 7 + 21 + 3 = 178

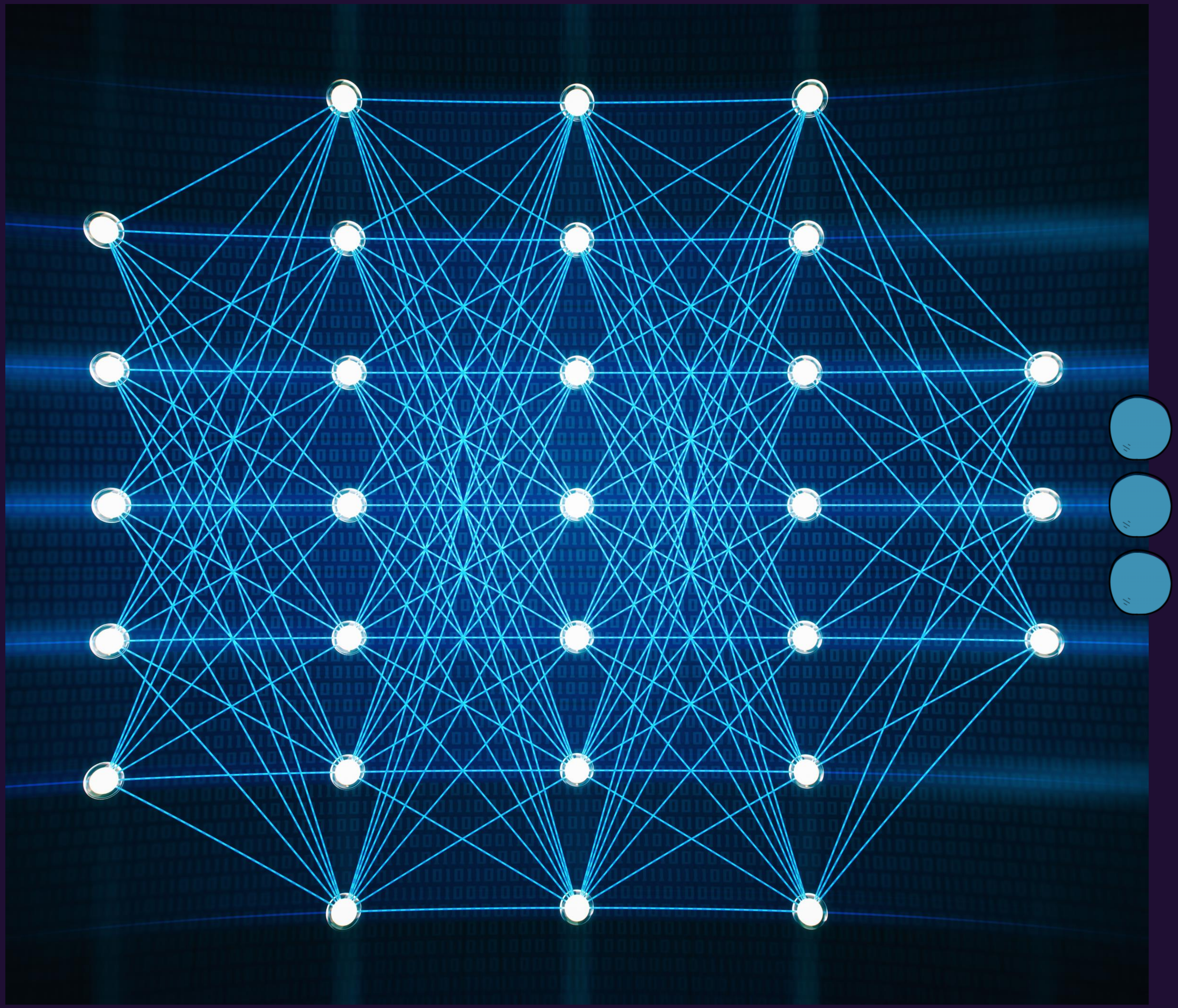
GRADIENT DESCENT



- Batch Gradient Descent: Calculates the gradient using the entire training dataset at each iteration. This can be computationally expensive for large datasets.
- Stochastic Gradient Descent (SGD): Calculates the gradient using a single, randomly selected data point at each iteration. This is much faster than batch gradient descent but can lead to noisy updates.
- Mini-Batch Gradient Descent: Calculates the gradient using a small, randomly selected subset (mini-batch) of the training data at each iteration. This provides a balance between the stability of batch gradient descent and the speed of stochastic gradient descent.

1 iteration – the forward propagation, loss calculation, backward propagation, and weight update steps have been performed once
Epoch – one complete pass through the entire training dataset.
Batch size can vary

MULTI-LAYER PERCEPTRON (STANDARD FORWARD NEURAL NETWORK)



DEEP LEARNING

Highlight dog

DEEP LEARNING – CNN

- Input Layer
- Convolutional Layers: The input image passes through one or more convolutional layers, where filters (kernels) are convolved with the input to extract features. The convolutional layers capture local patterns and spatial information in the image.
- Activation Functions: After each convolutional layer, an activation function like ReLU is applied element-wise to introduce non-linearity.
- Pooling Layers: Pooling layers (e.g., max pooling or average pooling) are often used to downsample the feature maps, reducing the spatial dimensions and providing translation invariance.
- Output Layer: The final output layer produces a segmentation mask, where each pixel is assigned a class label. The number of output channels in this layer corresponds to the number of classes (in this case, two: dog and background).

$y(t)=x(t)*h(t)$

WHAT'S CONVOLUTION

$(f * g)(x, y) = \Sigma(i, j)\ f(i, j)\ g(x-i, y-j)$

where (x, y) are the spatial coordinates of the output, and the summation is taken over all valid spatial positions (i, j) for which the kernel g is fully contained within the input f.

Two vectors a = [1, 2, 3, 4] and b = [5, 6, 7, 8]

The convolution of a and b, denoted as c = a * b

c[0] = a[0] * b[0] = 1 * 8 = 8

c[1] = a[0] * b[1] + a[1] * b[0] = 1 * 7 + 2 * 8 = 23

c[2] = a[0] * b[2] + a[1] * b[1] + a[2] * b[0] = 1 * 6 + 2 * 7 + 3 * 8 = 46

c[3] = a[0] * b[3] + a[1] * b[2] + a[2] * b[1] + a[3] * b[0] = 1 * 5 + 2 * 6 + 3 * 7 + 4 * 8 = 77

c[4] = a[1] * b[3] + a[2] * b[2] + a[3] * b[1] = 2 * 5 + 3 * 6 + 4 * 7 = 58

c[5] = a[2] * b[3] + a[3] * b[2] = 3 * 5 + 4 * 6 = 39

c[6] = a[3] * b[3] = 4 * 5 = 20

c = a * b = [8, 23, 46, 77, 58, 39, 20]

</>

```
import numpy as np
a = np.array([1, 2, 3, 4])
b = np.array([5, 6, 7, 8])
convolution_result = np.convolve(a, b)
```

$$y(t)=x(t)*h(t)$$

WHAT'S CONVOLUTION

More than 2D matrix – say 3 by 3

convolution involves "sliding" one matrix over another, calculating the sum of element-wise products at each position.

