BEYOND SURFACE:

From One Neuron To CNNs & Transformers

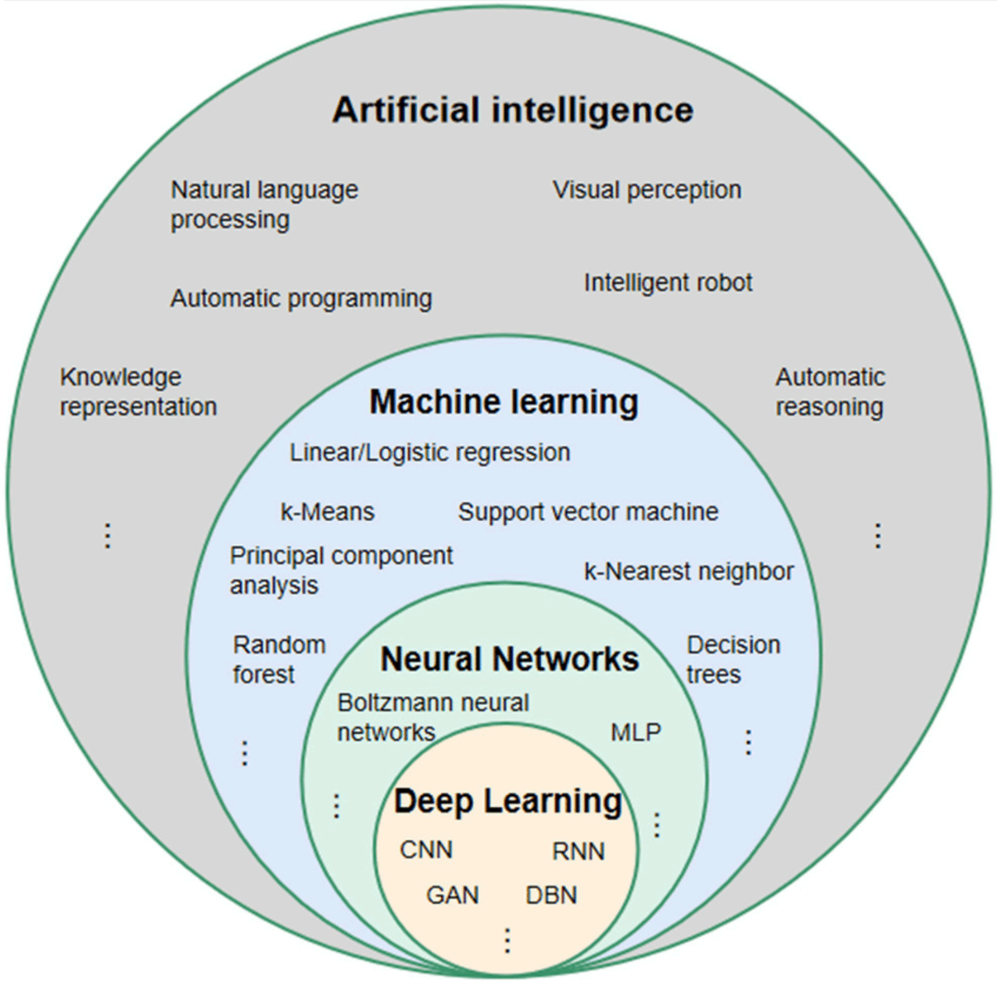
TEAM MEMBERS

Yan Zhe
Y2 DSE

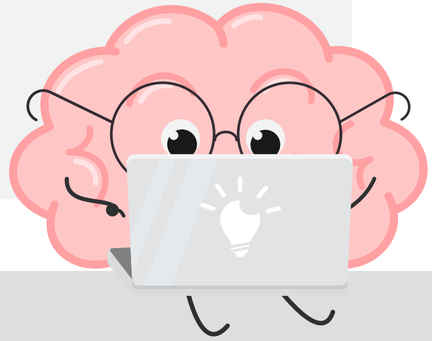
Jason
Y3 DSA

Harish
Y3 DSA

DEEP LEARNING



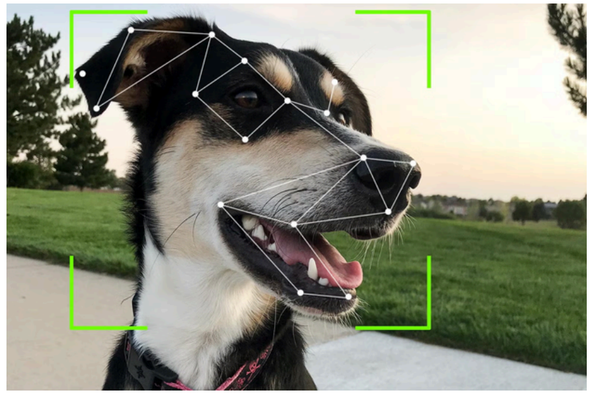
Deep learning is a subset of machine learning driven by multilayered neural networks whose design is inspired by the structure of the human brain.



APPLICATIONS OF DEEP LEARNING

Deep learning architectures we are exploring today...

CNNs drive vision-based AI like facial recognition and self-driving cars



Transformers power generative AI tools like ChatGPT and Gemini.

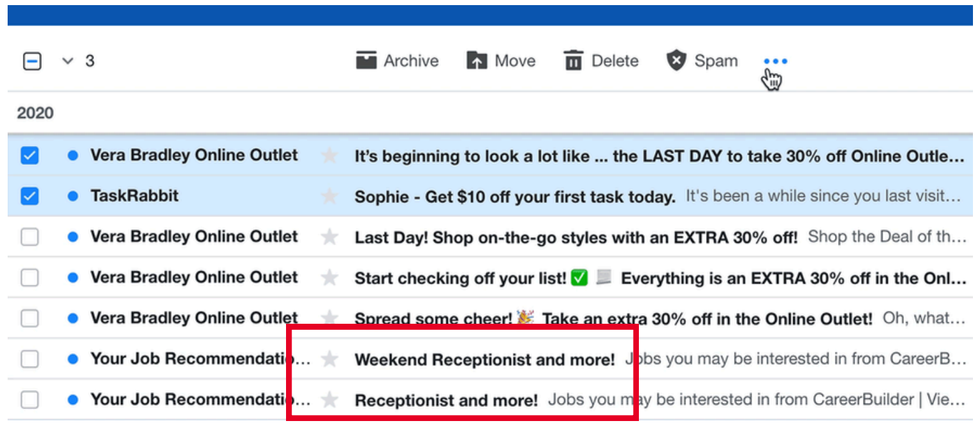
ChatGPT

Gemini

COMPARING CLASSICAL MACHINE LEARNING & DEEP LEARNING

Classical Machine Learning

- Different methods, like linear regression and decision trees
- Works well with structured, well-labelled data
- Requires extraction of features from raw data (feature engineering)

