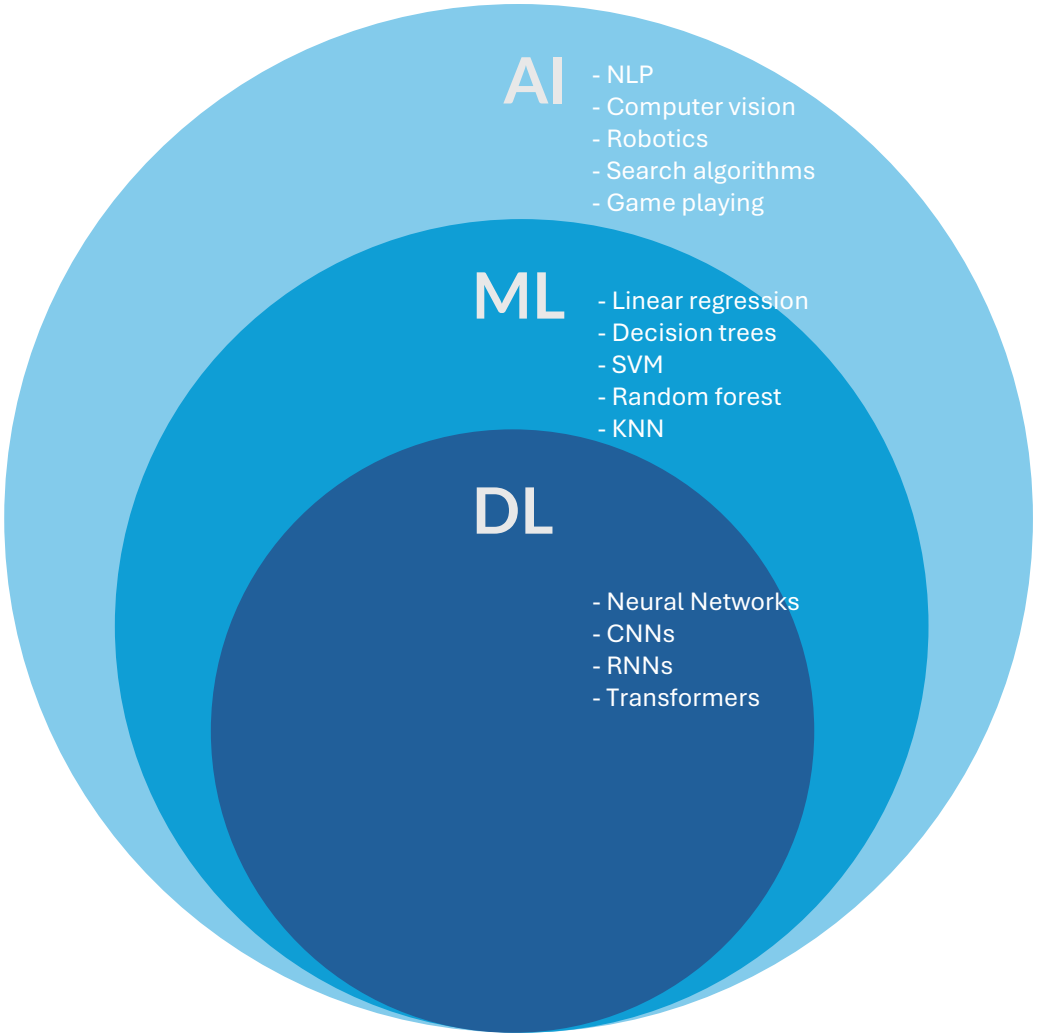
AI Basics
Neural Networks

The Attack Plan

- Touch of theory
- Basic Models
  - Linear Regression
  - Logistic Regression
  - Demo Code
- Neural Networks
  - Biological Inspiration
  - Perceptron
  - Bias and Weights
  - Training NN
  - Manual Calculation
  - Problem with perceptron
  - Activation Functions
  - Training Network
  - Backpropagation
- CNNs – Working with images
  - Convolution
  - Filters
  - Pooling
  - Flattening
  - FCL (MLP)
- RNNs – Sequences
  - Common Architectures
  - Vanishing/Exploding Gradient
- LSTM – Memory in NN
- Encoders & Decoders
  - Use cases
  - Anomaly Detection Demo

Touch of Theory



Artificial Intelligence (AI) Artificial Intelligence is the broad field of computer science focused on creating machines that can perform tasks typically requiring human intelligence. AI doesn't necessarily require learning from data - it can also use pre-programmed logic, search algorithms, and symbolic reasoning to solve problems and exhibit intelligent behaviour.

Machine Learning (ML) Machine Learning is a subset of AI that focuses on creating systems that automatically learn and improve their performance through experience, without being explicitly programmed for every specific task. Instead of following pre-written rules, ML algorithms analyse data to identify patterns, make predictions, or classify information.

Deep Learning (DL) Deep Learning is a specialised branch of machine learning that uses artificial neural networks with multiple layers (typically three or more) to model complex patterns in data. Inspired by the structure of the human brain, these "deep" networks can automatically discover intricate features and representations in data without manual feature engineering.

Touch of Theory - AI

Natural Language Processing (NLP) - This involves teaching computers to understand, interpret, and generate human language. Examples include chatbots, language translation (Google Translate), sentiment analysis of social media posts, voice assistants (Siri, Alexa), and text summarisation.

Computer Vision - This field enables machines to "see" and interpret visual information from the world. Applications include facial recognition systems, medical image analysis (such as detecting tumours in X-rays), autonomous vehicle navigation, quality control in manufacturing, and augmented reality filters on social media.

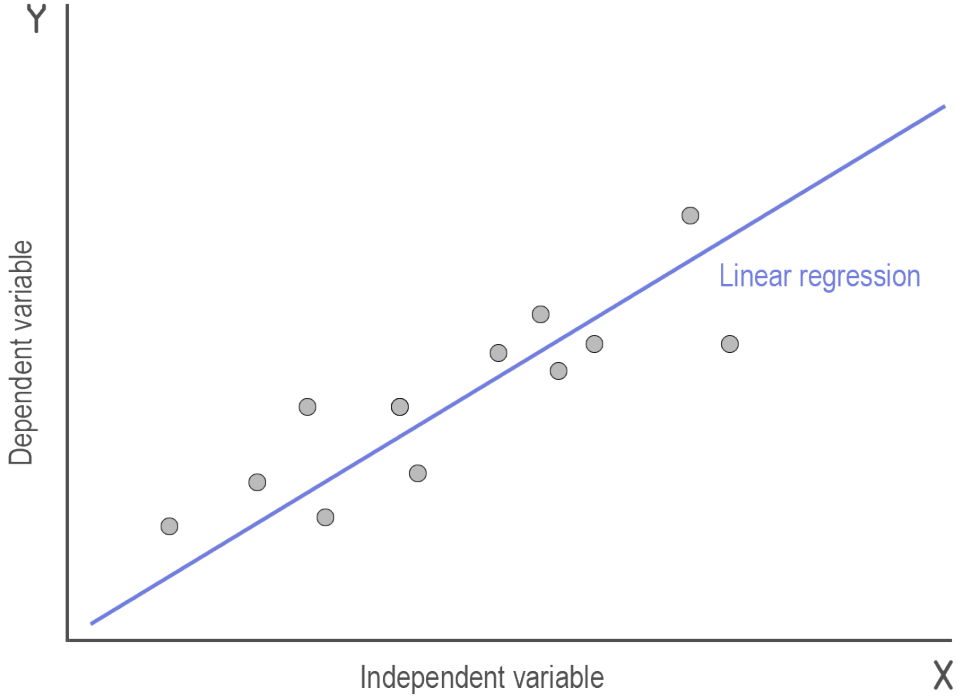
Robotics - This combines AI with mechanical engineering to create machines that can interact with the physical world. Examples range from industrial assembly robots and surgical robots to household vacuum cleaners (Roomba) and delivery drones. Often incl. other AI fields – like: computer vision to navigate, NLP to understand voice commands.

Search Algorithms - These are methods for finding optimal solutions or paths through large problem spaces. Examples include GPS navigation finding the shortest route, web search engines ranking relevant results, and game AI exploring possible moves. Search algorithms form the backbone of many AI systems, enabling them to efficiently explore possibilities and find effective solutions.

Game Playing - This was one of the earliest AI applications, where computers learn to play games at superhuman levels. Famous examples include Deep Blue's victory over chess champion Garry Kasparov, AlphaGo's mastery of the ancient game of Go, and modern AI's success in complex video games like **StarCraft II -** DeepMind (owned by Google/Alphabet – reinforced learning). Game-playing AI often combines search algorithms with learning techniques to develop sophisticated strategies.

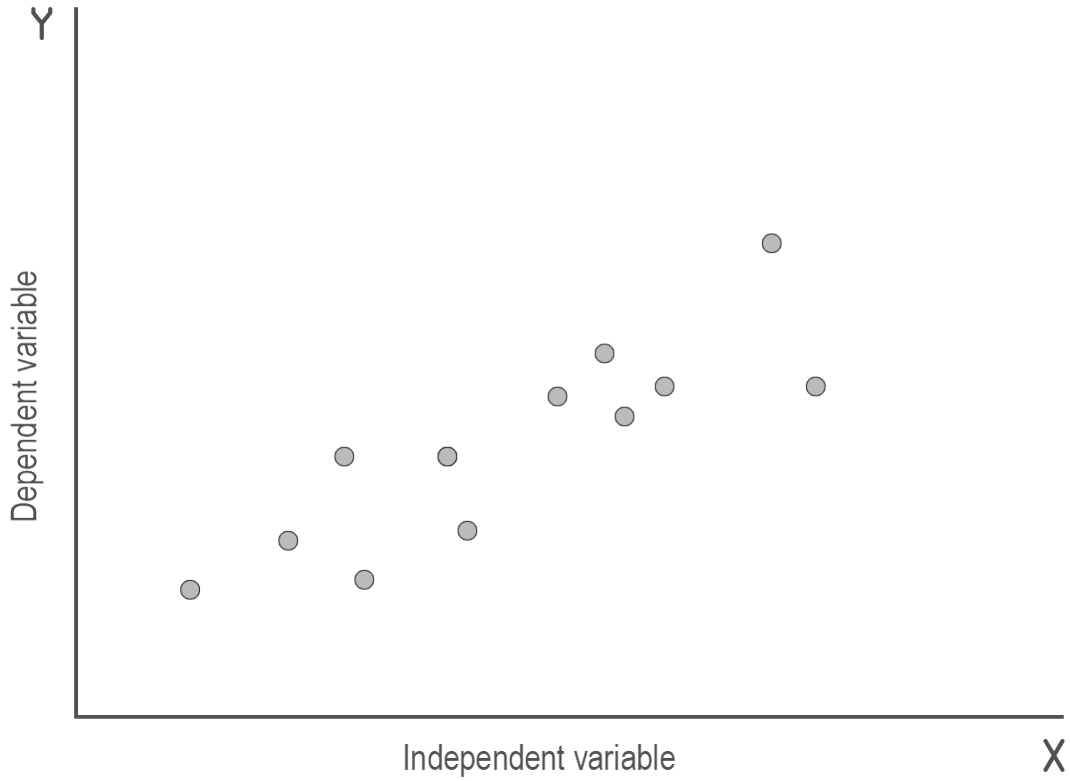
Touch of Theory – Basic Models – Linear Regression

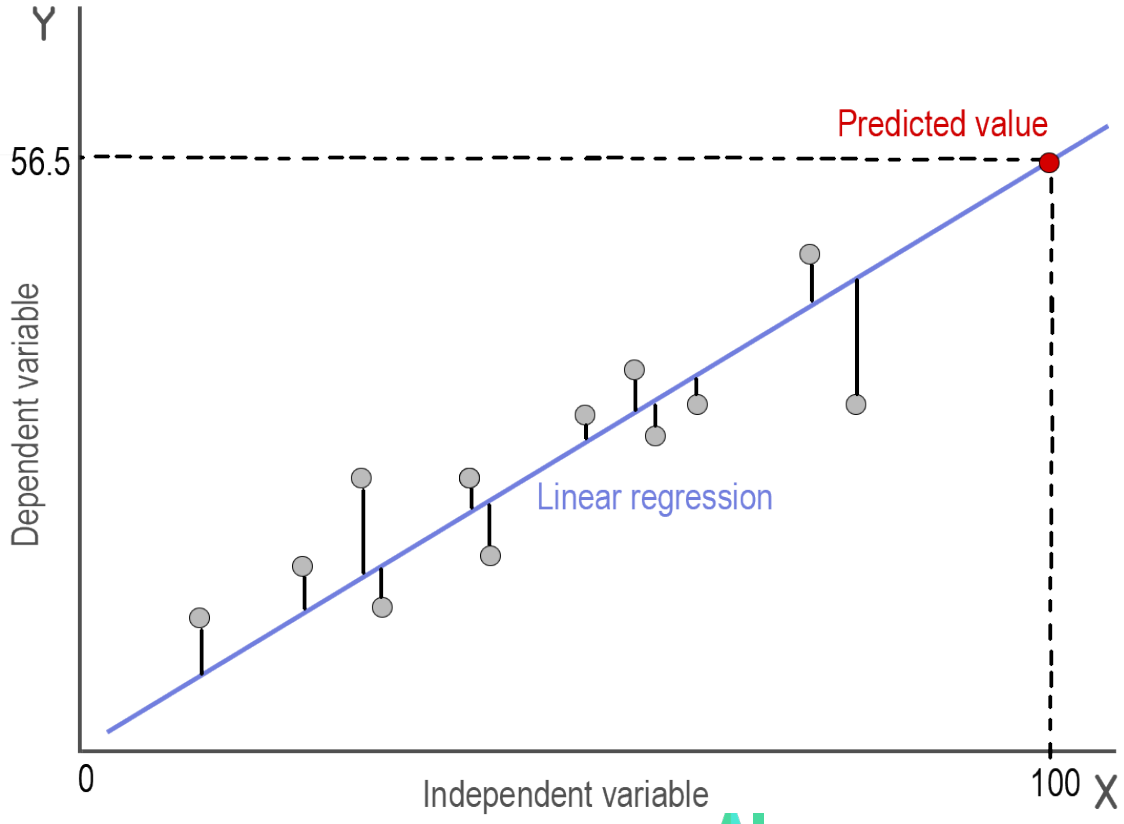
Linear Regression - This is one of the simplest and most widely used algorithms in Machine Learning. It models the relationship between a dependent variable and one or more independent variables by fitting a straight line. Commonly used for predicting numerical values - like housing prices, sales forecasts, or risk scores - linear regression provides a good starting point for understanding more complex models. It is a supervised machine learning model and a statistical model. **Predicts continuous numeric values**. Concept has existed since 1805! Wow !