Edge Handling: When convolving two matrices, handling the edges requires special attention.

```
[[  9  26  50  38  21]
 [ 42  94 154 106  54]
 [ 90 186 285 186  90]
 [ 54 106 154  94  42]
 [ 21  38  50  26   9]]
```

</>

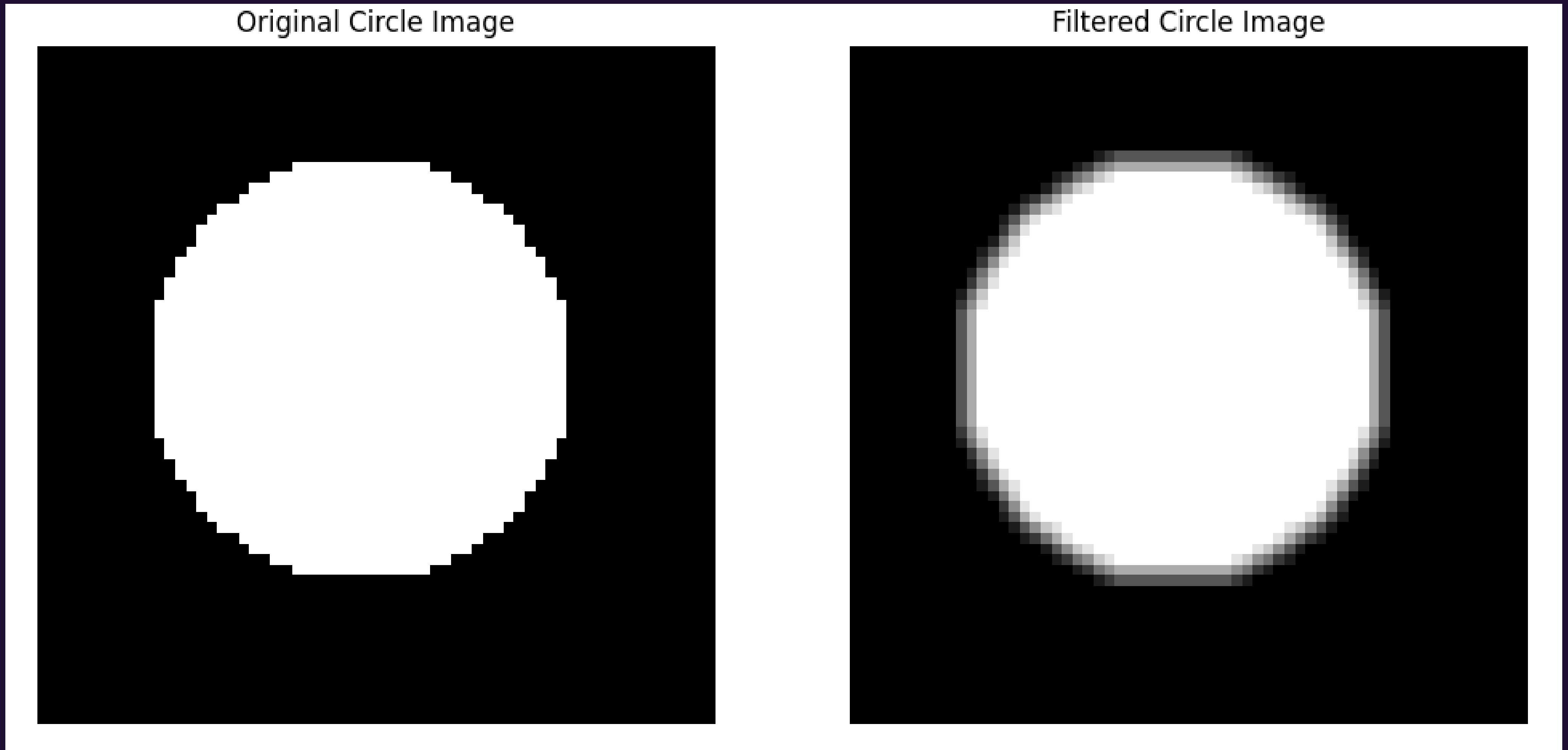
```python
import numpy as np
from scipy.signal import convolve2d
a = np.array([[1, 2, 3],
 [4, 5, 6],
 [7, 8, 9]])
b = np.array([[9, 8, 7],
 [6, 5, 4],
 [3, 2, 1]])
convolution = convolve2d(a, b, mode='full')
print(convolution)
```

$y(t)=x(t)*h(t)$

WHAT'S CONVOLUTION

Now we have an 64 by 64 image, applying a blurring (average) filter

convolution involves "sliding" one matrix over another, calculating the sum of element-wise products at each position.