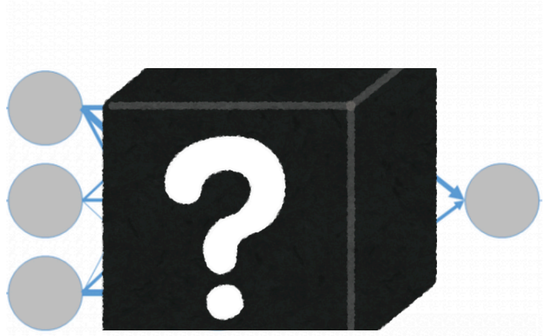


Problem: Predicting Spam Email
Raw Data: Text
Feature Engineering: Whether the mail subject contains !

Creds: https://aws.amazon.com/compare/the-difference-between-machine-learning-and-deep-learning/

CHALLENGES OF DEEP LEARNING

Lack of intepretability



Input → Black Box → Output

- Each neuron captures abstract patterns that are not human-readable
- Referred to as “black boxes” due to difficulty in understand how it produces output

Requires large amount of data and computational resources

Oracle, OpenAI Sign $300 Billion Cloud Deal

The majority of new revenue revealed by Oracle will come from OpenAI deal, sources say

By Berber Jin Follow

Updated Sept. 10, 2025 4:08 pm ET

Share Resize 109 Listen (2 min)

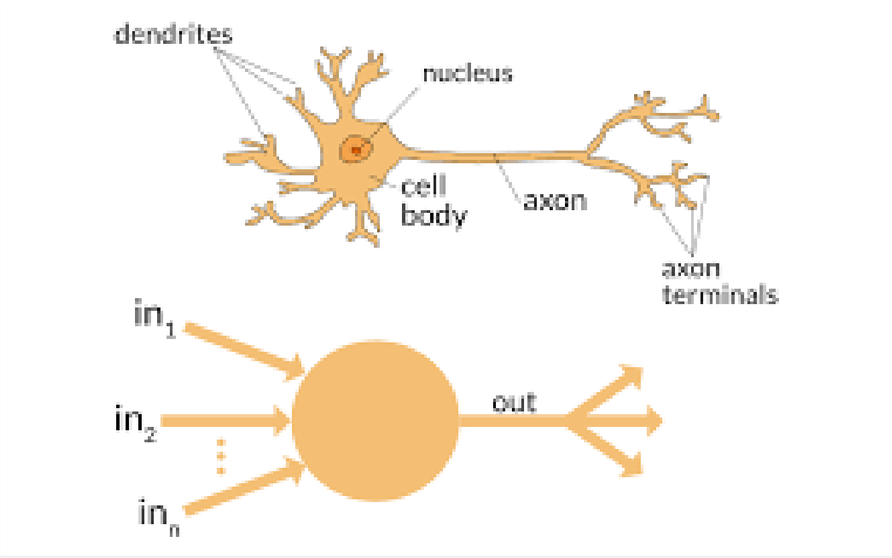
Surging share prices increased Oracle Chairman Larry Ellison's wealth by around $100 billion

- Training large models requires massive datasets that need to be stored and managed
- Demands many high-performance GPUs driving up expenses

Creds: https://aws.amazon.com/compare/the-difference-between-machine-learning-and-deep-learning/