Touch of Theory – Basic Models – Linear Regression

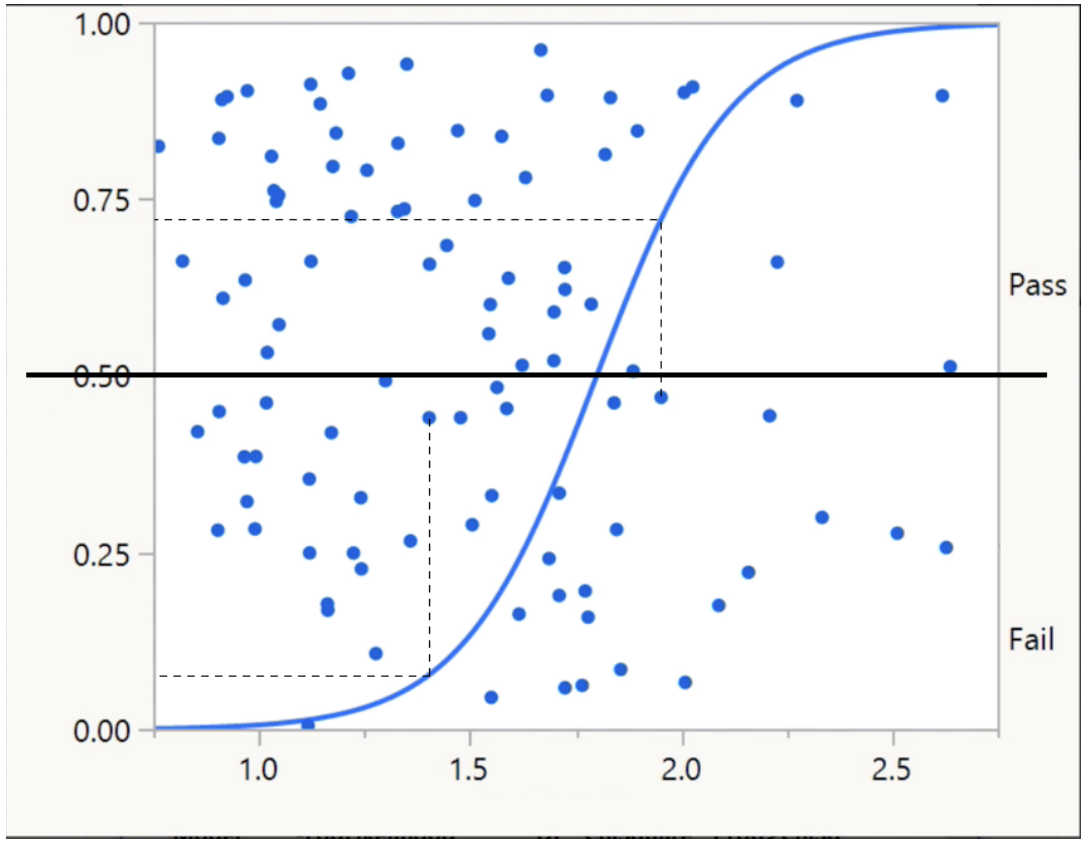
The best-fitted line in linear regression is the line that minimises **the sum of squared residuals** (differences between actual and predicted values). This is determined using the least squares method, which calculates the optimal slope and intercept that minimise the total squared error between the line and all data points.





Touch of Theory – Basic Models – Logistic Regression

Logistic Regression - Despite its name, this is actually a classification algorithm rather than a regression technique. It models the probability that an instance belongs to a particular category by using the logistic function to map any real-valued input to a value between 0 and 1. Commonly used for binary classification problems - like email spam detection, medical diagnosis, or customer churn prediction - logistic regression extends the linear approach to handle categorical outcomes. It is a supervised machine learning model and a statistical model. **Predicts probabilities and categorical outcomes. Binary Classification.** More classes? KNN, SVM, **NN**, Decision Trees, Random Forests. Concept has existed since 1838! Wow x 2 !



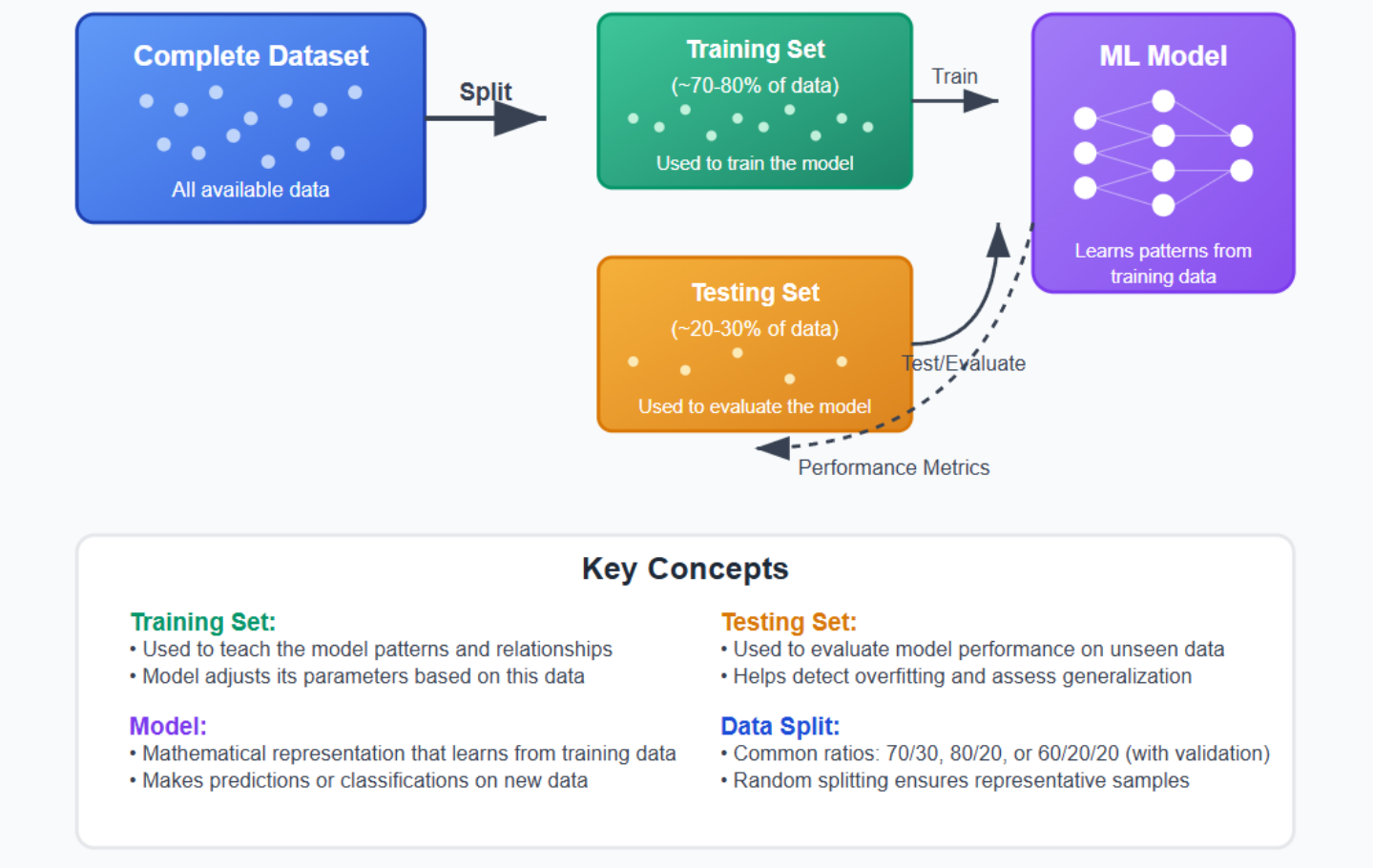
Neural Networks – Why Linear and Logistic Regression for NN ?

1. **Both are the Building Blocks of Neural Networks**
   - A single-layer neural network with no activation function is essentially linear regression.
   - A single-layer neural network with a sigmoid activation is logistic regression**.**
2. **Math - Both linear and logistic regression use:**
   - Weights and biases (w and b)
   - Dot product operations (w * x + b)
   - Gradient descent to minimise a loss function
3. **Training principles are the same**
   - A loss function (like MSE for regression, cross-entropy for classification)
   - Backpropagation, which is just a fancy chain of partial derivatives, built from the same derivatives you see in linear/logistic regression
4. **Both represent core use cases**
   - Linear regression: continuous output → think house prices, temperatures, etc.
   - Logistic regression: binary classification → think spam detection, tumour benign/malignant, etc.
5. **Interpretability**

   While neural networks are often black boxes, linear and logistic regression are transparent. Many machine learning beginners are introduced to these models first so they can:
   - Understand how features affect output
   - Learn how model training works
   - Build intuition before diving into deeper models

DEMO APPS

Training Models and Labelled Data



LABELS

Labels refer to the **output or target values** that you're trying to predict or analyse. They are the **known results** you use during training in **supervised learning**.

Example:

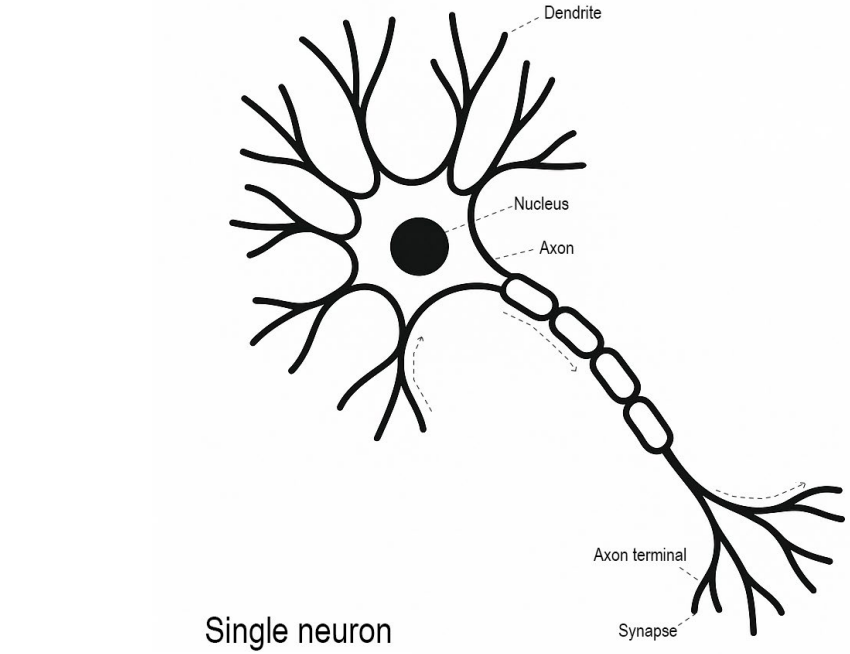
Image 1 - cat <- label
Image 2 - dog <- label
Image 3 - bird <- label

Neural Networks – Biological Inspiration

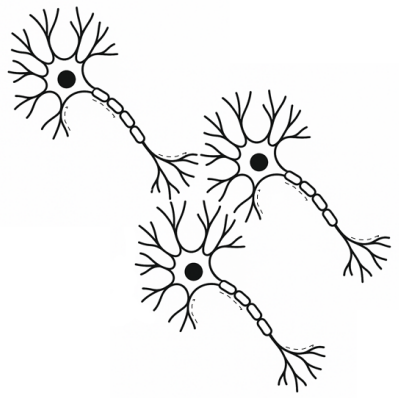
The concept of neural networks in artificial intelligence is deeply inspired by the structure and function of the human brain. At the core of this inspiration is the biological neuron - a specialised cell that transmits electrical signals through a network of connections. Each neuron receives inputs through its dendrites, processes the information in the cell body (soma), and sends output signals through its axon to other neurons via synapses. This complex web of interconnected neurons allows the brain to learn, adapt, and make decisions. Similarly, artificial neural networks attempt to mimic this process by using interconnected artificial neurons (or nodes) that process and transmit information in layers, enabling machines to recognise patterns, learn from data, and perform intelligent tasks.

~ 86 billion neurons in a human brain

~ 100 trillion synapses

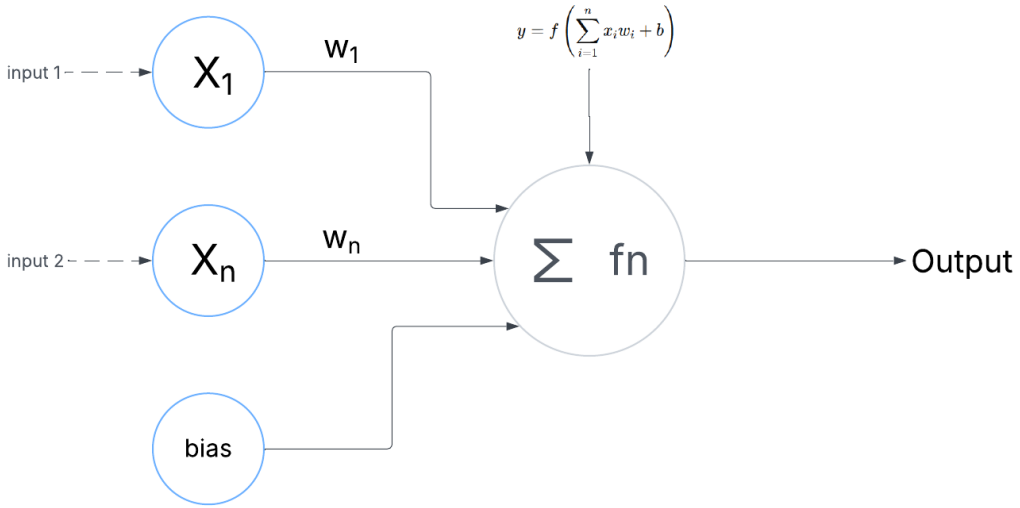


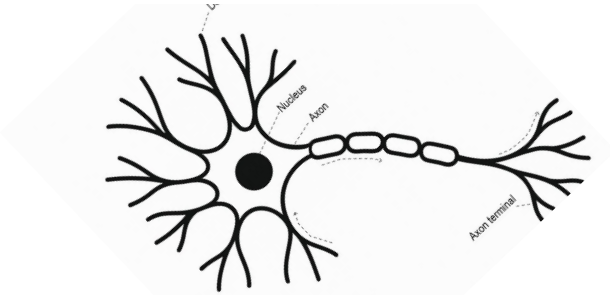
Single neuron



Neural Networks - Perceptron

The **perceptron** is the simplest type of artificial neural unit, introduced by **Frank Rosenblatt** in 1958 as a model for binary classification. It mimics a basic neuron by taking multiple input values, each multiplied by an associated **weight**, summing them up along with a **bias**, and passing the result through an **activation function -** typically a **step function -** to produce a binary output (0 or 1). If the weighted sum exceeds a certain threshold, the perceptron "fires" (outputs 1); otherwise, it doesn't (outputs 0). While the perceptron is foundational in neural network theory, it has a major limitation: it can only solve problems that are **linearly separable**. For example, it cannot correctly learn the **XOR function**, which cannot be represented by a single straight line. This limitation highlighted the need for **multi-layer architectures**, leading to the development of more powerful neural networks.





Neural Networks – Perceptron – Bias and Weights

Bias allows the neuron to activate even when all inputs are zero. It helps the network learn patterns more effectively. Without bias, the model would be much less flexible and powerful.