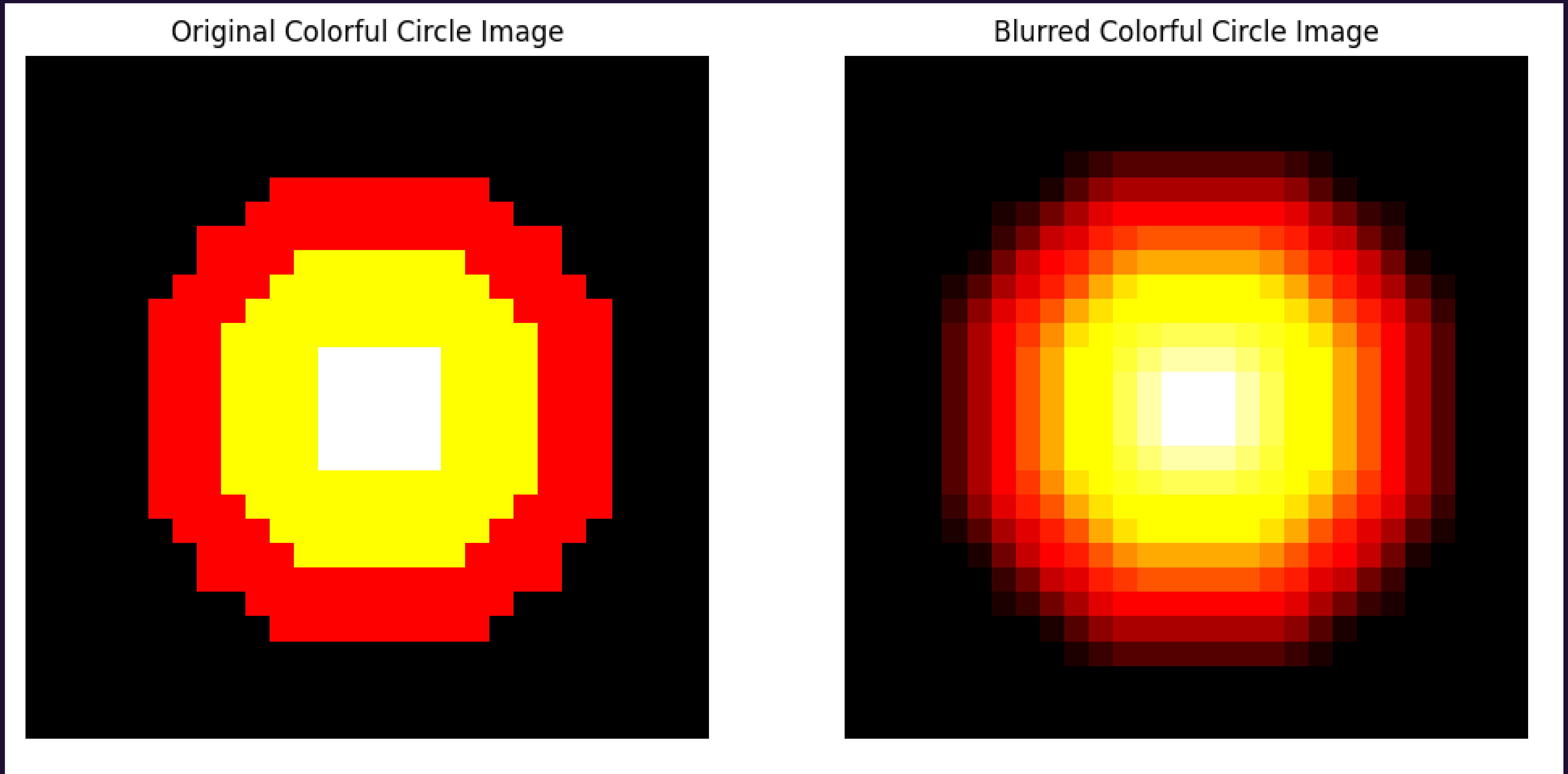



```
import matplotlib.pyplot as plt
import numpy as np
from scipy.signal import convolve2d
kernel = np.ones((3, 3)) / 9
image_circle = np.zeros((64, 64))
center = (30, 30)
radius = 20
for x in range(image_circle.shape[0]):
 for y in range(image_circle.shape[1]):
  if (x - center[0])**2 + (y - center[1])**2 <
radius**2:
  image_circle[x, y] = 2
image_circle_normalized = image_circle
convolved_circle_image = convolve2d(
 image_circle_normalized, kernel, mode='same')
```

$y(t)=x(t)*h(t)$

WHAT'S CONVOLUTION

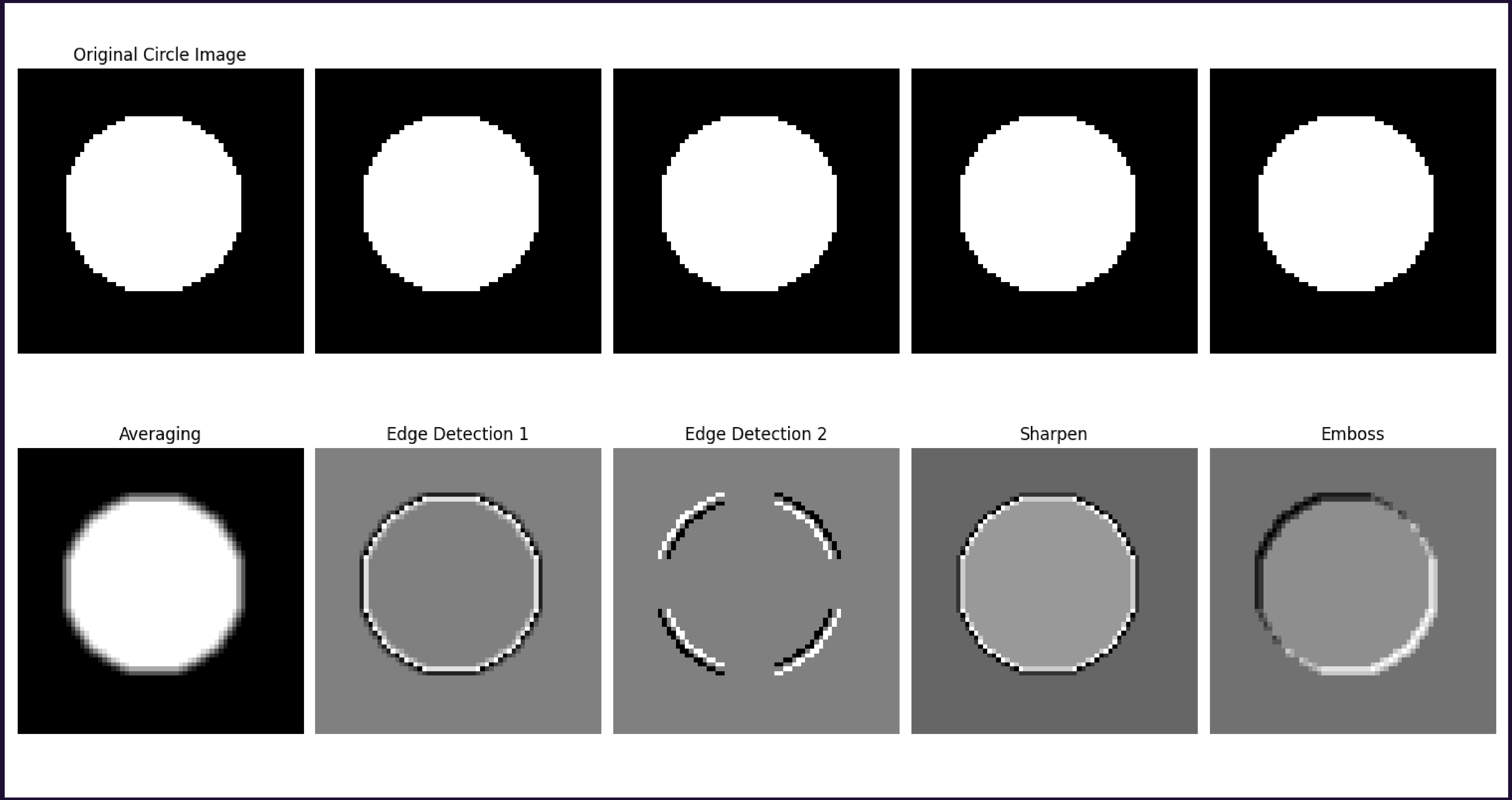
Colored Image: RBG 3 channels. The Convolution doesn't mix the channels; it applies the kernel separately to each channel. Convolution doesn't mix the channels; it applies the kernel separately to each channel.