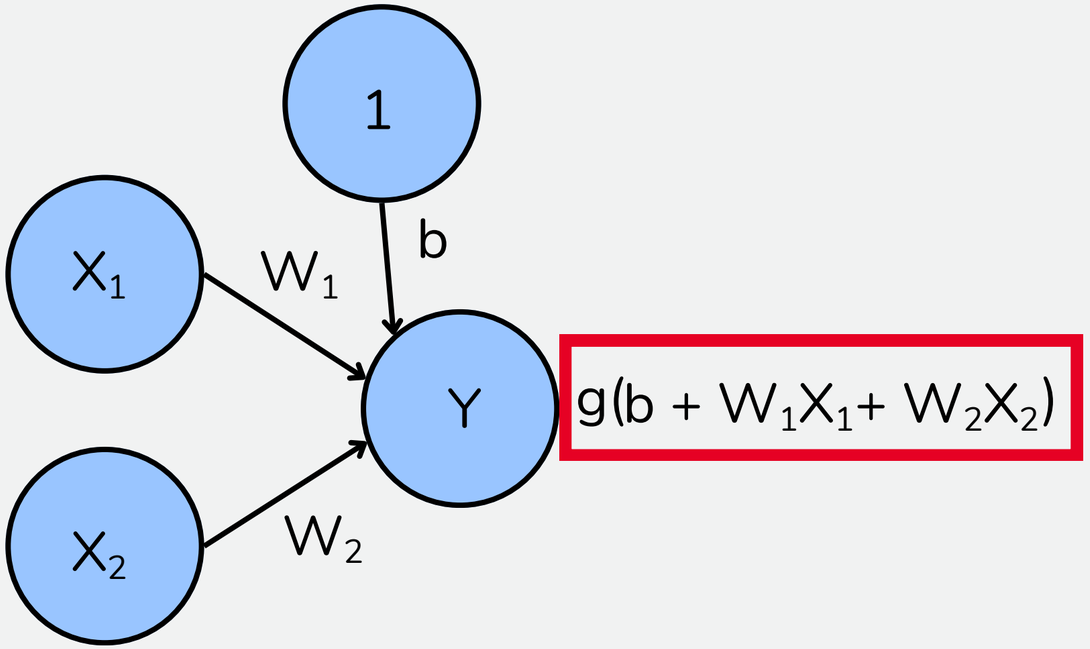
NEURAL NETWORKS

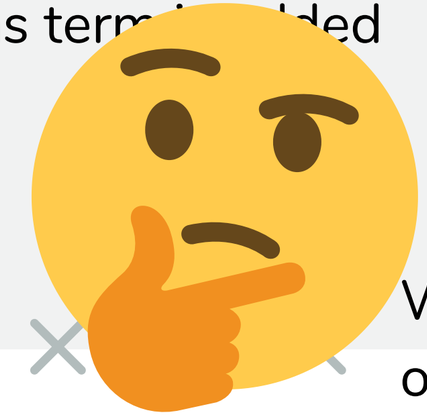
- To study how humans learn, scientists looked into the human brain
- A network of interconnecting units called neurons that takes in electrical signals and outputs electrical signals
- Artificial neuron models this input to output.
- Instead of “electrical signals”, the inputs and outputs are numbers



PERCEPTRON

- A simple artificial neural network that also models f(X_1, X_2)→ Output (Y)
- Has two inputs X_1 and X_2
- Contribution of each input adjusted by its weight W_1 and W_2
- Bias term included





What is the difference between the function of a perceptron and linear regression?

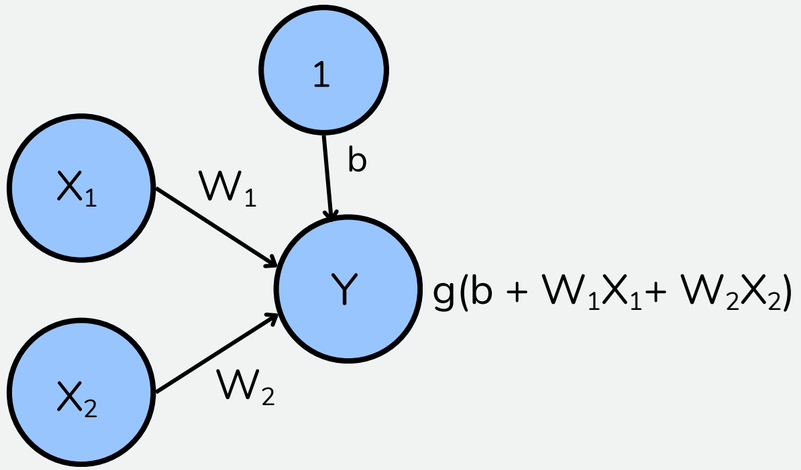
ACTIVATION FUNCTIONS

Behaviour of biological neuron:

- A biological neuron decides whether to fire and send an electrical signal.
- If it fires, the signal intensity varies based on the input.

How activation functions model this behaviour:

- Transforms the linear combination of inputs into output (Y) that is generally not linear in X_1, X_2 , ...X_n
- Affects final output of perceptron



ACTIVATION FUNCTIONS

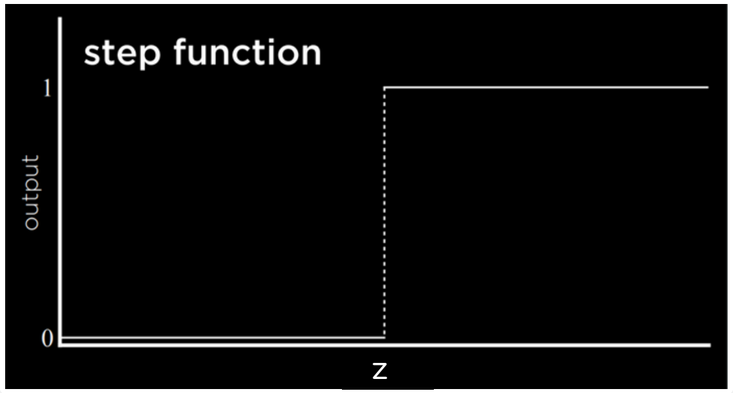
Step function

Behaviour of AND logic operator:

X_1	X_2	Y
True	True	True
True	False	False
False	True	False
False	False	False

Activation Function:

g(Z) = 1 if Z ≥ 0, else 0



Classification:

Let W_1 , W_2 be 1 and let b = -2

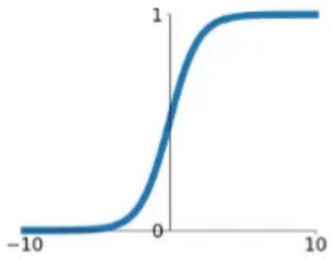
X_1	X_2	$b + W_1X_1 + W_2X_2$	$g(b + W_1X_1 + W_2X_2)$	
1	1	0	1	TRUE
1	0	-1	0	FALSE
0	1	-1	0	FALSE
0	0	-2	0	FALSE

ACTIVATION FUNCTIONS

Sigmoid

Applications: Desired output is between 0 and 1

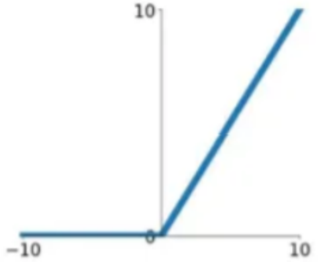
$$g(x) = \frac{1}{1+e^{-x}}$$



Rectified Linear Unit (ReLU)

Application: Default activation function in many neural networks

g(X) = max(0,X)