Biases and **weights** in a neural network are typically assigned **random values initially** when training begins. **Bias is a fine-tuning adjustment**

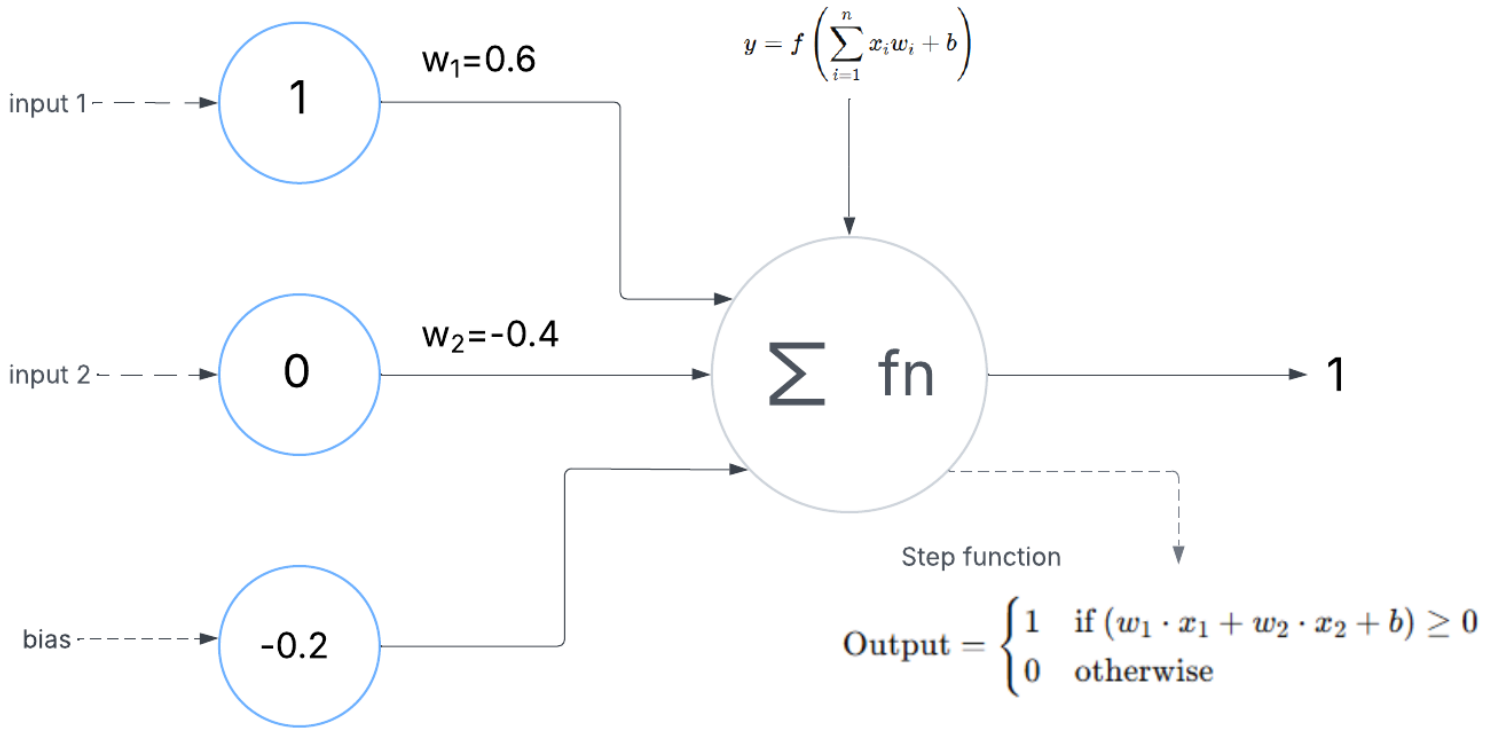
- Random initialisation helps break symmetry.
- If all neurons started with the same weights and biases, they would all learn the same thing, making the network ineffective.
- Starting with different random values allows each neuron to learn different features from the data during training.

What are weights?

Weights are the 'knowledge' of the network.

Each weight controls how important an input is to a neuron. During training, the network adjusts weights to reduce error - **this is how it learns**. The better the weights match the patterns in the data, the better the network performs.

Neural Networks – Perceptron – Manual Calculation 1/2

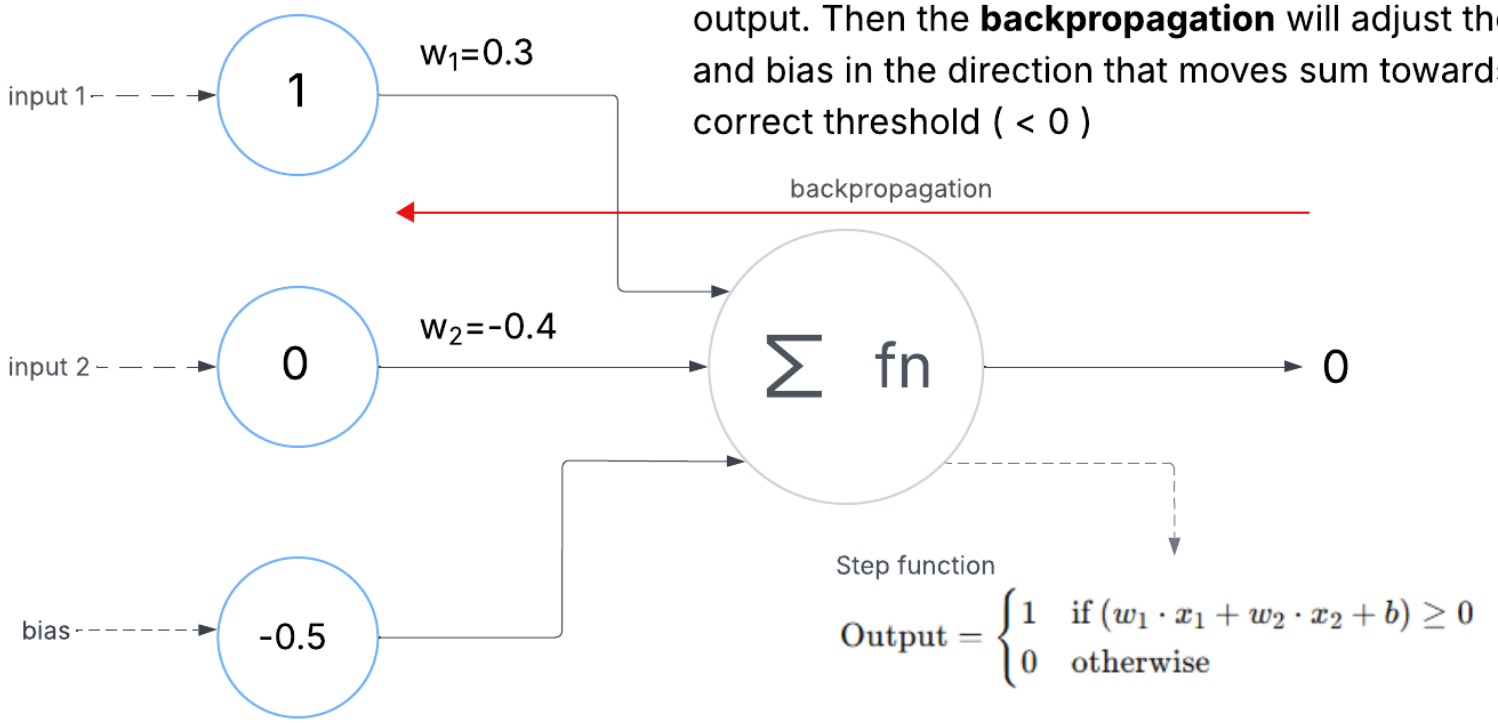


Sum = (0.6 * 1) + (-0.4 * 0) + (-0.2) = 0.6 + 0 - 0.2 = 0.4

Since 0.4 >= 0 then the output is 1

fn(0.4) = 1

Neural Networks – Perceptron – Manual Calculation 2/2 - Backpropagation



Same example but imagine that network expects 0 as the output. Then the **backpropagation** will adjust the weights and bias in the direction that moves sum towards the correct threshold (< 0)

Sum = (0.3 * 1) + (-0.4 * 0) + (-0.5) = 0.3 + 0 - 0.5 = -0.2

Since -0.2 < 0 then the output is 0

fn(-0.2) = 0

Neural Networks – Why ? Problem & XOR

A single-layer perceptron can only solve problems that are linearly separable, meaning you can draw a straight line to separate the outputs (0s from 1s).

But XOR is not linearly separable. You can't draw one straight line to divide the 1s and 0s in XOR.

XOR examples in data:

Input A: Patient has symptom X
Input B: Patient has symptom Y
Output: Disease is present only if **exactly one** of the symptoms is present, not both or none.

Drug A and **Drug B** each help on their own.
But taken together, they cancel each other's effect or cause side effects.
The treatment **is successful only** if a single drug is used.

AND

	T	F
T	T	F
F	F	F

OR

	T	F
T	T	T
F	T	F

XOR ⊕

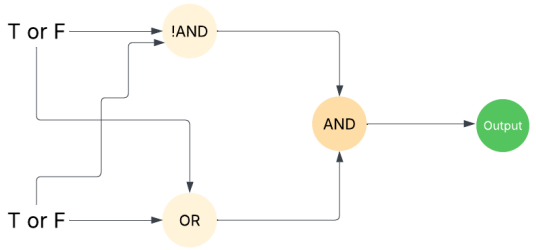
	T	F
T	F	T
F	T	F

XOR (Exclusive OR) - impossible to draw one line to separate T and F

XOR is OR but !AND
XOR(A,B)=(A OR B) AND ¬(A AND B)

Basic Logical Operators

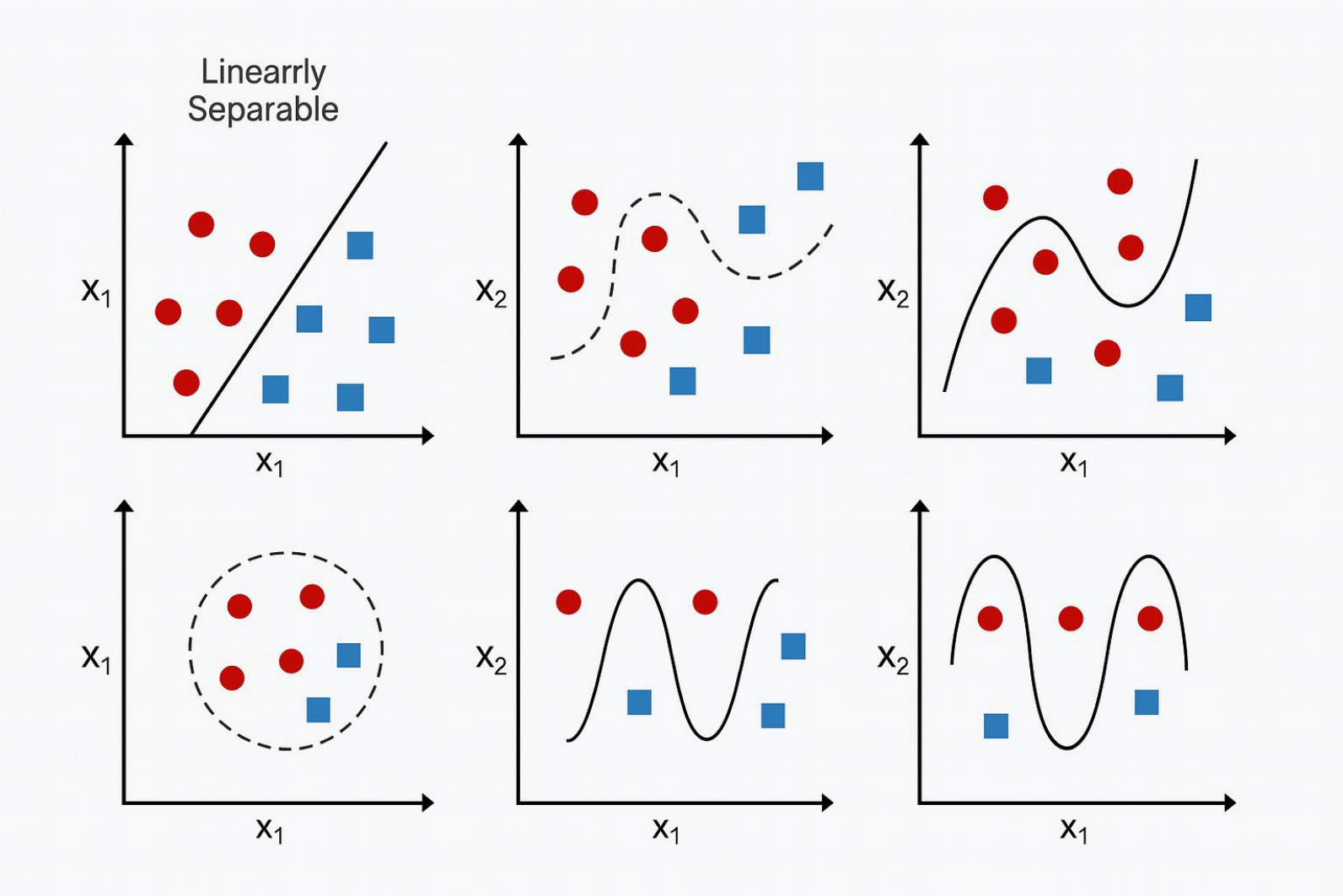
AND and OR are linearly separable
XOR isn't



Input A: Chemical A is reactive
Input B: Chemical B is reactive
Together, they might cancel out or form a stable compound, but **separately**, they cause danger.

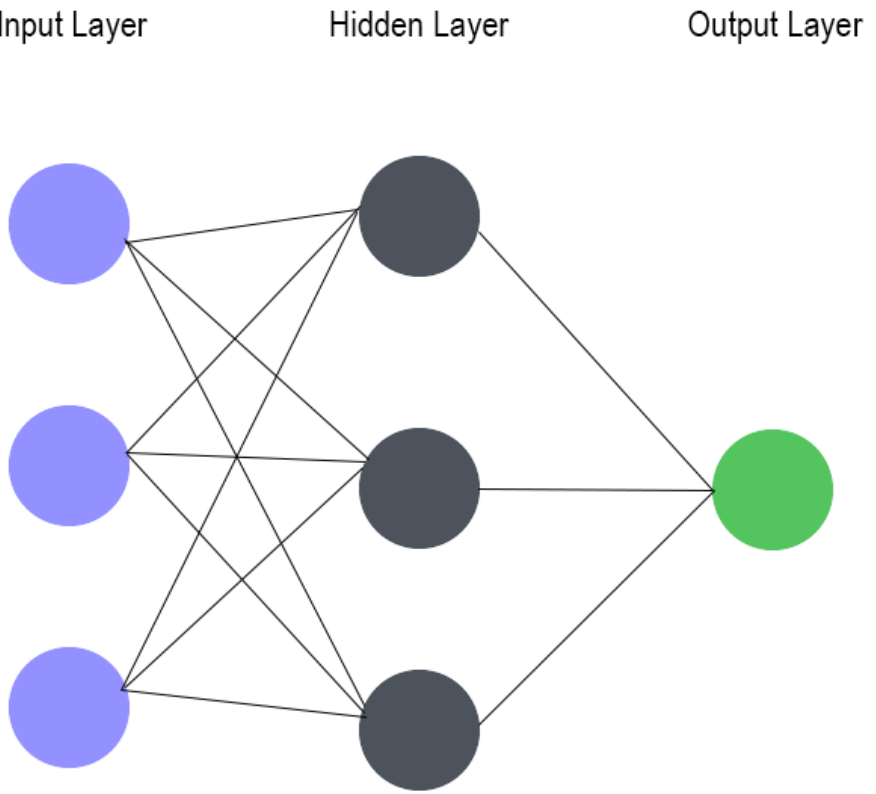
Input A: Gene X is dominant
Input B: Gene Y is recessive
Certain traits appear only when **one gene** is active, but not when both are active.