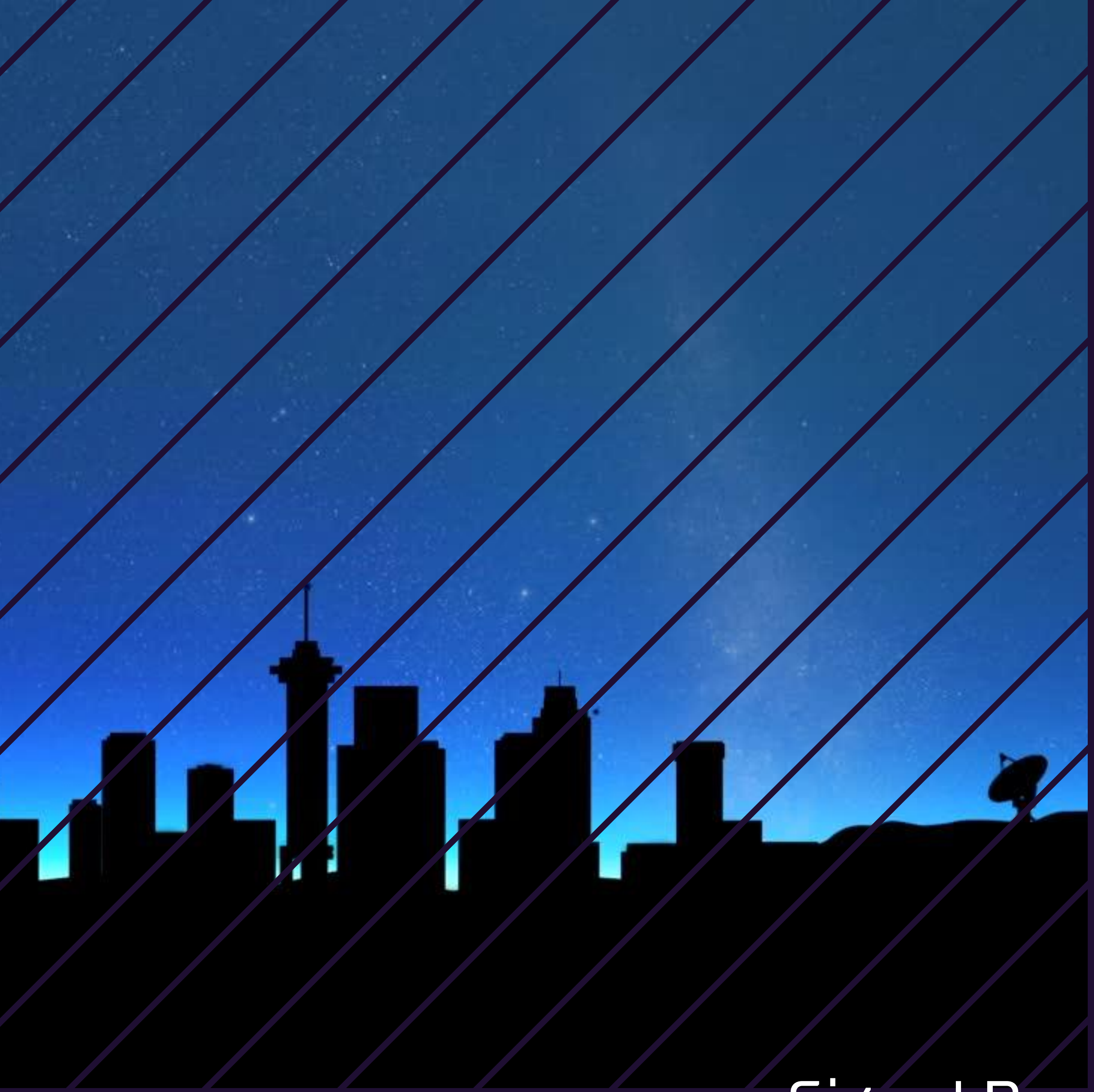


WHAT'S CONVOLUTION

$y(t)=x(t)*h(t)$

Now we have the 64 by 64 circle image, applying 5 filters (3 by 3 kernels)





WHAT IF I HAVE 1 MILLION HD IMAGES?

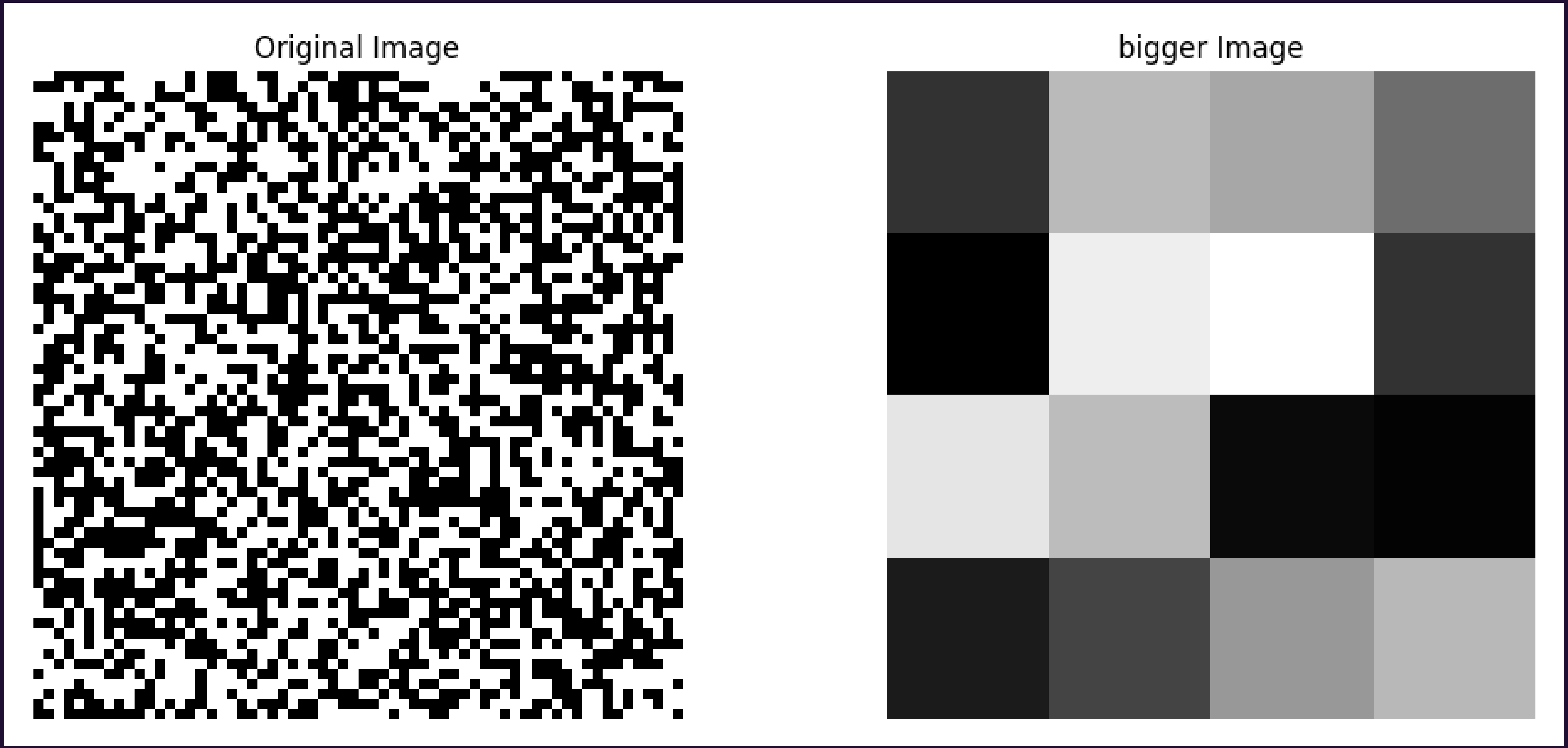
Convolution Theorem: under suitable conditions, the Fourier transform of the convolution of two signals is the point-wise product of their Fourier transforms.