ACTIVATION FUNCTIONS

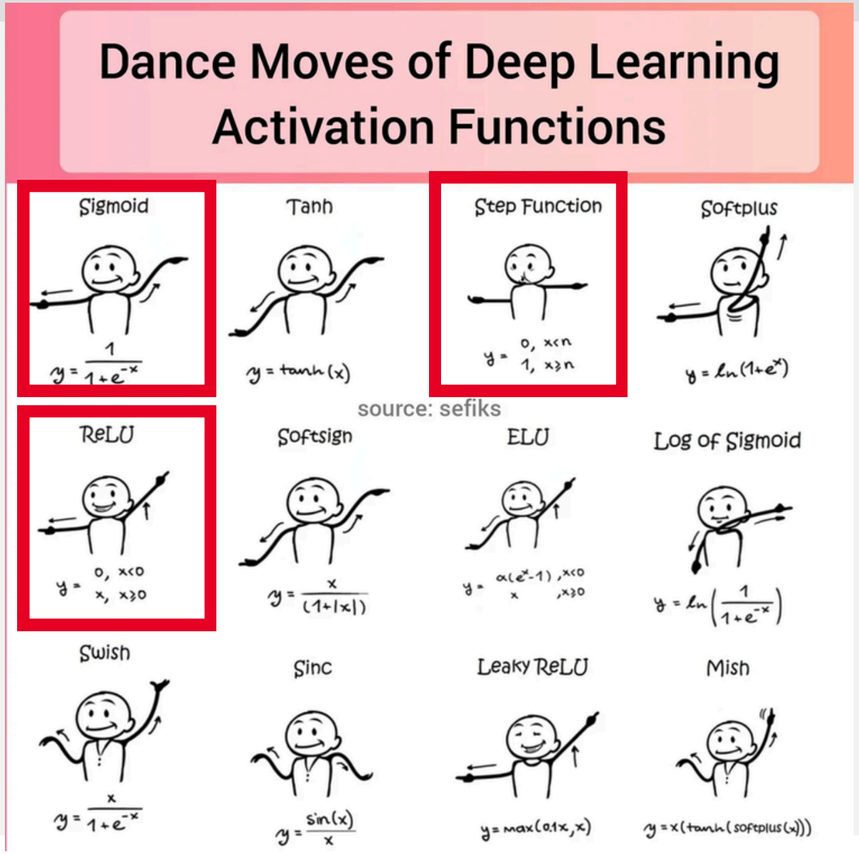


Image creds: https://www.reddit.com/r/learnmachinelearning/comments/lvehmi/deep_learning_activation_functions_using_dance/

WEIGHTS & BIAS

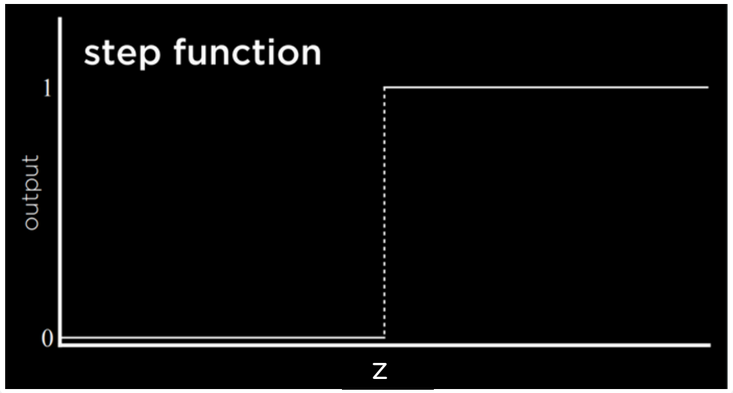
Step function

Behaviour of OR logic operator:

X_1	X_2	Y
True	True	True
True	False	True
False	True	True
False	False	False

Activation Function:

g(Z) = 1 if Z ≥ 0, else 0



Classification:

Let W_1 , W_2 be 2 and let b = -2

X_1	X_2	$b + W_1X_1 + W_2X_2$		$g(b + W_1X_1 + W_2X_2)$	
1	1		2	1	TRUE
1	0		0	1	TRUE
0	1		0	1	TRUE
0	0		-2	0	FALSE

WEIGHTS & BIAS

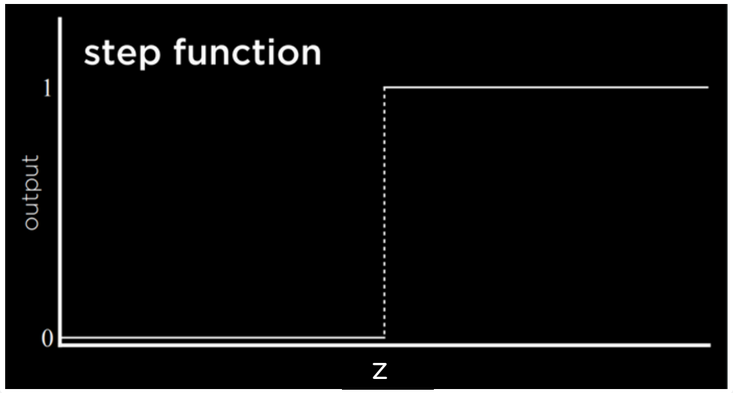
Step function

Behaviour of OR logic operator:

X_1	X_2	Y
True	True	True
True	False	True
False	True	True
False	False	False

Activation Function:

g(Z) = 1 if Z ≥ 0, else 0



Classification:

Let W_1 , W_2 be 1 and let b = ?

X_1	X_2	$b + W_1X_1 + W_2X_2$	$g(b + W_1X_1 + W_2X_2)$	
1	1	?	1	TRUE
1	0		1	TRUE
0	1		1	TRUE
0	0		0	FALSE

WEIGHTS & BIAS

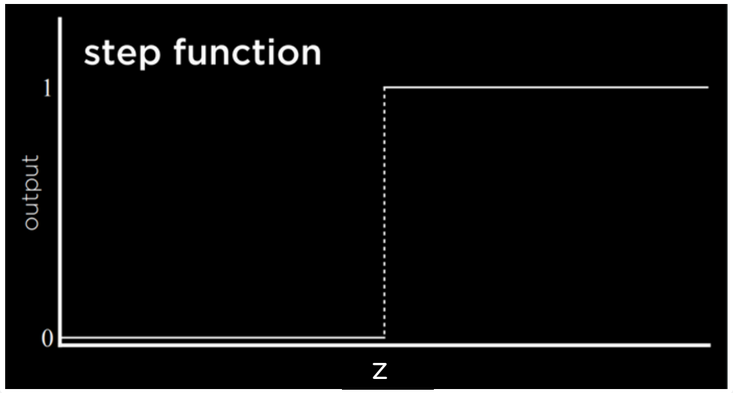
Step function

Behaviour of OR logic operator:

X_1	X_2	Y
True	True	True
True	False	True
False	True	True
False	False	False

Activation Function:

g(Z) = 1 if Z ≥ 0, else 0



Classification:

Let W_1 , W_2 be 1 and **let b = -1**

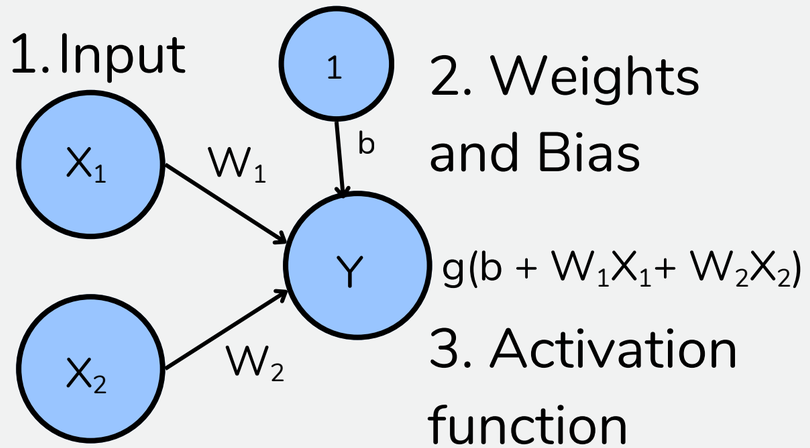
X_1	X_2	$b + W_1X_1 + W_2X_2$		$g(b + W_1X_1 + W_2X_2)$	
1	1		1	1	TRUE
1	0		0	1	TRUE
0	1		0	1	TRUE
0	0		-1	0	FALSE

Changing the weights and bias of neural networks can change their output and behaviour

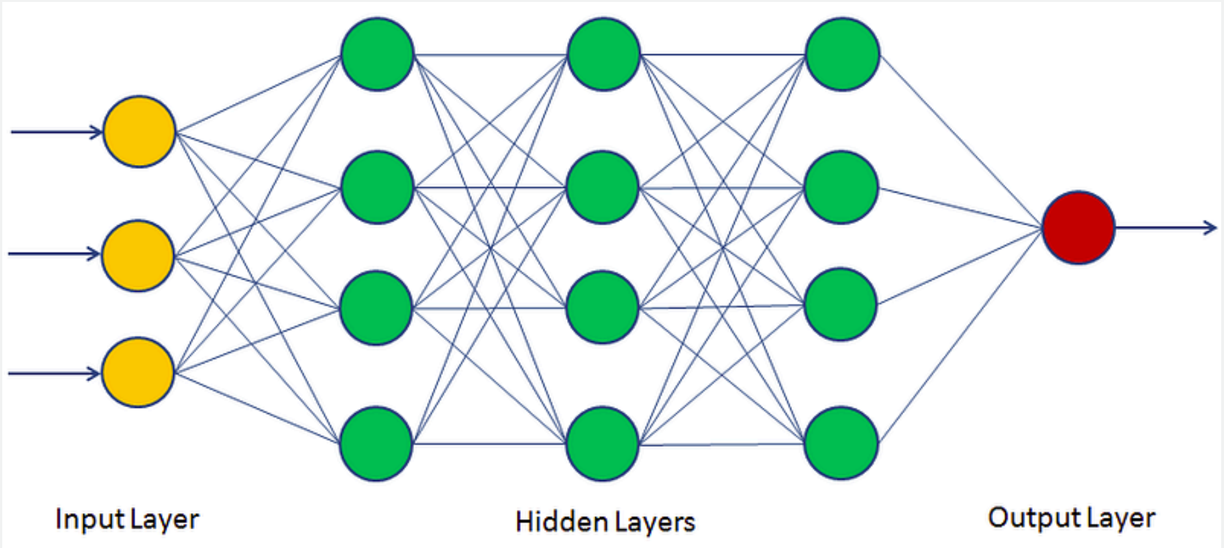
ARTIFICIAL NEURAL NETWORKS

So far:

We looked at the characteristics of a perceptron which forms the simplest artificial neural network



How ANN looks like:



DECISION BOUNDARIES

What a perceptron is good at:

Identify linear decision boundaries in input space that lead to different outcomes

- Certain combinations of X_1 and X_2 lead to red outcomes.
- Straight line separates red and blue outcome
- Weights adjust slope of line and bias adjusts intercept of line

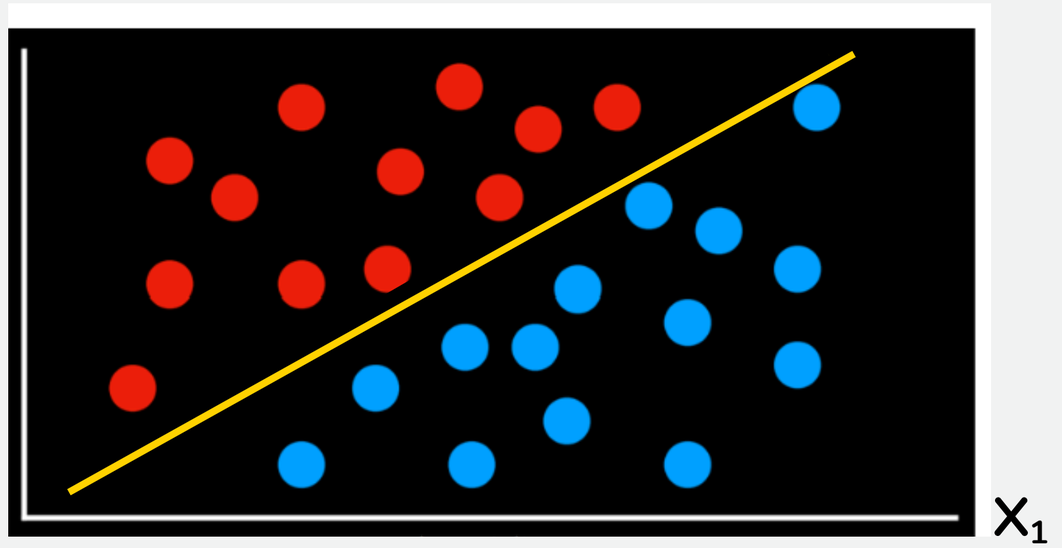


Image creds: https://cs50.harvard.edu/extension/ai/2020/fall/notes/5/

DECISION BOUNDARIES

What if the input space appears like this?
A single perceptron cannot establish a non-linear decision boundary like this