Neural Networks – Non-Linearly Separable Data

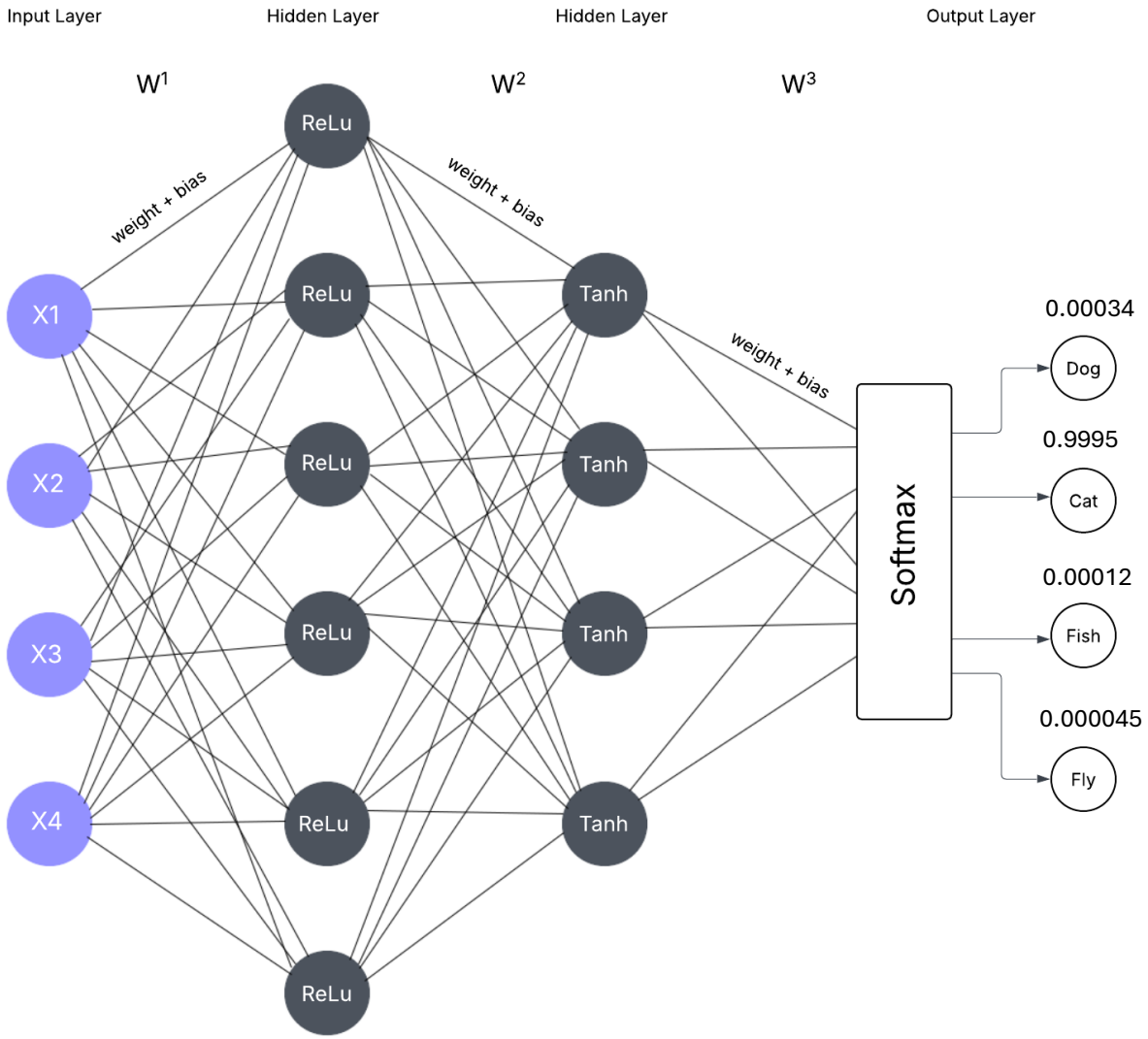


Neural Networks - Basics

- Neural networks are inspired by the human brain.
- Composed of **layers**:
  - **Input layer** – receives data
  - **Hidden layers** – perform computations
  - **Output layer** – returns predictions
- Each **neuron** computes a **weighted sum** of its inputs.
- An **activation function** (e.g. ReLU, Sigmoid) adds non-linearity.
- The network **learns** by adjusting its weights during training.



Neural Networks – Simple Network



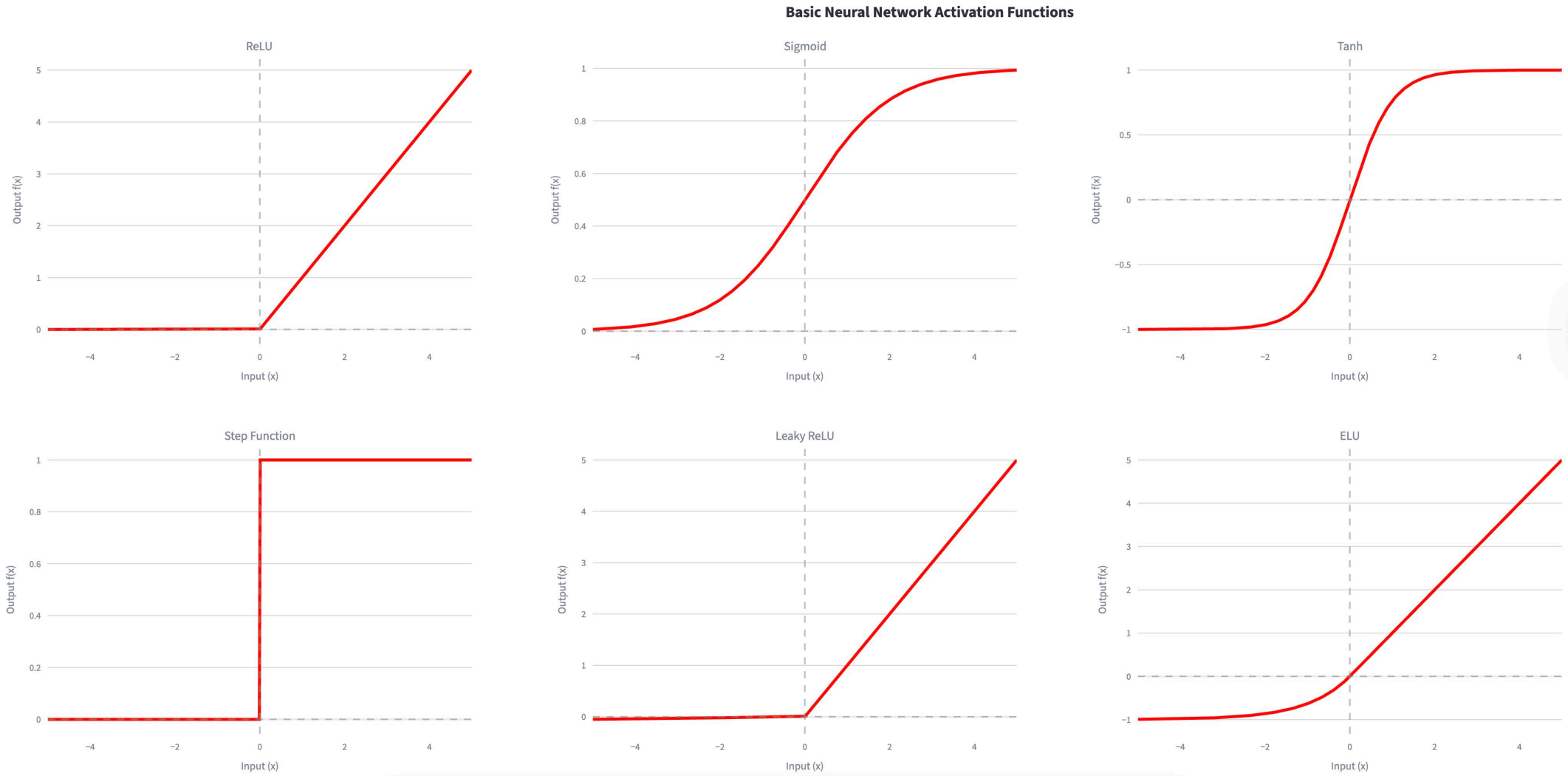
Simple deep network with two hidden layers. Each layer contains a full neuron with an activation function.
The last layer is an output layer with a Softmax function that allows for the output of probabilities for classes.

Activation function – it must be a non-linear function. It allows learn complex, non-linear patterns. Solving real-world problems. Creates decision boundaries that are curved, layered, or multidimensional – not just lines. They can also handle linear problems, so no worries!

Most common linear functions:
ReLU – Most popular, simple and fast
Sigmoid – Good for probabilities, but causes vanishing gradient
Tanh – Zero-centred, used in RRNs
Softmax – Used in the output layer for classification

DEMO APP

Neural Networks – Activation Functions



Activation Functions – Demo App

DEMO APP

Neural Networks – Training Network

Training a neural network is the process of teaching it to make accurate predictions by learning from data. It begins with random weights and biases, and then the network makes a prediction using its current parameters. The result is compared to the correct answer using a **loss function**, which measures how wrong the prediction was. Then, using a method called **backpropagation**, the network calculates how much each weight and bias contributed to the error. Finally, an optimisation algorithm like **gradient descent** adjusts the weights and biases slightly to reduce the error. This cycle - predict, compare, adjust - is repeated multiple times with numerous examples until the network's predictions become as accurate as possible. Over time, the network "learns" by fine-tuning its internal parameters to capture patterns in the training data.

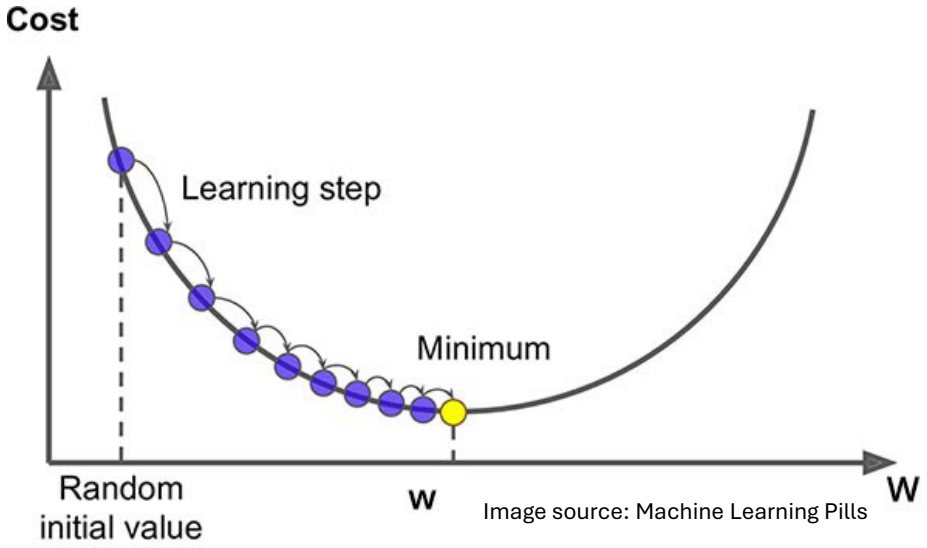


Image source: Machine Learning Pills

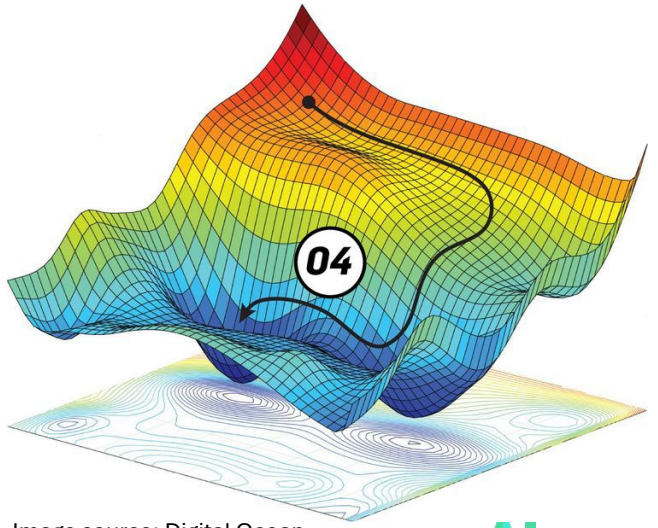
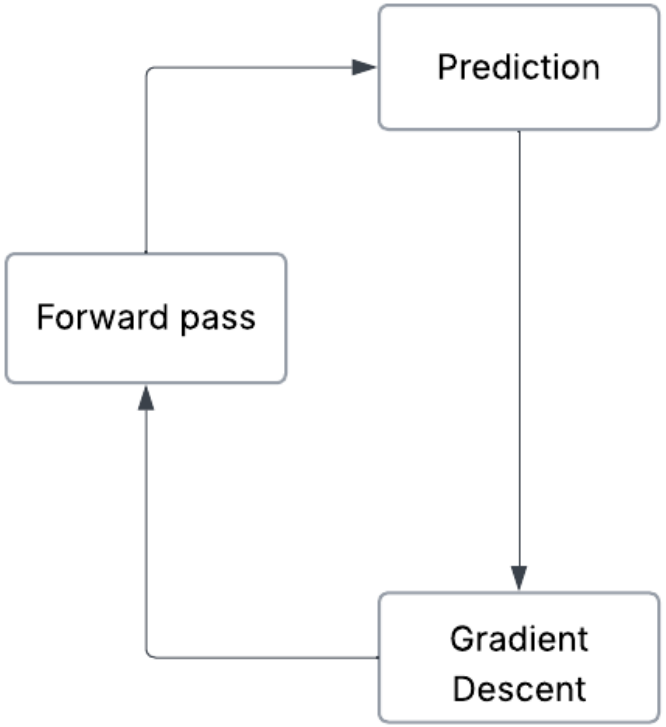


Image source: Digital Ocean

Neural Networks – Backpropagation

Neural networks learn by iteratively improving their predictions through a process called **backpropagation**:

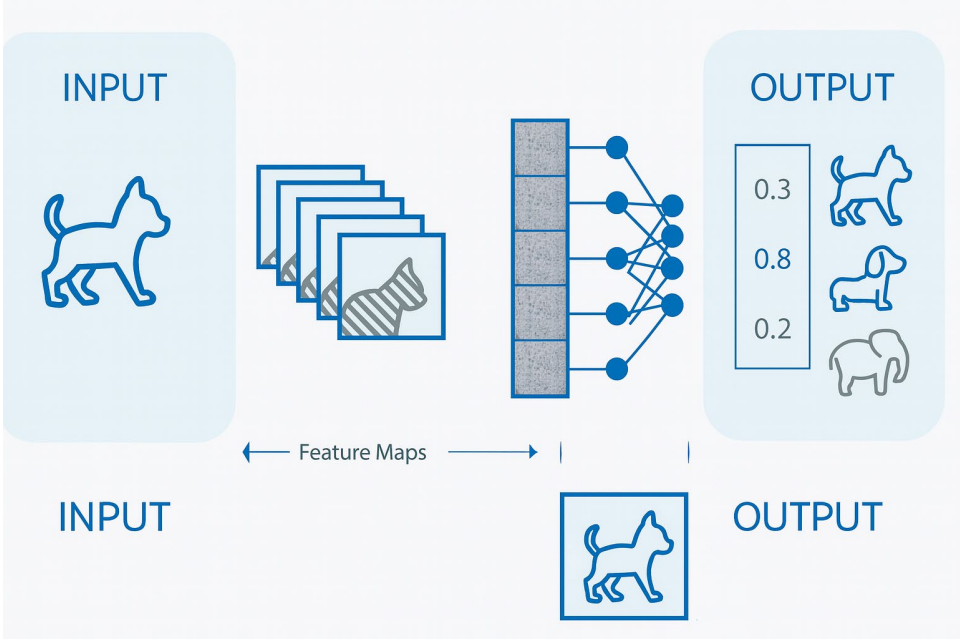
- **Forward Pass**: The Input data is passed through the network to generate a prediction.
- **Loss Function**: Compares the prediction to the actual target value to calculate the error.
- **Backward Pass**: The error is propagated backwards through the network to update the weights.
- **Gradient Descent**: Adjusts weights to minimise the error, step by step.
- **Repetition Over Time**: The process is repeated across many training examples, gradually improving the model's accuracy.