Efficiency: The FFT is a highly efficient algorithm for computing the Fourier transform, especially for data sizes that are powers of 2. Its O(nlogn)complexity makes the entire process much faster for large datasets compared to the direct convolution approach, which has O(n2)complexity for two signals of size n.

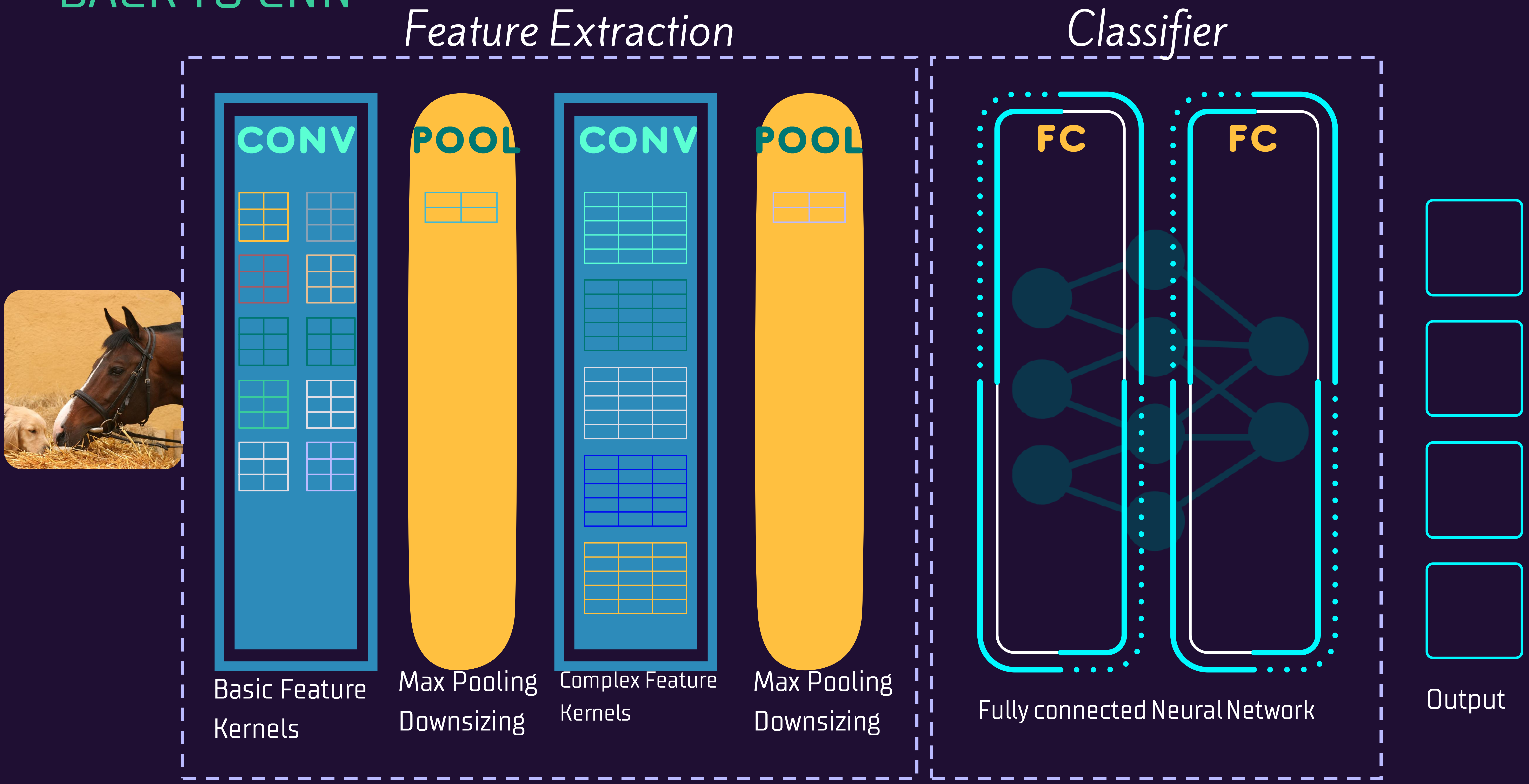
- Signal Processing: Filtering signals, analyzing frequency components
- Image Processing: Applying filters
- Deep Learning: in convolutional neural networks (CNNs), where convolution operations are fundamental, though implemented differently for training efficiency.

BACK TO CNN

8 Bit per pixel : 0-255



BACK TO CNN


Feature Extraction
Classifier
CONV
POOL
CONV
POOL
FC
FC
Basic Feature Kernels
Max Pooling Downsizing
Complex Feature Kernels
Max Pooling Downsizing
Fully connected Neural Network
Output

CNN

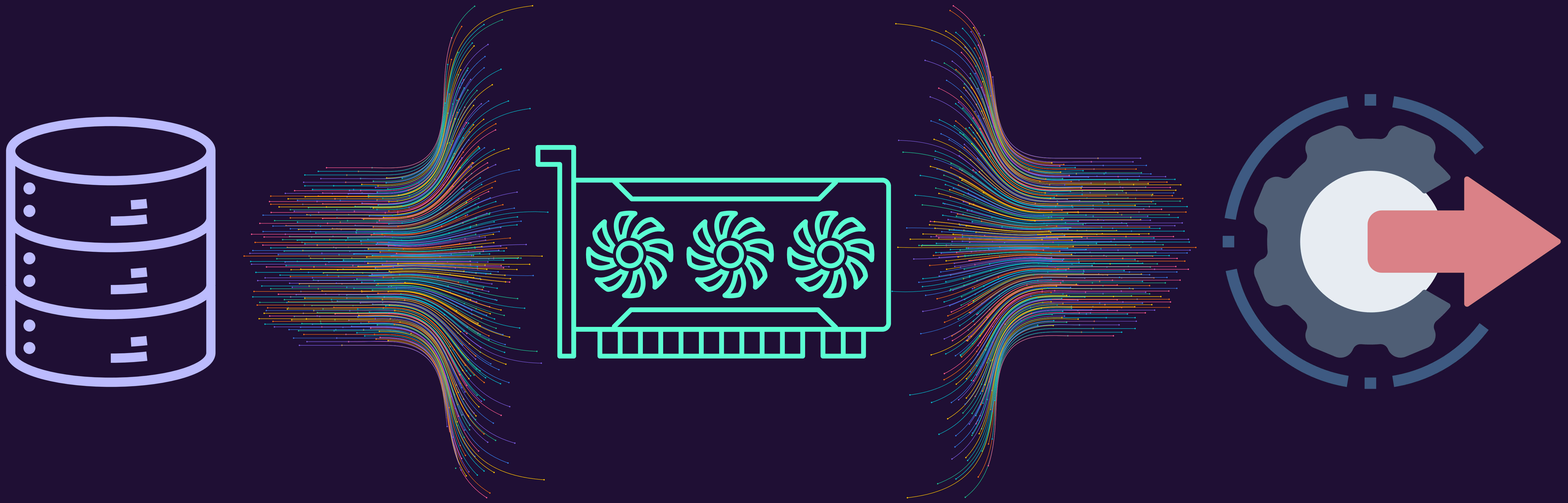
Deep Learning Frameworks:

Frameworks like PyTorch (Meta) and TensorFlow (Google) provide high-level APIs and libraries to define, train, and deploy deep learning models, including CNNs.