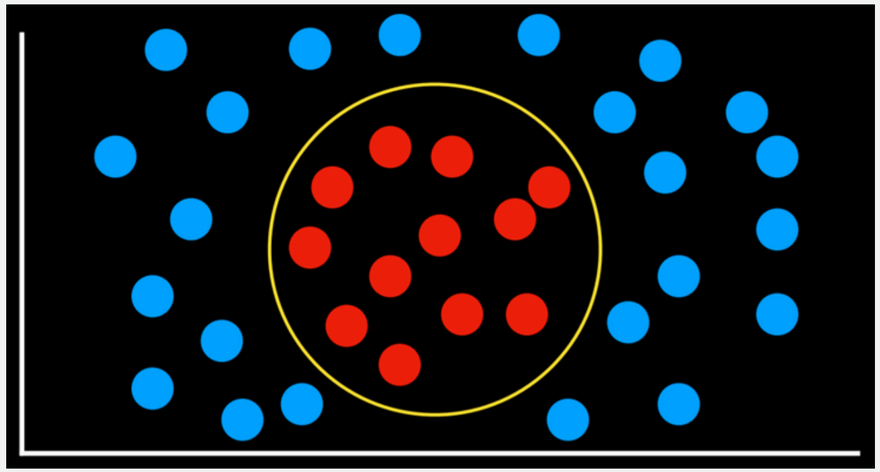


Image creds: https://cs50.harvard.edu/extension/ai/2020/fall/notes/5/

MULTI-LAYER PERCEPTRON(MLP)

- Inputs: X_1 (horizontal axis), X_2 (vertical axis)
- Hidden layer neurons: V_1 - V_4 each detect linear boundary
- **Activation function of V_n: :** Output 1 if the point lies on the red side of boundary, else 0
- **Activation function for Y :** Output 1 (classify red) if $\sum V_n = 4$, else Y = 0 (classify blue)

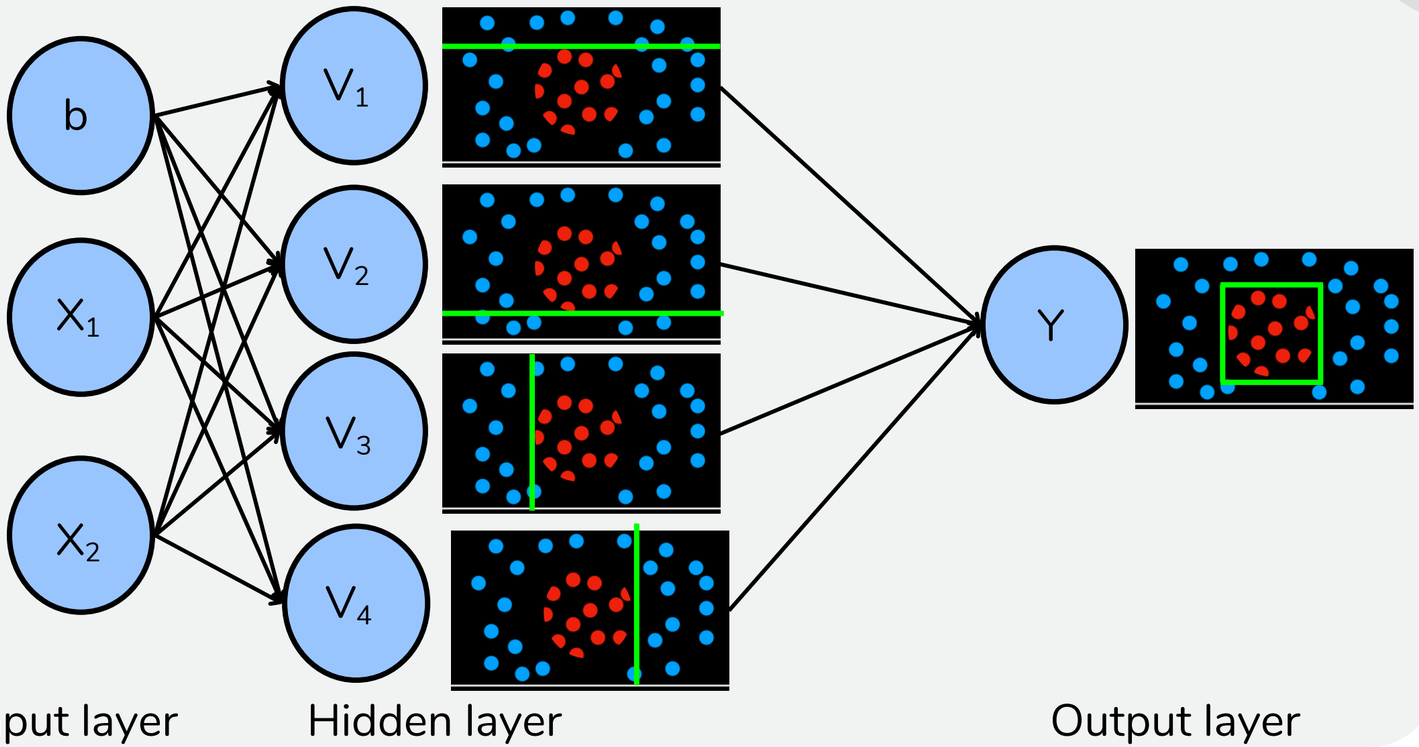


Image creds: https://cs50.harvard.edu/extension/ai/2020/fall/notes/5/

MULTI-LAYER PERCEPTRON(MLP)

Try it yourself:

playground.tensorflow.org

MULTI-LAYER PERCEPTRON(MLP)

More often than not, input space are not linearly separable

To capture non-linearities of input space:

1. Use of hidden layers
2. Use of activation functions

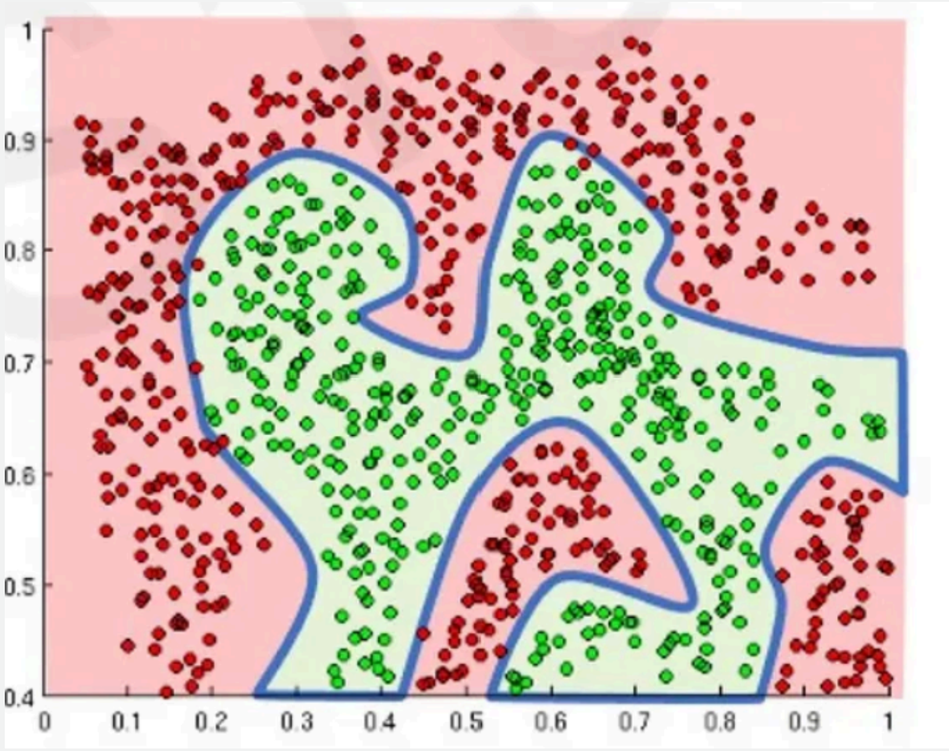


Image creds: https://medium.com/@jiaop24/why-activation-function-is-necessary-in-neural-networks-5058ac91aa71

Training ANN

MEASURING LOSS:
LOSS FUNCTIONS

Continuous case

$$L = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

- Mean Squared Error (MSE)
- This is a common loss function for continuous outputs
- Average of the sums of errors squared
- Large errors are punished more heavily because they are squared

Classification case

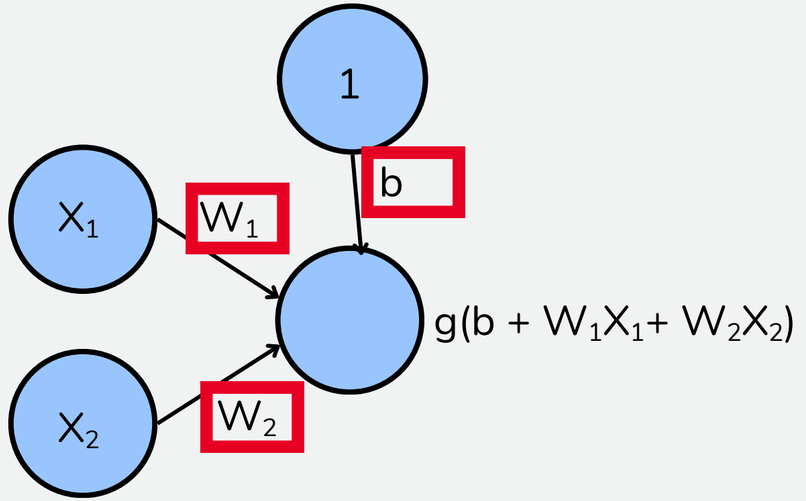
$$\text{Log Loss} = -\frac{1}{N}\sum_{i=1}^{N}\left[y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)\right]$$

- Known as Binary Cross-Entropy loss
- This is a common loss function for binary problems where true Y is either 0 or 1
- The prediction(ŷ) will be the probability of class 1
- Punishes confident wrong predictions heavily

INITIALISING LEARNING PROCESS

Learning in neural networks happens by adjusting the weights and bias to minimise error

1. Weights are assigned random values initially
2. Forward pass occurs, inputs flow through neural network producing prediction using current weights
3. Prediction and actual value of training data is compared and loss is measured



BACKPROPAGATION: RATATOUILLE VERSION

What if...
The food served tasted bad

- Sauce was too salty
- Chef tried compensating for thicker than usual vegetables

Portion of blame:

- Chef admits vegetables were cut too thick
- Claims vegetables were not fresh

Portion of blame:

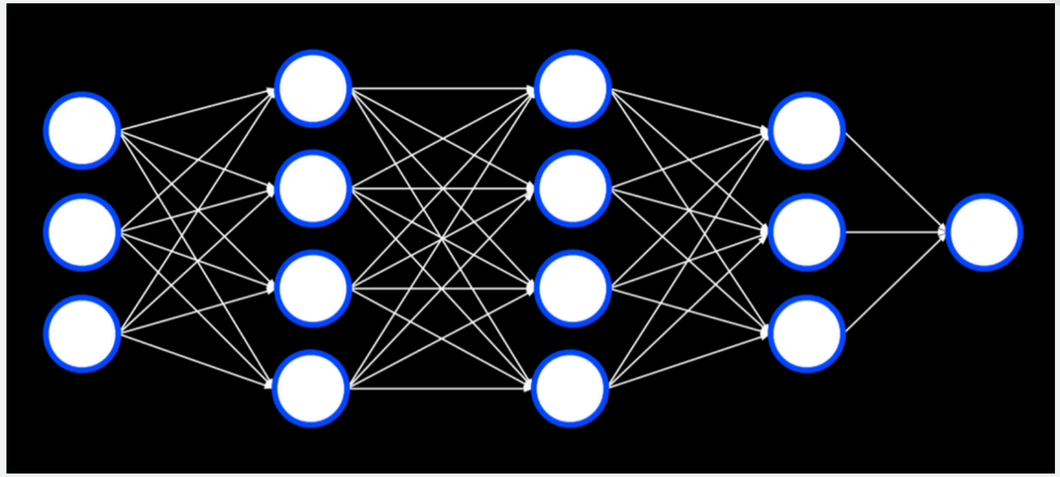
- Air-conditioning was not working in storage room
- Vegetables were rotting