Neural Networks – CNNs – Working with images

A **Convolutional Neural Network (CNN)** is a specialised type of deep neural network designed to process data with a grid-like topology, such as images. Unlike traditional neural networks that treat all inputs equally, CNNs are structured to recognise spatial hierarchies in data by using convolutional layers that scan for patterns like edges, textures, or shapes. This makes them exceptionally effective at understanding visual content. CNNs are preferred for image-related tasks because they can automatically and efficiently learn **spatial features** without manual feature extraction. Their architecture also reduces the number of parameters compared to fully connected networks, making them more scalable and faster to train. CNNs power a wide range of real-world applications, including image classification, facial recognition systems, object detection in autonomous vehicles, and even medical image analysis for detecting diseases in X-rays and MRIs.

Spatial features refer to the **visual patterns and structures** found in data that are related to **position, shape, and arrangement** within space, especially in images.



Neural Networks – CNNs – Working with images

CNN Learning Progression: Like a Child Learning to Draw

Age 3	Age 5	Age 8	Age 12	Age 16	Professional
 Basic shapes	 Adding ears	 Better proportions	 Adding texture	 Adding shading	 Professional detail
Basic edges	Simple shapes	Better features	Textures	Complex patterns	Full recognition

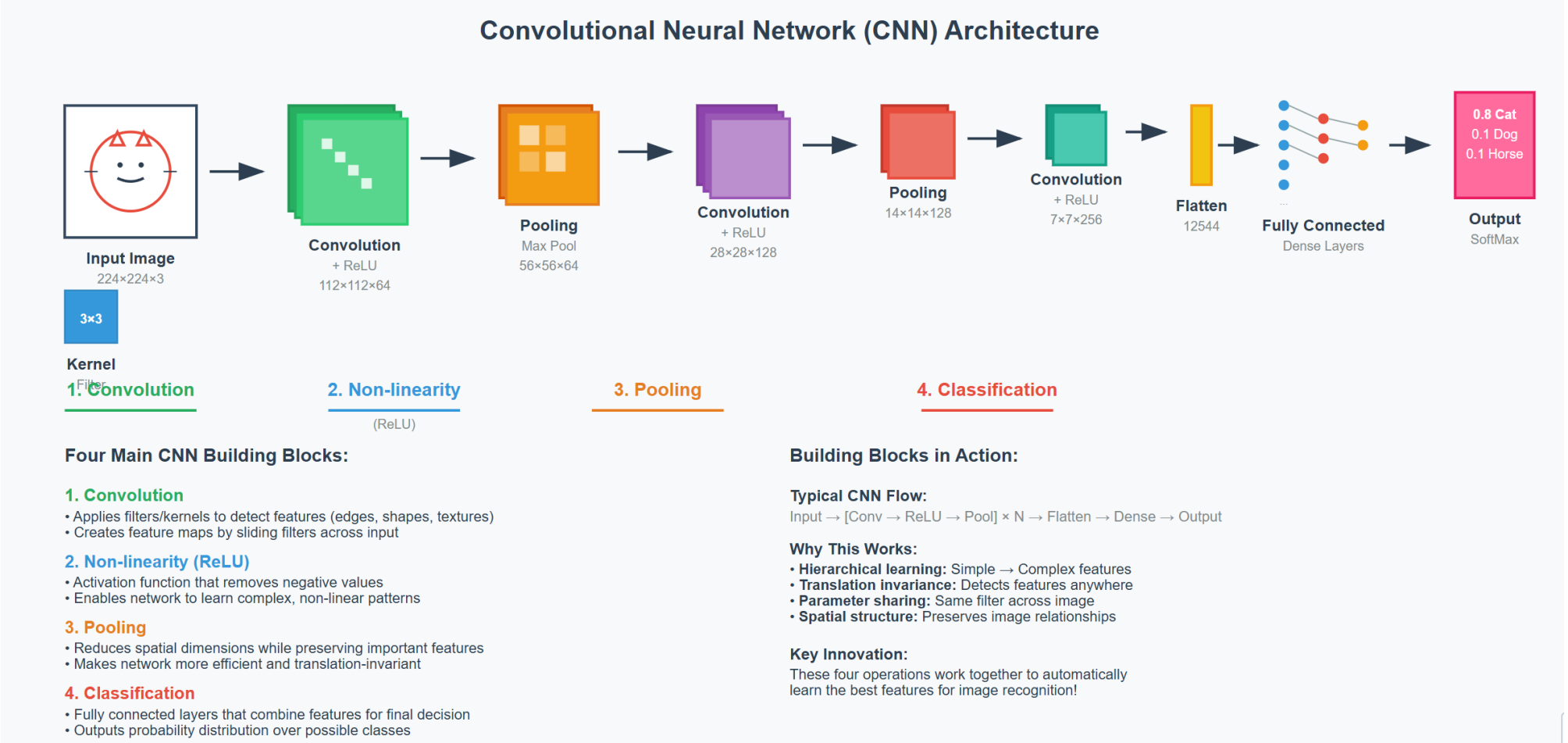
CNN Layer Progression:

Early layers detect edges and simple features → Middle layers combine into shapes → Deep layers recognize complex objects

Just like how a child's drawing skills develop from simple shapes to detailed, realistic representations!

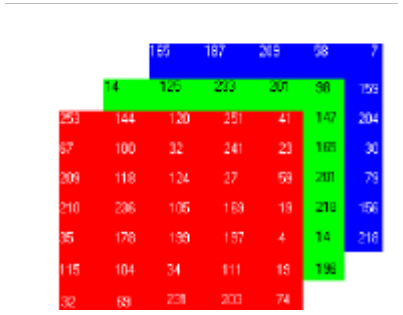
Neural Networks – CNNs – Working with images

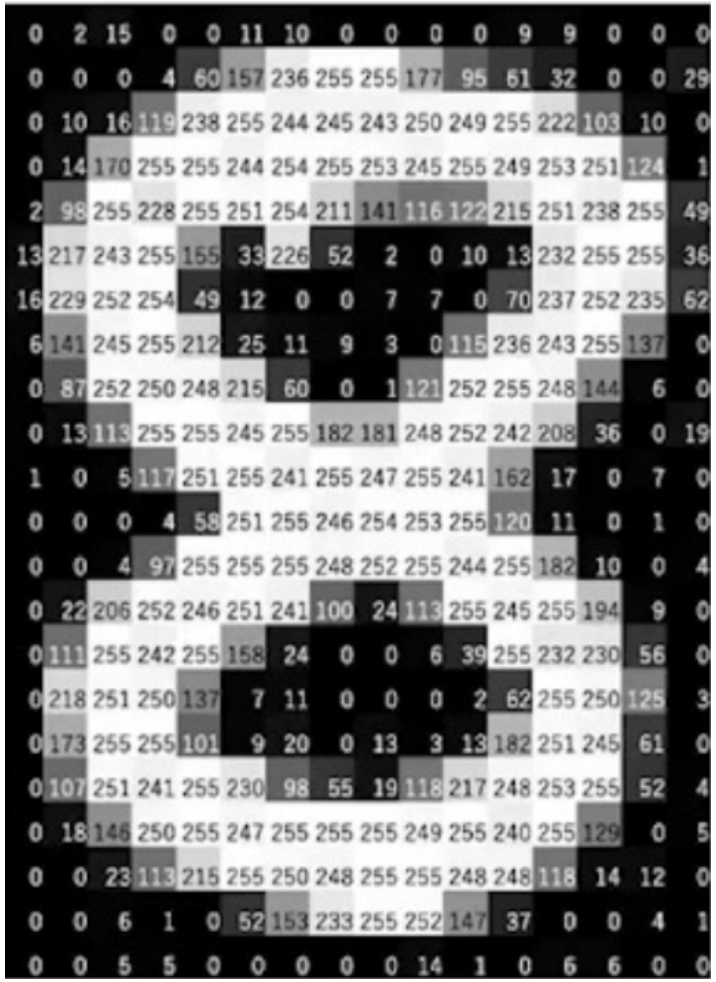
There are four main operations (Convolution, Non-linearity, Pooling and Classification) – building blocks



Neural Networks – CNNs – Working with images

- Every image can be represented as a matrix of pixel values.
- An image from a standard digital camera typically has three channels – red, green, and blue (Then the image is represented as a 3D array height x width x 3 channels [RGB])
- Three 2D-matrices stacked over each other (one for each colour), each having pixel values in the range 0 to 255.
- A grayscale image has only one channel.
  - A single 2D matrix representing an image.
  - The value of each pixel in the matrix will range from 0 to 255, with zero indicating black and 255 indicating white.

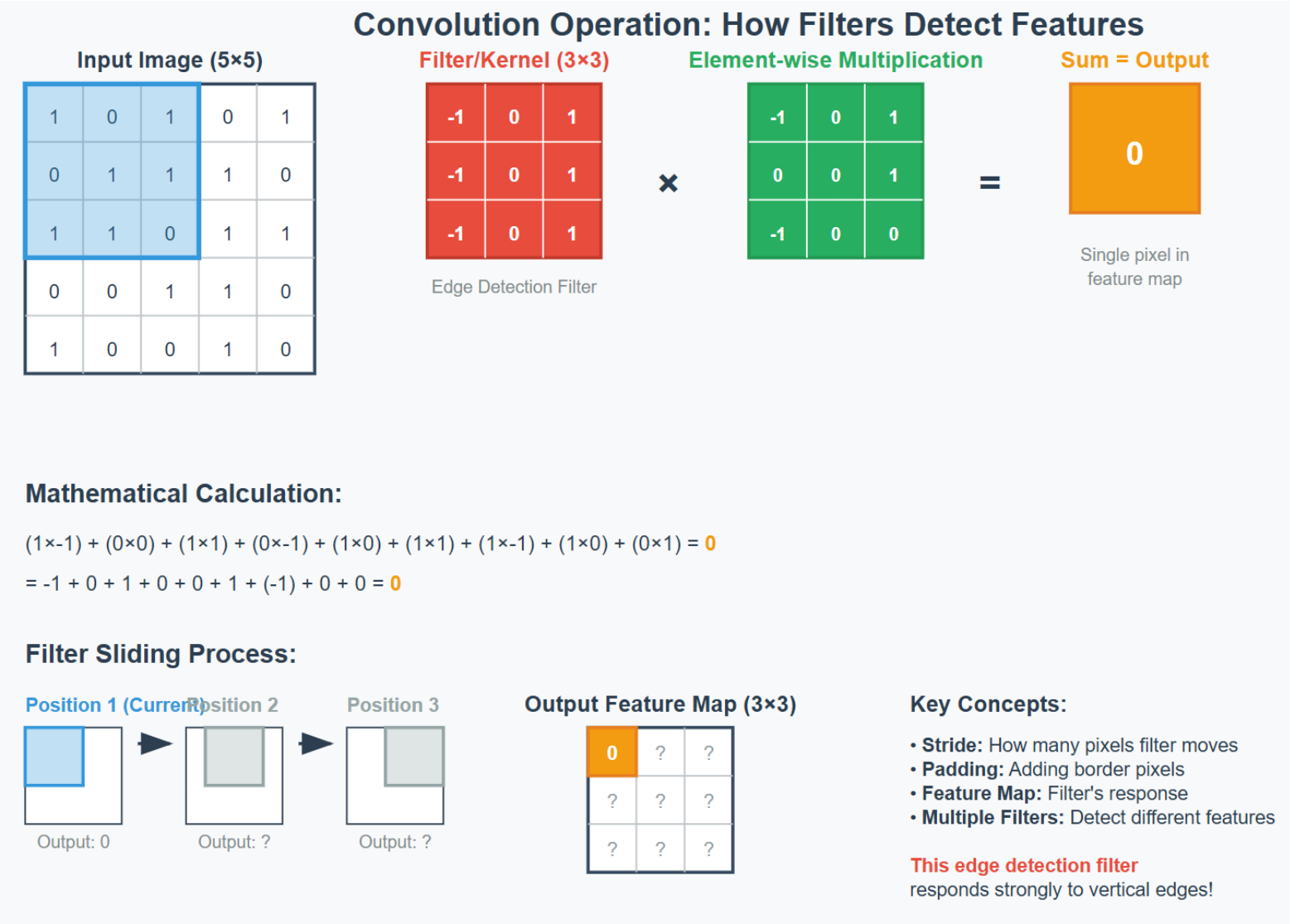




Neural Networks – CNNs – Convolution

In Convolutional Neural Networks (CNNs), **convolution** is the mathematical operation used to extract important features from an input image, such as edges, textures, or patterns. This process involves a small matrix, known as a filter (or kernel), which slides across the input image in steps (called a stride). At each position, it performs an **element-wise multiplication** between its values and the overlapping values in the image. The results are summed to produce a single number, which becomes part of a new output matrix called a **feature map** (or activation map). Each filter is designed to detect a specific kind of feature - some may highlight horizontal edges, others vertical or diagonal patterns. As CNNs stack multiple convolutional layers, they are able to learn increasingly complex features, going from simple edges in early layers to full objects or shapes in deeper layers. This ability to automatically learn and combine features makes convolution a powerful tool for image understanding.

Neural Networks – CNNs – Convolution



This image shows how **convolution** works in a CNN. A small **filter** (red) slides over the input image (blue), multiplies the values, and adds them up, placing the result into the **feature map** (orange). This example uses an **edge detection filter** that highlights vertical edges. Key terms such as **stride**, **padding**, and **multiple filters** enable the network to **detect** various **patterns**.