These frameworks offer a wide range of built-in layers, loss functions, optimizers, and other utilities that make it easier to build and train complex models

NVIDIA CUDA (Compute Unified Device Architecture) is a parallel computing platform and programming model.

- CUDA allows developers to leverage the power of NVIDIA GPUs for parallel processing.
- CUDA provides a set of libraries, tools, and compilers that enable efficient utilization of NVIDIA GPUs.



- Linear Regression
- Multi-layer Perceptron
- Convolutional Neural Network
- Transformer
- ...

TRANSFORMER

Megatron LM (nVidia)

DeepSpeed (Microsoft)

MaxText (Google)

Where is Apple?

Hmmm....

Metal and CoreML

Transformer is the T of GPT

Transformers, the "T" in GPT (Generative Pre-trained Transformer), are a type of deep learning model architecture that has revolutionized natural language processing (NLP) tasks.

The CORE of all the AI buzz!

Pile and Pile of Matrices

2017 “Attention is all you need” from Google

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł. and Polosukhin, I., 2017. Attention is all you need. Advances in neural information processing systems, 30.

Cited:115919 OMG

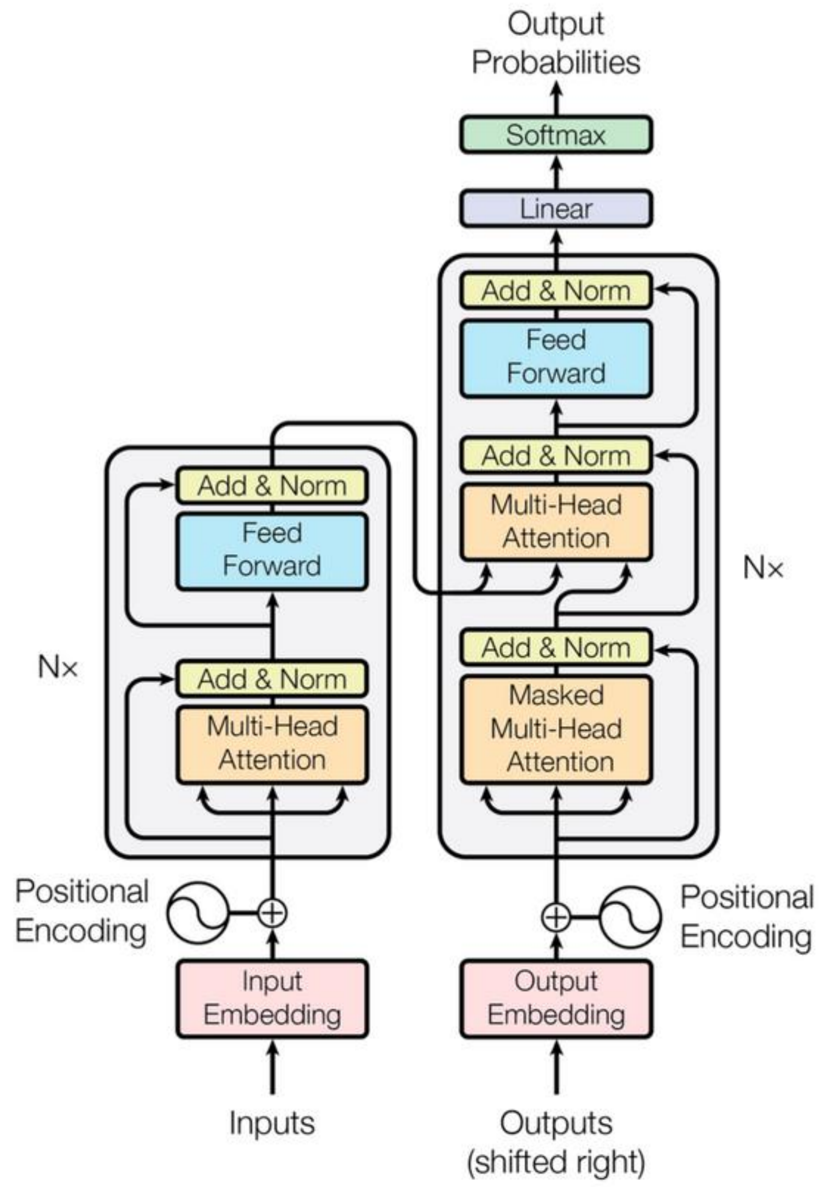


Figure 1: The Transformer - model architecture.

The Transformer follows this overall architecture using stacked self-attention and point-wise, fully connected layers for both the encoder and decoder, shown in the left and right halves of Figure 1, respectively.

How does this work?

I don't care that they stole my idea. I care that they don't have any of their own.

1

TOKEN

TOKENIZATION -BYTE PAIR ENCODING (BPE)

```
from transformers import GPT2Model, GPT2Config, GPT2Tokenizer
tokenizer = GPT2Tokenizer.from_pretrained('gpt2')
config = GPT2Config.from_pretrained('gpt2')
sentence = "I don't care that they stole my idea. I care that they don't have any of their own."
tokens = tokenizer.tokenize(sentence)
token_ids = tokenizer.convert_tokens_to_ids(tokens)
embedding_size = config.hidden_size
print(f"This sentence has {len(tokens)} tokens, they are: {
tokens}, each token vector's embedded dimension is {embedding_size}.")
```

This sentence has 22 tokens, they are: ['I', 'Ġdon', "'t", 'Ġcare', 'Ġthat', 'Ġthey', 'Ġstole', 'Ġmy', 'Ġidea', '.', 'ĠI', 'Ġcare', 'Ġthat', 'Ġthey', 'Ġdon', "'t", 'Ġhave', 'Ġany', 'Ġof', 'Ġtheir', 'Ġown', '.'], each token vector's embedded dimension is 768.