Neural Networks – CNNs – Convolution - Filter

Vertical Edge Detection

Input Image (8×8)

10	10	10	10	0	0	0	0
10	10	10	10	0	0	0	0
10	10	10	10	0	0	0	0
10	10	10	10	0	0	0	0
10	10	10	10	0	0	0	0
10	10	10	10	0	0	0	0
10	10	10	10	0	0	0	0
10	10	10	10	0	0	0	0

Edge between columns 4 and 5

3×3 Vertical Edge Filter

1	0	-1
1	0	-1
1	0	-1

Detects dark→light edges

—

Output Feature Map (6×6)

0	0	30	30	0	0
0	0	30	30	0	0
0	0	30	30	0	0
0	0	30	30	0	0
0	0	30	30	0	0
0	0	30	30	0	0

Columns 3-4: Strong edge response

Neural Networks – CNNs – Convolution - Filter

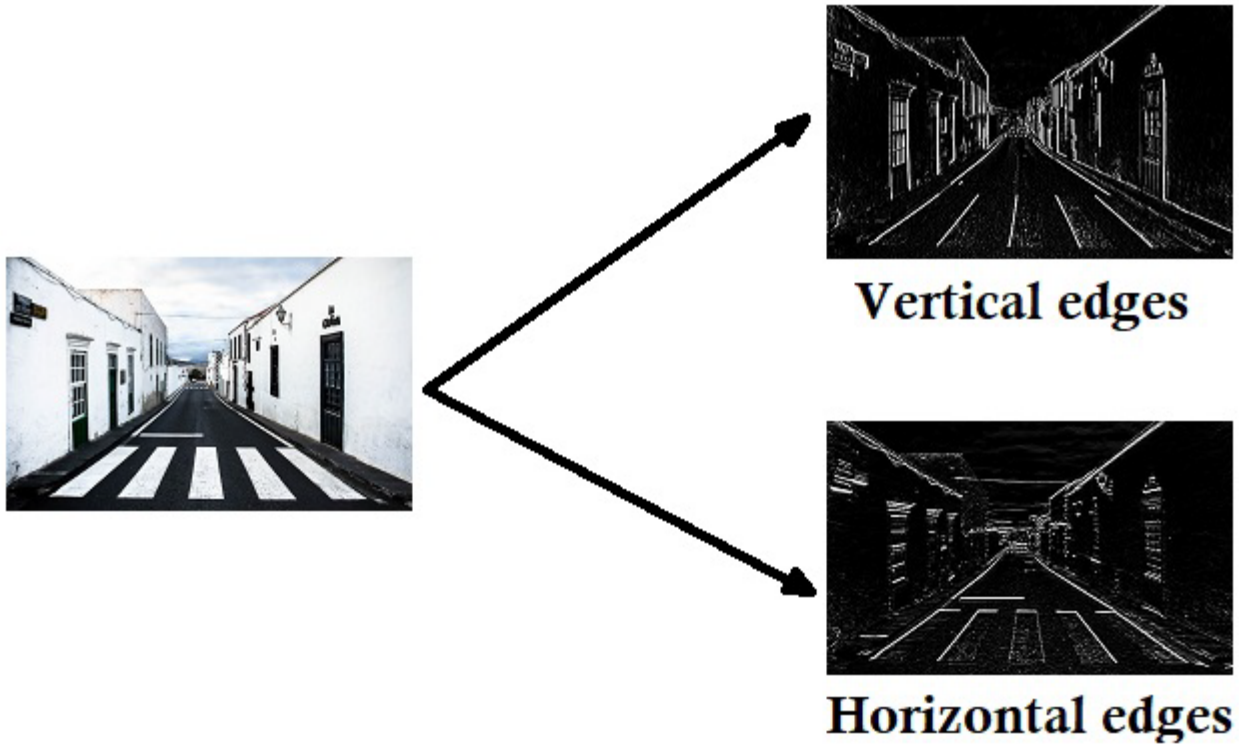


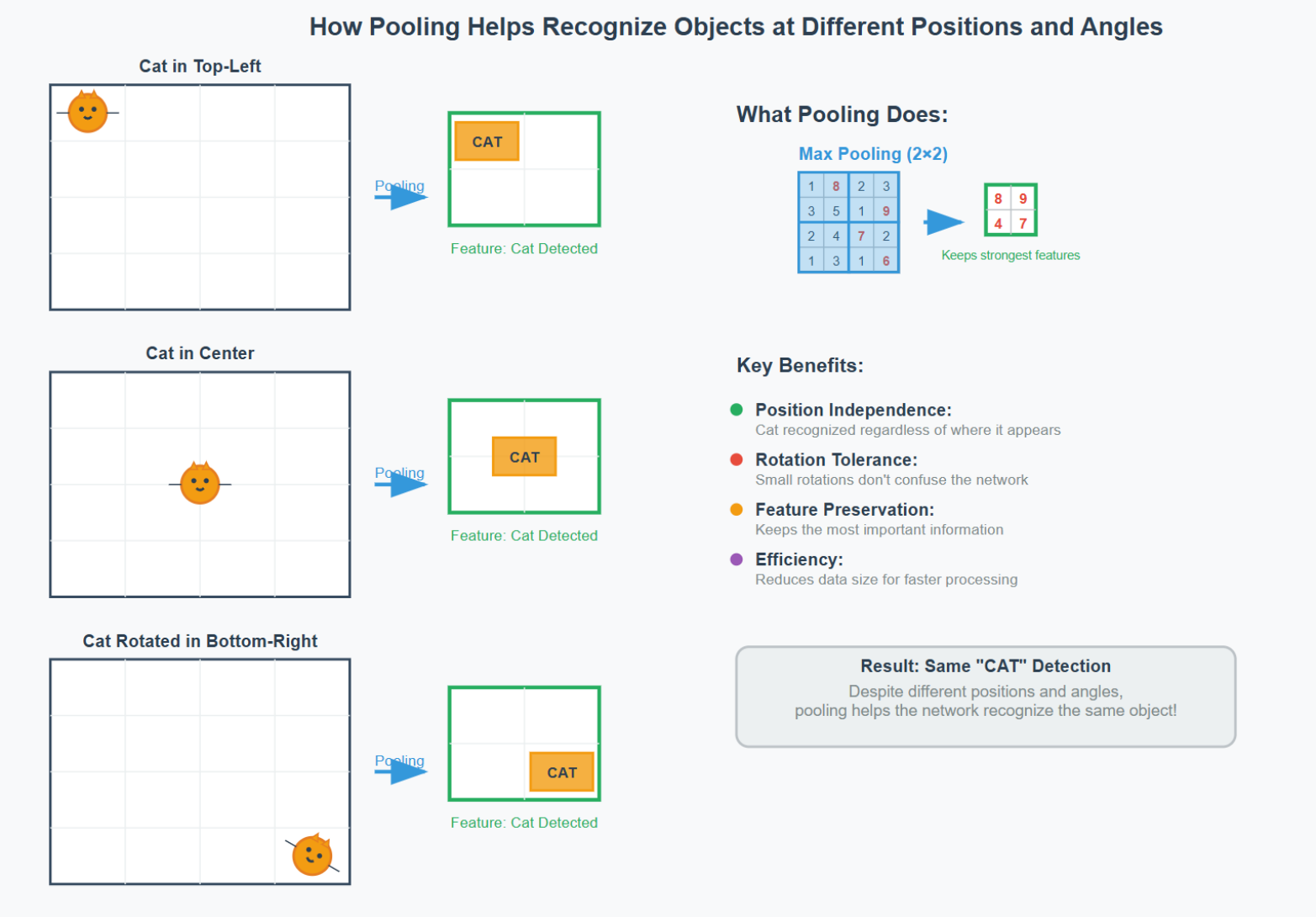
Image source: datahacker.rs

This image illustrates how Convolutional Neural Networks (CNNs) identify various types of edges in an image. The original photo of a street is passed through two different filters - one specialised for detecting **vertical edges** and the other for **horizontal edges**. The resulting outputs highlight the structural features in those directions, demonstrating how CNNs extract important patterns, such as building edges, road lines, and crosswalks. This is a fundamental step in feature extraction that allows CNNs to recognise shapes and objects in images.

Neural Networks – CNNs – Pooling

Pooling helps neural networks recognise objects even when they appear in different positions or angles in an image. Just like how you can recognize your friend's face whether they're standing on the left or right side of a photo, or whether their head is tilted slightly, pooling allows the computer to do the same thing. It works by examining small areas of the image and retaining only the most crucial information, while disregarding precise positioning details. This means the network can still identify a cat as a cat, whether it's sitting in the corner of the photo, lying in the centre, or even if the camera angle is slightly different. Without pooling, the computer might consider a cat in the top-left corner to be completely different from the same cat in the bottom-right corner.

Neural Networks – CNNs – Pooling

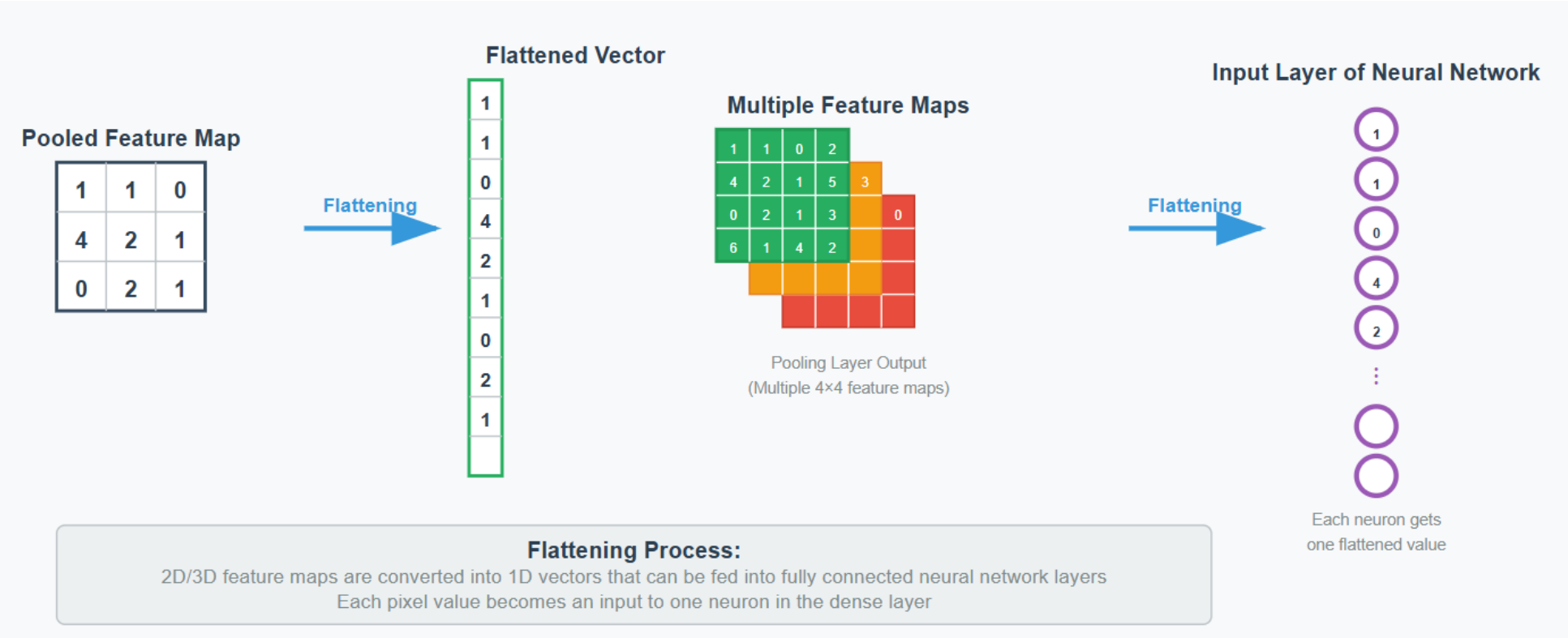


Spatial Pooling (subsampling or downsampling) reduces the dimensionality of each feature map but retains the most important information.

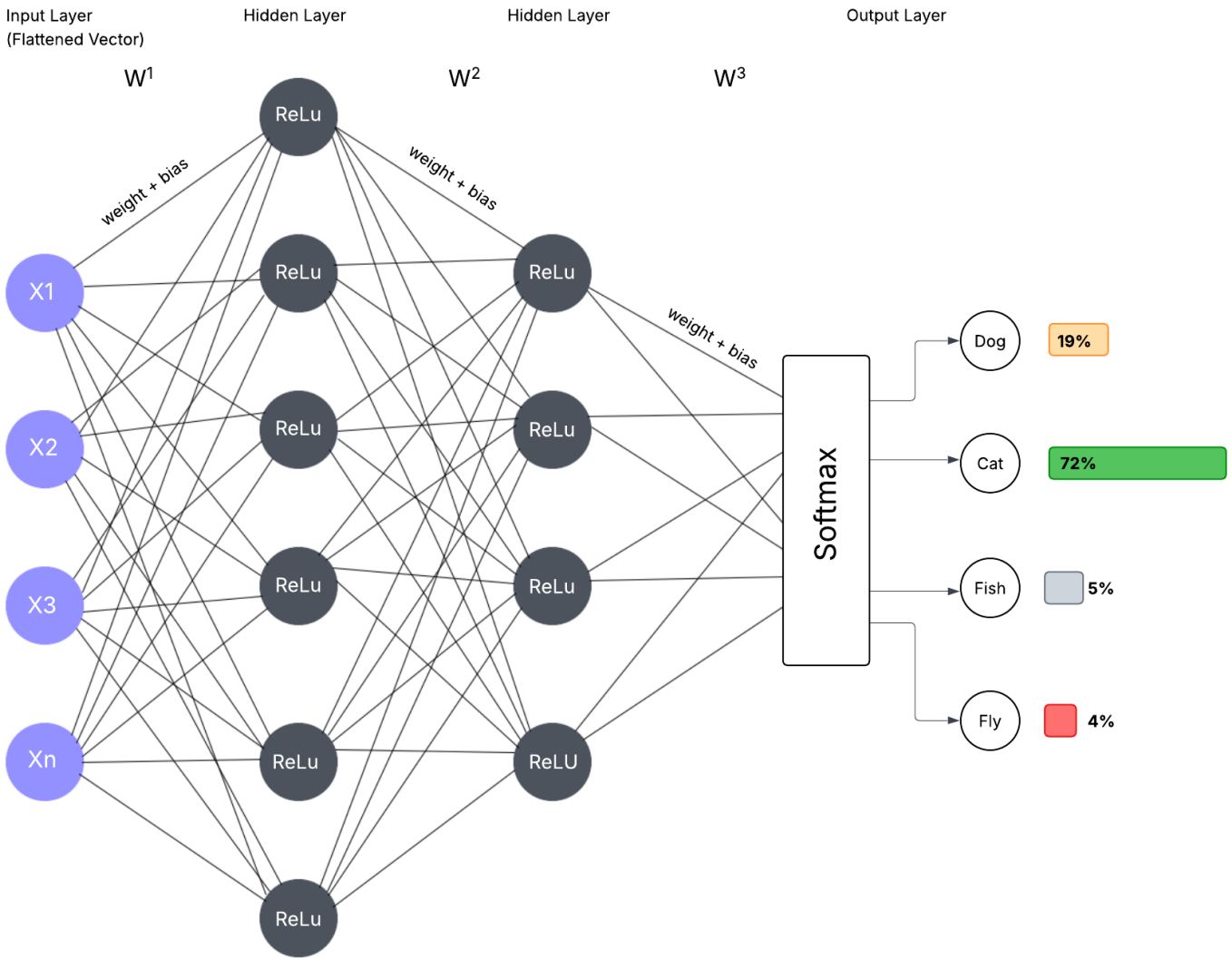
- Spatial Pooling can be of different types: Max, Average, Sum, etc.
- In the case of Max Pooling, we define a spatial neighbourhood (for example, a 2×2 window) and take the largest element from the rectified feature map within that window.
- In case of Average Pooling, instead of taking the largest element, we could also take the average of all elements in that window.
- In practice, Max Pooling has been shown to work better.

Neural Networks – CNNs – Flattening

After pooling, it's time to pass the extracted features into a traditional neural network. In this phase, the **pooled feature maps are flattened**, meaning they are reshaped into a **single one-dimensional vector** (a column of values). This flattened vector serves as the input to the fully connected layers of the neural network, enabling it to utilise the learned features for classification or decision-making.



Neural Networks – CNNs – Fully Connected Later (MLP)



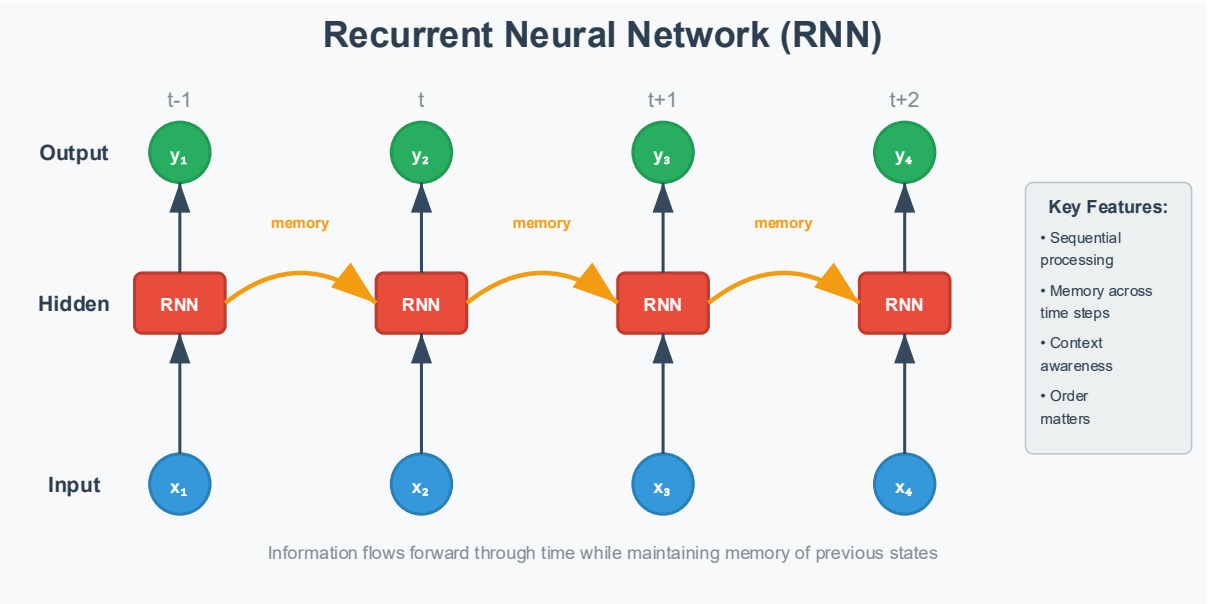
After flattening the pooled feature map into a one-dimensional vector, it is passed into a **fully connected neural network**. Each value in this vector becomes an **input neuron** that connects to all neurons in the next layer. Through layers of weighted connections and activation functions (like ReLU), the **network learns complex patterns**. Finally, the output layer uses a **softmax** function to assign probabilities to different classes, such as "cat", "dog", or "fish“, allowing the model to predict the most likely category based on the extracted image features.

DEMO APP

Neural Networks – Beyond CNNs - RNNs

RNN – Recurrent NN

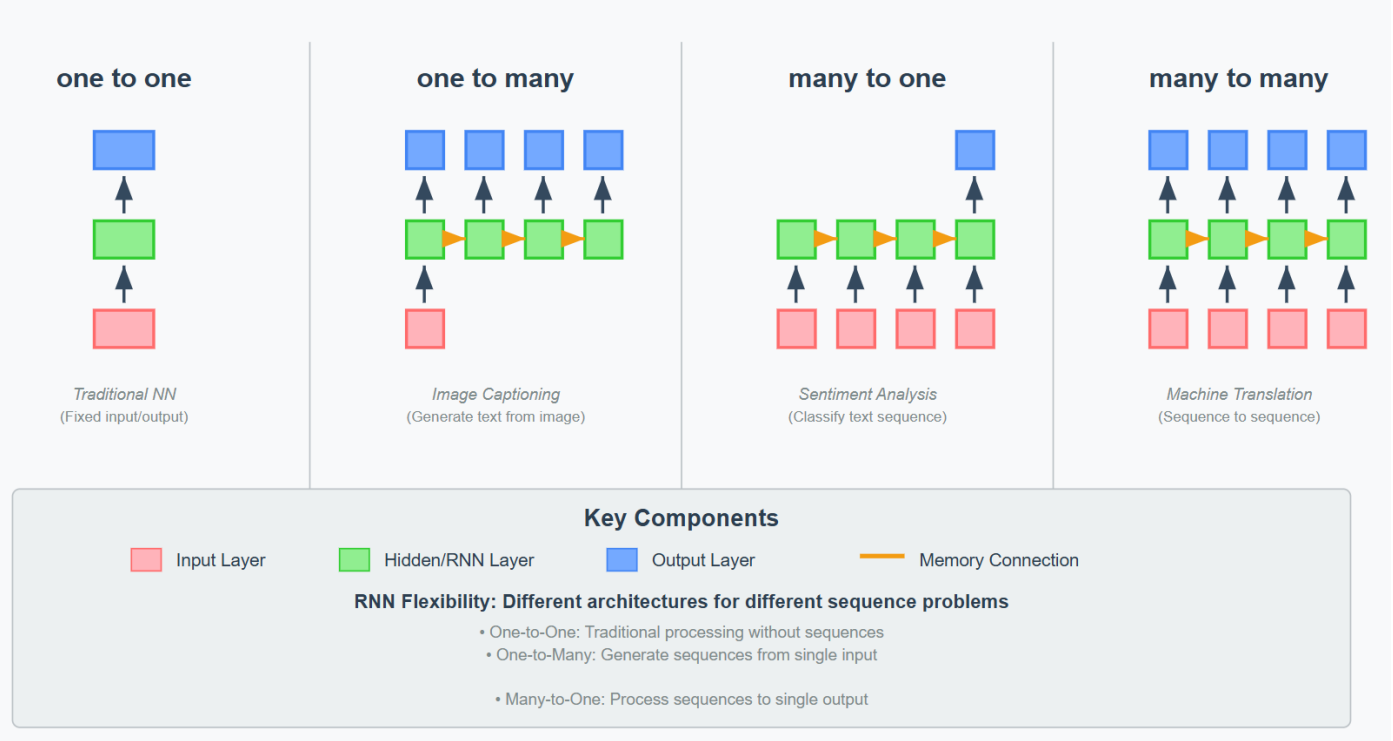
Traditional neural networks are robust, but they treat each input as if it exists in isolation, meaning they don't consider the *order* or *context* of data. This works fine for tasks like image classification or basic predictions, but completely falls apart when working with sequences, like sentences, speech, or time series, where each piece of information depends on what came before. For example, the meaning of a word in a sentence can change based on the words that came before it. Standard NNs can't remember previous inputs. That's why Recurrent Neural Networks (RNNs) were introduced – they have loops that allow information to be passed from one step to the next, giving the model a kind of "memory" to handle sequential data.



Sentence: "I went to the *bank* to deposit money."
The word *"bank"* depends on the earlier context ("deposit money") to mean *a financial institution, not a river bank*.
Now switch it:
Sentence: "I sat by the *bank* and watched the ducks."
Same word, completely different meaning because of the **sequence**.

Neural Networks – RNNs Architectures

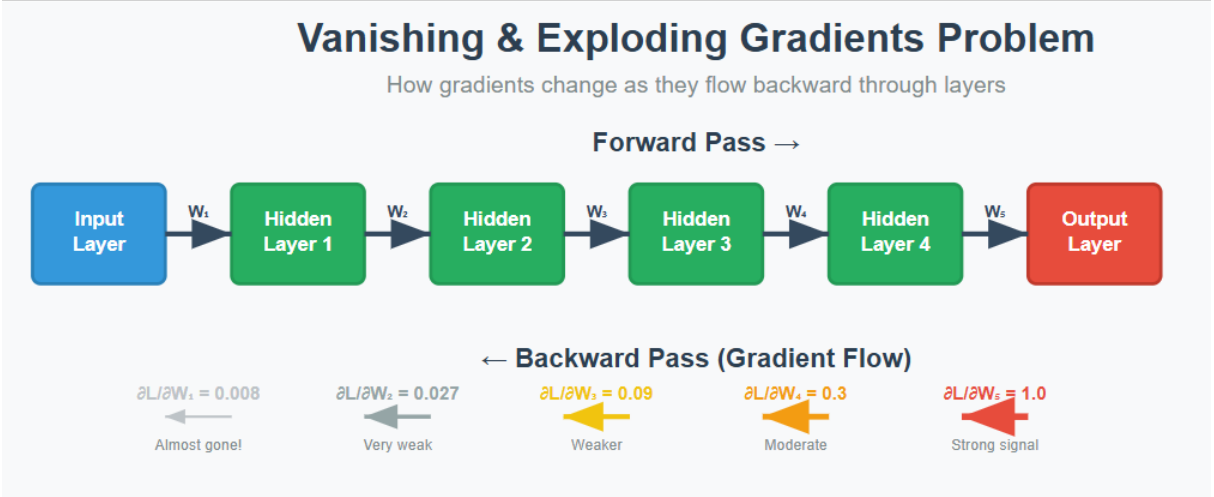
- **One to One** - Traditional neural network (not really RNN, shown for comparison)
- **One to Many** - Single input produces sequence output (e.g., image captioning)
- **Many-to-one** - Sequence input produces a single output (e.g., sentiment analysis)
- **Many-to-many** - Sequence input produces sequence output (e.g., machine translation, video classification)



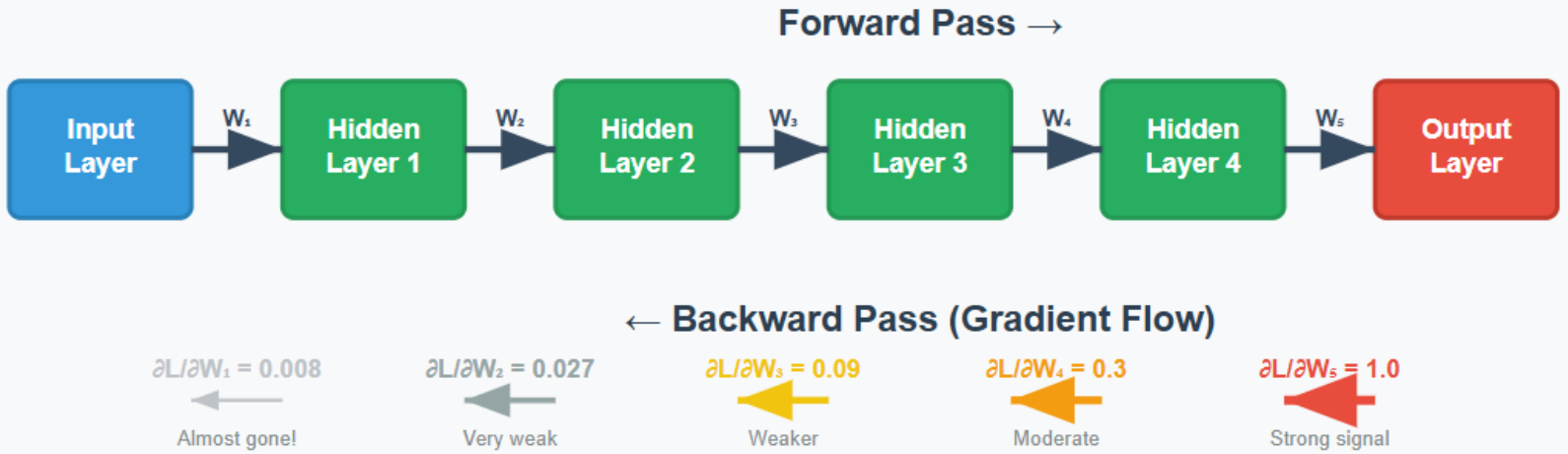
Neural Networks – RNN – Vanishing/Exploding Gradient

Vanishing & Exploding Gradients – What Went Wrong?

When the network adjusts its weights during training, it uses gradients, which are often small numbers. If you keep **multiplying small numbers together** (like 0.034 × 0.0234 × 0.01...), the result quickly becomes even **smaller**, eventually approaching zero. This is what causes **vanishing gradients** – the learning signal becomes so tiny that earlier layers in the network barely learn anything. On the other hand, if those values are large, repeated multiplication can cause the numbers to become huge, rendering the model unstable. This is especially problematic in **recurrent networks**, where the same weights are used repeatedly across multiple time steps, resulting in repeated multiplication. That's why managing gradients is so important – and why we needed smarter architectures like **LSTM** to keep learning stable.



Neural Networks – RNN – Vanishing/Exploding Gradient



Mathematical Explanation

Chain Rule in Backpropagation:

$\partial L/\partial W_1 = \partial L/\partial W_5 \times \partial W_5/\partial W_4 \times \partial W_4/\partial W_3 \times \partial W_3/\partial W_2 \times \partial W_2/\partial W_1$

Example Calculation (assuming each derivative ≈ 0.3):

$\partial L/\partial W_1 = 1.0 \times 0.3 \times 0.3 \times 0.3 \times 0.3 = 1.0 \times 0.3^4 = 0.0081$

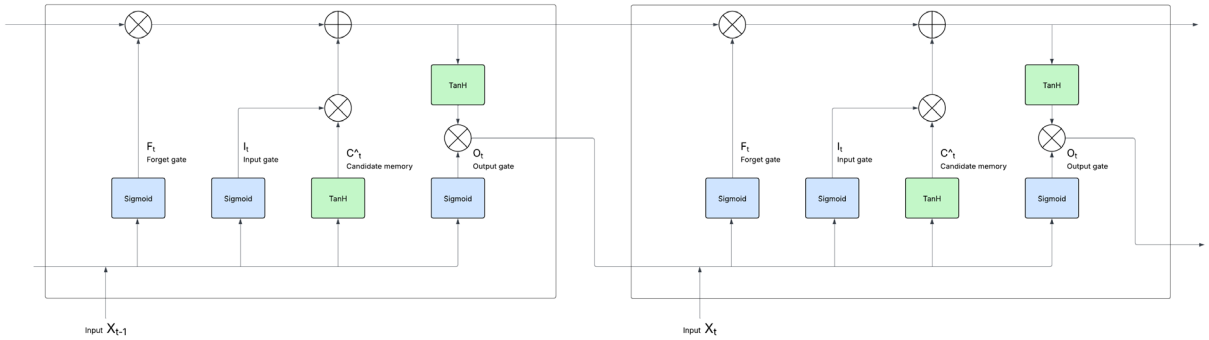
The Problem:

- When derivatives < 1: Gradients shrink exponentially → Vanishing Gradients
- When derivatives > 1: Gradients grow exponentially → Exploding Gradients