Attention – Embedding Input

2

I don't care that they stole my idea. I care that they don't have any of their own.

now Each token is a high dimension vector [768,1], GPT3 is [12288,1].
I printed first 10 for "I" for the vector e

-0.0796,
-0.0654,
-0.0842,
-0.0337,
-0.0758,
-0.2051,
-0.4378,
-0.1028,
-0.1290,
0.0155,
…
…

```
Shape of 'I' embedding vector: torch.Size([768])
Some values of 'I' embedding vector: tensor([-0.0796, -0.0654, -0.0842, -0.0337, -0.0758, -0.2051, -0.4378, -0.1028, -0.1290,  0.0155])
```

High-Dimensional Feature Space
Contextual Information

Single “Head” of Attention

I don't care that they stole my idea. I care that they don't have any of their own.

3

encode the position of “I”. Positional encodings help to incorporate the notion of token order.

Embedding Vector ei=E[tokeni]
Positional Encoding Vector pi
Combined Vector ei′=ei+pi

Form Query, Key, and Value

$$Q = e_i' W^Q$$

$$K = e_i' W^K$$

$$V = e_i' W^V$$

4

0.1 0.0 -0.20
.20.2 0.1 0.0
-0.1-0.1 0.2 0.1
0.00.0 0.3 0.2
0.1
... ...

WQ

0.1796,
-0.0654,
0.4842,
-0.0637,
-2.0758,
-1.2051,
-
...

ei

=

Q

SINGLE HEAD ATTENTION MECHANISM

I don't care that they stole my idea. I care that they don't have any of their own.

Calculation of Attention Scores, Dot product

5 $$Score(Q,K) = QK^T$$

Scaling and Normalization

6 $$Score_{scaled}(Q,K) = \frac{QK^T}{\sqrt{d_K}}$$

Application of Attention Weights to Value Vectors

7 $$Attention(Q,K,V) = Softmax\left(\frac{QK^T}{\sqrt{d_K}}\right)V$$

Multi-Head Attention

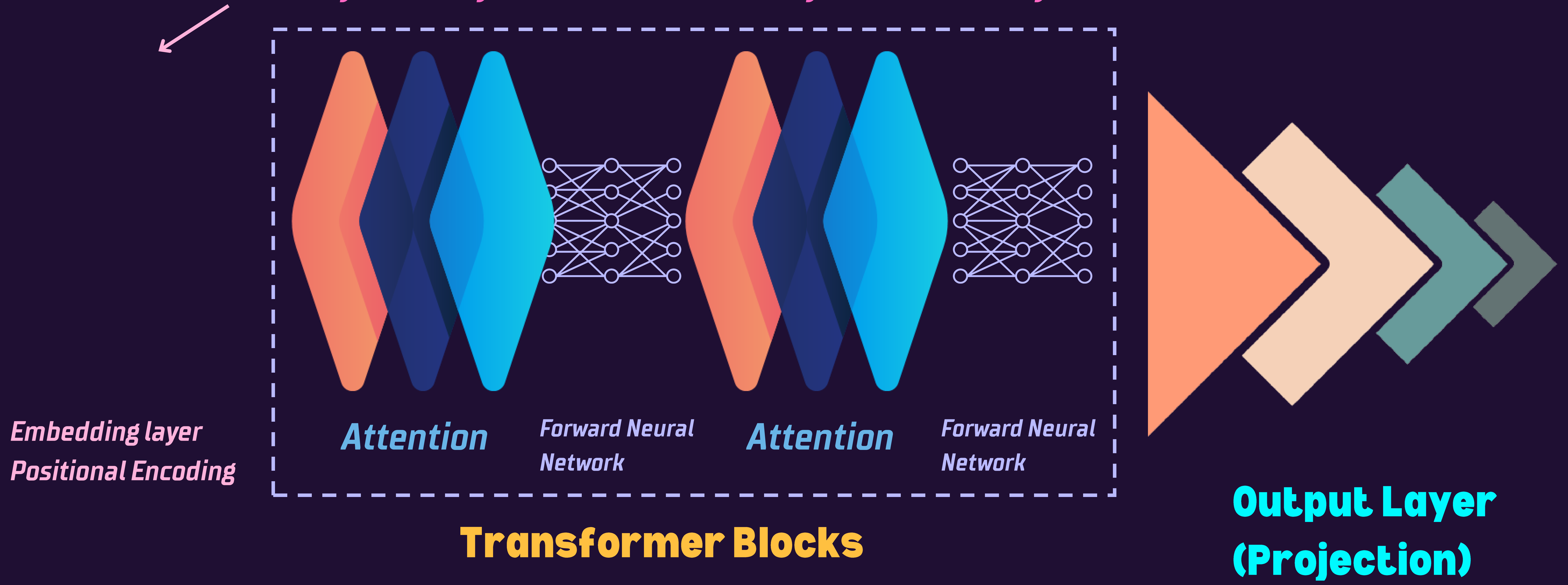
I don't care that they stole my idea. I care that they don't have any of their own.

Each head may learn to focus on different types of relationships between tokens (e.g., syntactic vs. semantic relationships).
The outputs of all heads are concatenated and then linearly transformed into the final output of the multi-head attention layer.

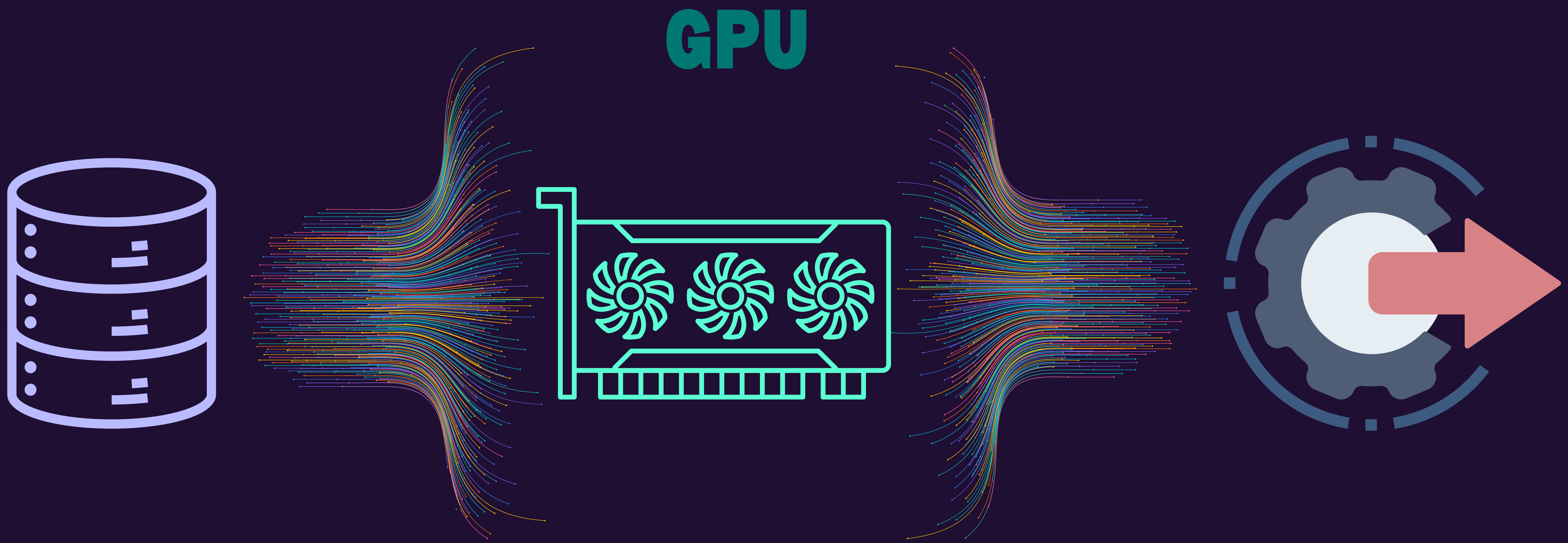
ADD & NORM

Transformer Model Pipeline

I don't care that they stole my idea. I care that they don't have any of their own.