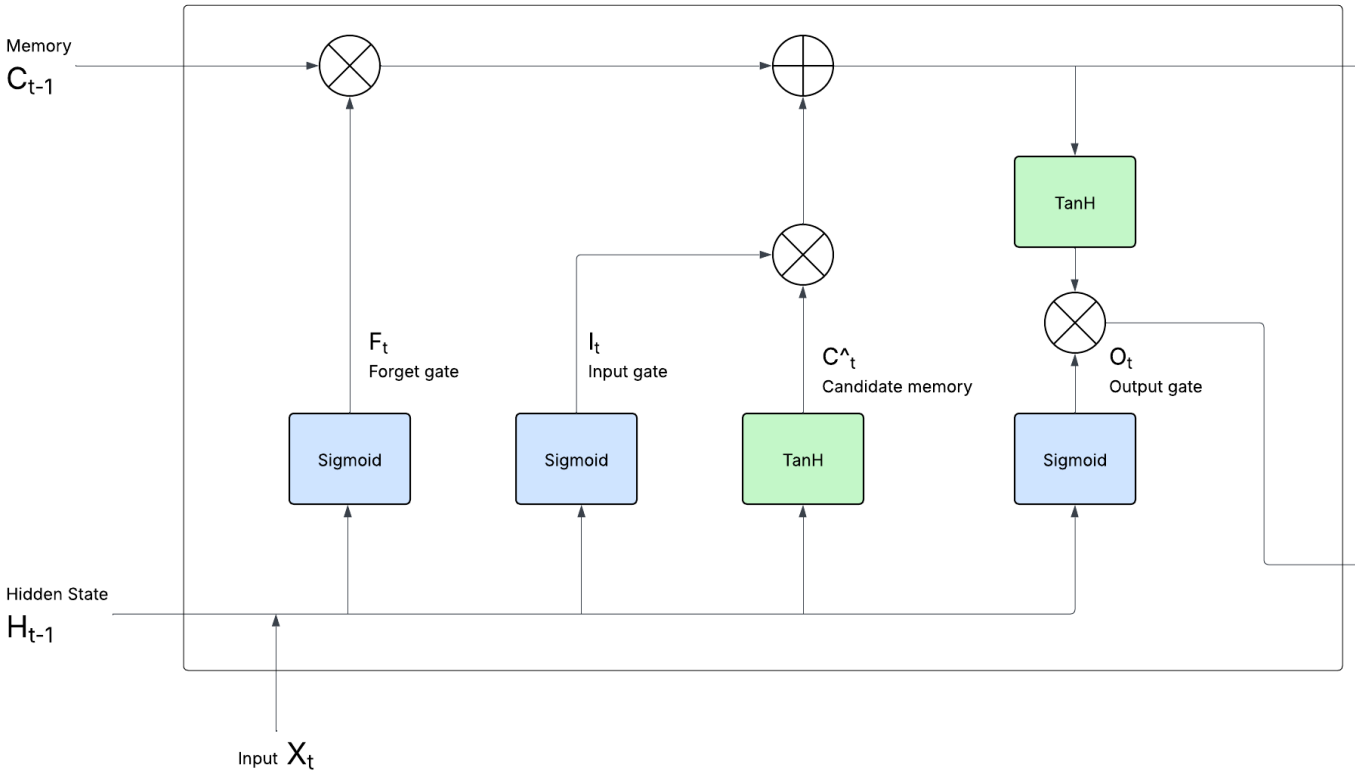
Neural Networks – Beyond CNNs - LTSMs

LSTM - Long Short-Term Memory

RNNs introduced the ability to remember previous steps in a sequence. Still, they struggle when that memory needs to span longer distances, such as recalling the beginning of a sentence by the time you reach the end. This is known as the long-term dependency problem, and it's where RNNs often fail due to issues such as the vanishing gradient. To fix this, researchers developed **Long Short-Term Memory (LSTM)** networks. LSTMs employ a sophisticated structure with gates – forget, input, and output – to control what information is retained, updated, or discarded at each step. This allows them to retain important details for much longer. They've been a game changer in tasks like translation, chatbots, and music generation. However, while they're powerful, LSTMs are still complex and slow to train, which eventually led to the rise of even more efficient models like Transformers.



Neural Networks – LTSM Block



$C_t = F_t * C_{t-1} + I_t * C^_t$

σ (Sigmoid) blocks — These are **gates**, and each is small neural network layer with Sigmoid activation (0 to 1)
gate output=$\sigma(W \cdot [H_{t-1}, X_t] + b)$
Purpose: Controls how much information passes through

Candidate memory: proposed update to the cell's long-term memory
Why candidate ? Because it is not guaranteed to be added - must be filtered by the input gates before adding to memory.

Neural Networks – LTSM Breakdown with example sentence

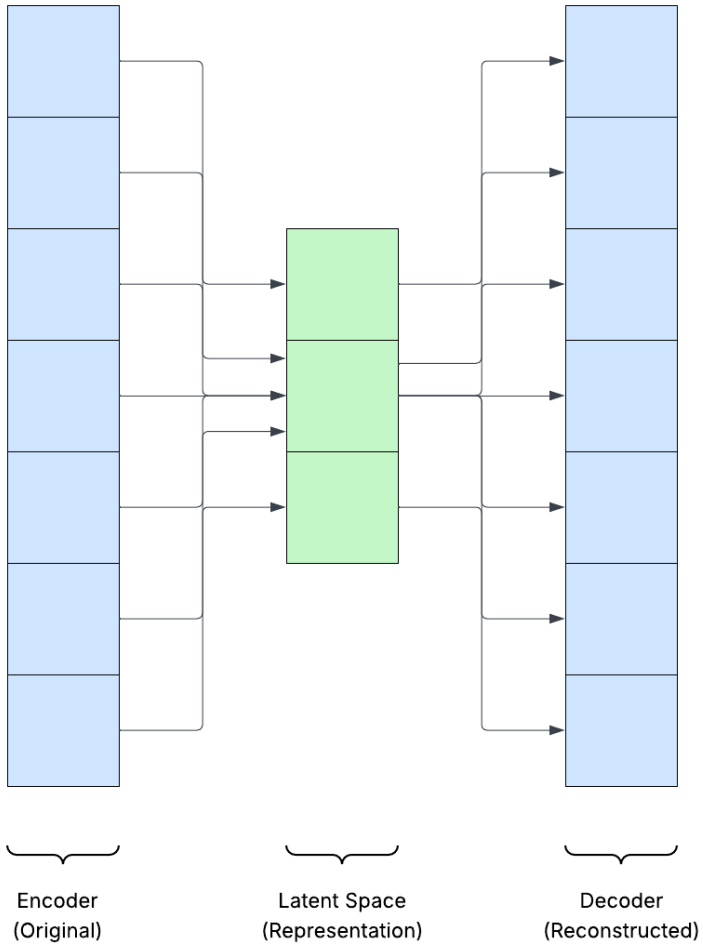
Example sentence to see how an LSTM might **remember**, **forget**, and **update** its memory:
"**The doctor** who examined the patients for hours **was exhausted.**“

1. **“The doctor”** – subject of the sentence
   - Remember: The identity of the subject is important for interpreting the verb “was”
   - ✅ LSTM keeps this in memory via the cell state
2. **“who examined the patients for hours”**
   - This is a relative clause (extra info)
   - **Candidate memory:** This is detailed and might be worth remembering temporarily, but not long-term.
   - ❌ **Forget:** Once the clause ends, LSTM can reduce the importance of this detail (via **forget gate**) to focus on the main sentence again.
3. **“was exhausted”**
   - This is the **main verb and predicate**.
   - ✅ **Use memory:** LSTM needs to recall **"The doctor"** as the subject, so it can correctly connect **"was exhausted.“**
   - ✅ **Output gate:** Helps send out the right hidden state for classification, translation, etc.

Neural Networks – Encoders and Decoders

Encoders & Decoders – Input to Output in Sequences

While models like LSTMs can handle sequences, they struggle with more complex tasks, such as translating long sentences or generating full responses, where it is necessary to process the entire input before generating output. That's where the **encoder-decoder architecture** comes in. Instead of handling inputs and outputs one step at a time, this design **separates the process into two parts**: the **encoder** reads and compresses the entire input sequence into a compact representation, and the **decoder** takes that summary and generates the output sequence. This approach improved tasks like machine translation, summarisation, and question answering. It was originally used with RNNs and LSTMs, but later became the foundation of the **Transformer architecture**, which took this concept to a whole new level with attention mechanisms.



Neural Networks – Encoders and Decoders – Use cases

Machine Translation

- **Use:** Convert a full sentence in one language into a compressed representation that captures meaning and context.
- **Example:** English -> French translation (Encoder processes English sentence)

Sentence Embeddings / Semantic Search

- **Use:** Encode sentences or documents into fixed-length vectors that capture meaning.
- **Example:** Searching a database of FAQs by converting both queries and answers into vectors and finding the most similar ones.

Text Classification

- **Use:** Encode the input text (e.g. reviews, tweets, emails) to a vector for classification.
- **Example:** Spam detection, sentiment analysis, topic classification.

Image Feature Extraction

- **Use:** CNN-based encoders are used to process images and extract key features.
- **Example:** Object detection, image classification, face recognition.

Audio and Speech Recognition

- **Use:** Encode raw audio or spectrograms into meaningful features before passing to a decoder or classifier
- **Example:** Voice-to-text systems (e.g. Siri, Google Assistant).Sentence Embeddings / Semantic Search
- **Use:** Encode sentences or documents into fixed-length vectors that capture meaning.
- **Example:** Searching a database of FAQs by converting both queries and answers into vectors and finding the most similar ones.

Anomaly Detection / Autoencoders

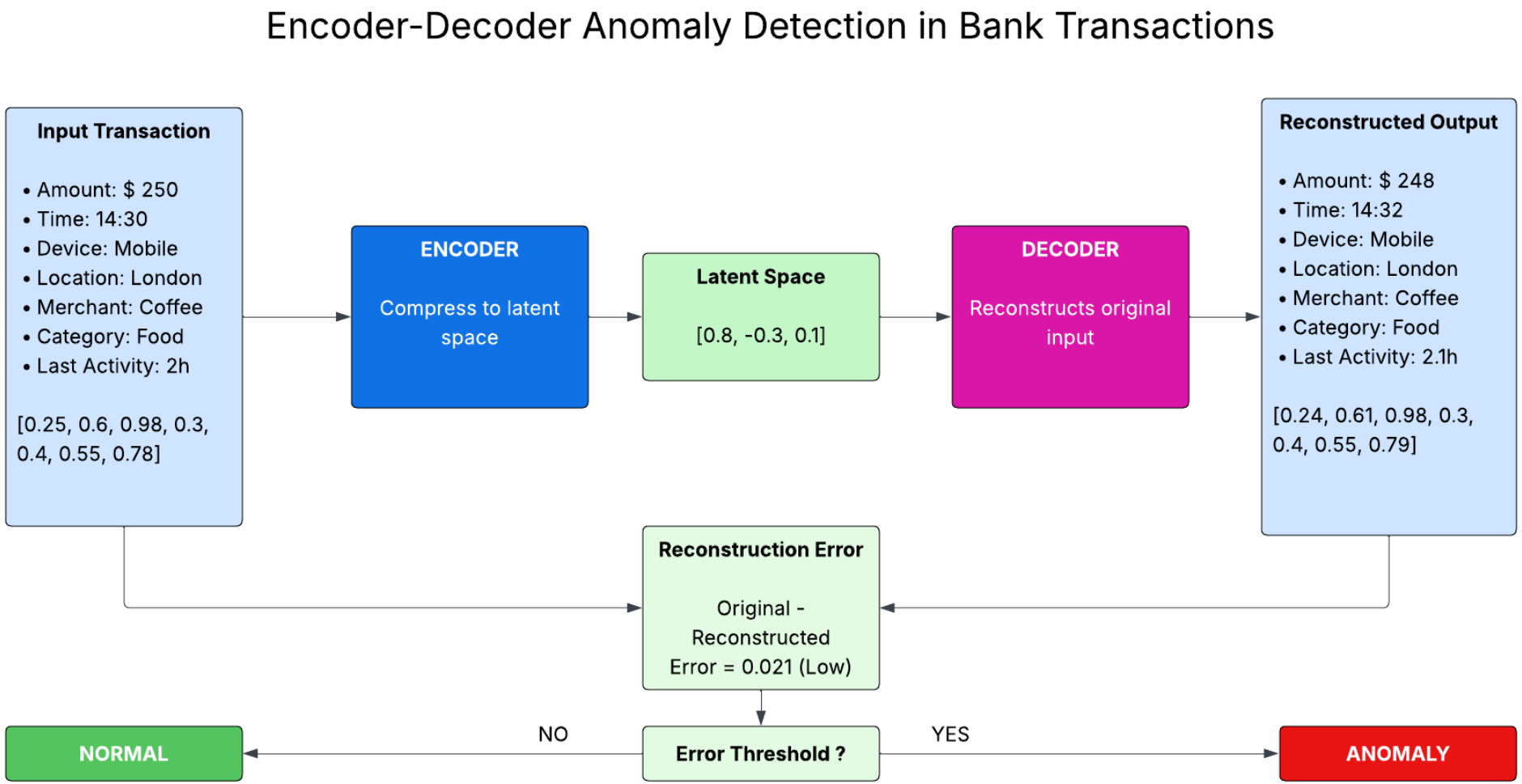
- **Use:** An encoder compresses input data (e.g. network traffic, sensor readings), and a decoder tries to reconstruct it. Large reconstruction errors suggest anomalies.
- **Example:** Fraud detection, industrial fault detection.

Neural Networks – Encoders and Decoders – Anomaly Detection Demo

Encoders – Anomaly Detection in Bank Transactions

Encoders can be incredibly useful in detecting unusual or fraudulent activity in banking. One common approach is to use an **autoencoder**, which learns to compress and then reconstruct normal transaction data. Each transaction, including details like amount, time since last activity, device type, location, and merchant category, is first transformed into a structured vector. The **encoder** takes this input and compresses it into a smaller, abstract representation that captures the key patterns of what a "normal" transaction looks like. Then, the **decoder** tries to reconstruct the original transaction from this compressed version. If the transaction is typical, the model reconstructs it accurately with only a small error. However, if the transaction is unusual or suspicious, for example, a large transfer from an unknown device or a sudden change in country, the reconstruction error will be significantly higher, as the model has never encountered anything like it during training. By setting a threshold on this error, we can flag transactions that don't fit the learned pattern, making encoders a powerful unsupervised tool for real-time fraud detection.

Neural Networks – Encoders and Decoders – Anomaly Detection Demo



DEMO APP

Neural Networks - Transformers

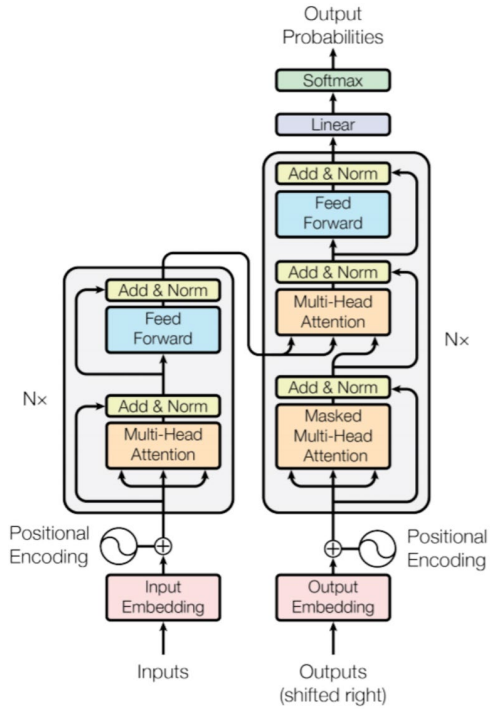
“Attention Is All You Need”

2017, Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin.

Transformers are a powerful type of neural network architecture introduced in 2017 with the idea that **“attention is all you need.”** Unlike older models like RNNs, Transformers don’t process data step-by-step - they use **self-attention** to understand the entire input at once, making them faster and more accurate for language tasks.

Transformers are the foundation behind many modern AI models, including **GPT (Generative Pre-trained Transformer)**, which uses this architecture to generate human-like text

… but that is material for a separate presentation



Neural Networks - End

Thank You!

Any Questions?