Extremely Parallelizable Architecture



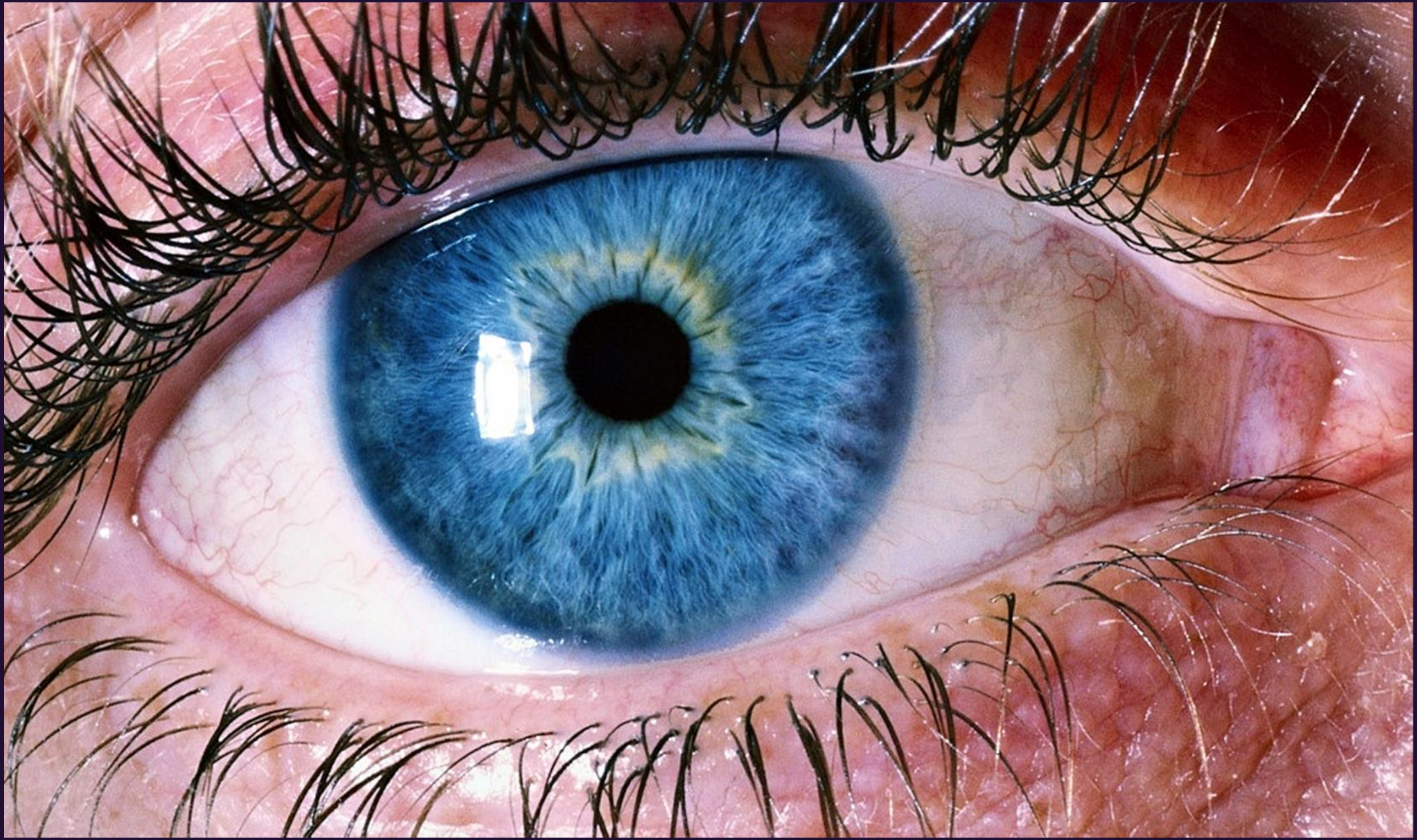
Transformers Recent Development

- BERT (Bidirectional Encoder Representations from Transformers): Pre-trained on large-scale unlabeled text data, a wide range of NLP tasks.
- GPT (Generative Pre-trained Transformer): GPT models have pushed the boundaries of language generation: text completion and dialogue generation.
- Vision Transformers (ViT): Computer vision: image classification and object detection.

Aspect	RNNs (Recurrent Neural Networks)	Transformers
Architecture	Sequential processing with hidden states	Parallel processing with self-attention
Long-term Dependencies	Struggle with long-term dependencies due to vanishing gradients	Excel at capturing long-range dependencies through self-attention
Computational Efficiency	Sequential processing, can be computationally expensive for long sequences	Parallel processing, more efficient, especially for longer sequences
Positional Information	Inherently capture positional information through sequential processing	Require explicit positional encodings for parallel processing
Training	Challenging to train due to vanishing and exploding gradients	Easier to train and optimize, enabling deeper and larger models
Contextual Understanding	Capture contextual information within the sequence	Capture contextual information globally through self-attention
Scalability	Limited scalability due to sequential processing	Highly scalable, allowing for training on large datasets and handling longer sequences

Semantic Medical Image Segmentation

Iris nevus vs melanoma



Semantic Medical Image Segmentation

Iris nevus vs melanoma

- Data Acquisition
- Data Processing
- Model Architecture
- Training
- Evaluation
- Post-processing
- Deployment and Application

Generative CAD Design with FEA for 3D Printing Optimization

- In Fusion 360, input design goals, materials, manufacturing methods…
- Generative Adversarial Networks (GANs)
- Variational Autoencoders (VAEs)
- …

Finite Element Analysis (FEA):

- Numerical simulation technique to analyze structural behavior under loading conditions.
- Divides design into smaller elements and solves equations to predict stress, strain, and deformation.
- Identifies areas of high stress concentration, potential failure points, and optimization opportunities.
- Fusion 360 includes built-in FEA tools for design simulation and analysis.

SensiScyther – embedded AI

- Picovoice (Porcupine + Rhino)
- Mozilla DeepSpeech
- CMSISNN

AI-Driven Synthesis Workflow

Natural Language Processing

Parameter Mapping

On-Device AI Inference


Thank you!
BE NAIVE
RAN'S LAB PRESENTS
RAN YANG
2024