Portion of blame:

Tracing contributors to loss (bad food)

BACKPROPAGATION

- In each hidden layer, every neuron contributes to the loss of a prediction because their weights are not “set correctly” to approximate the $f(x_1, \ldots, x_n) \rightarrow y$
- Idea of backpropagation is to move backwards from the loss at the output and distribute them layer by layer across neurons



GRADIENT OF WEIGHTS

Gradient of $W_{ij} = \frac{\partial L}{\partial w_{ij}}$

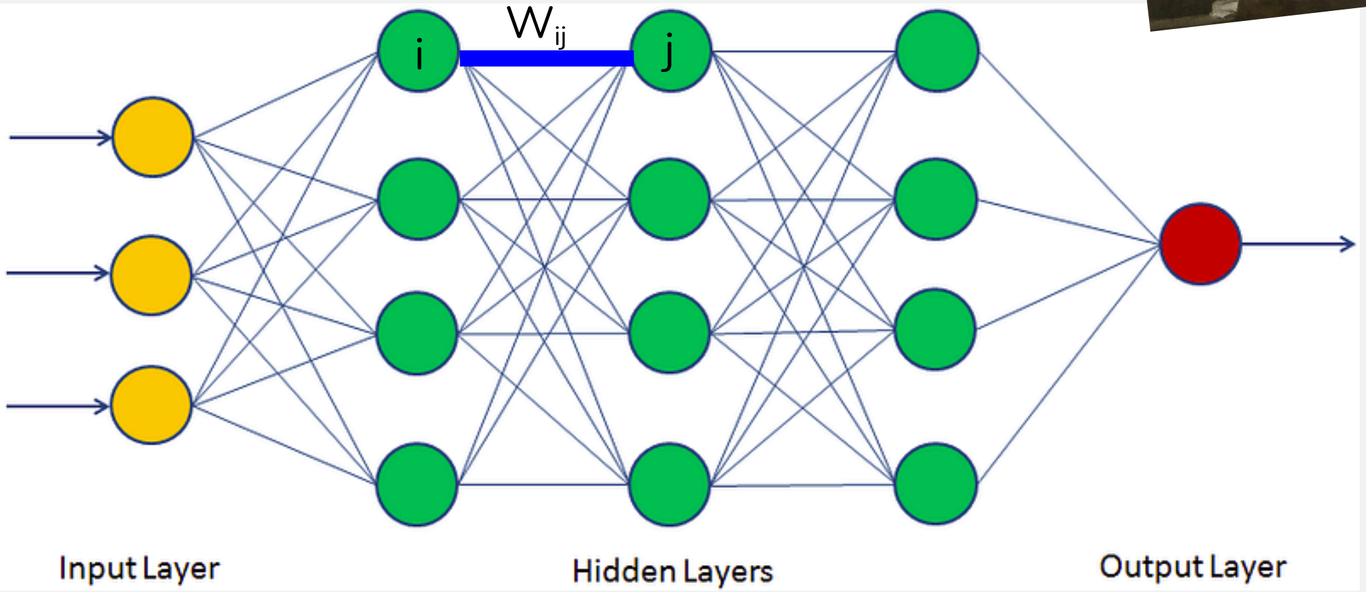
- It refers to how loss changes when W_{ij} changes slightly

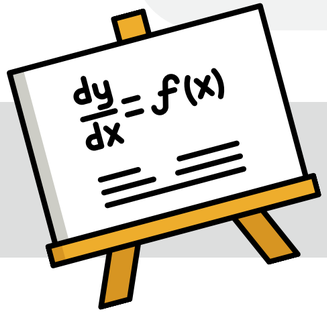
- How to compute gradient of W_{ij}

$$\frac{\partial L}{\partial w_{ij}} = \delta_j \times a_i$$

δ_j is the error distributed to neuron j

a_i is the output of neuron i

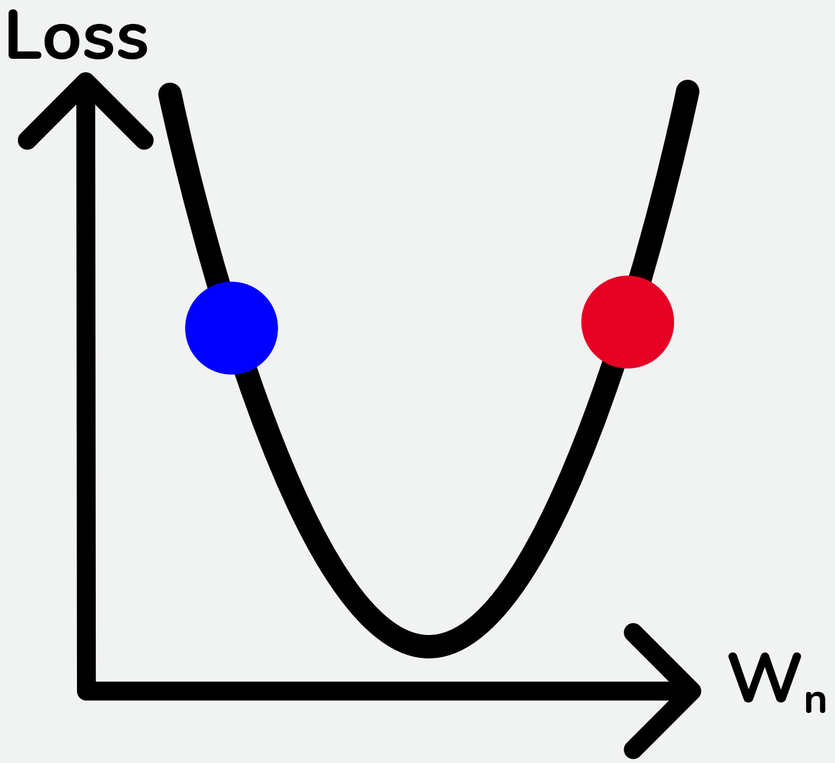




GRADIENT DESCENT

- Each point on the hills represents a possible weight value
- The height shows the loss for that weight
- Goal: move toward the valley (minimum loss).
- The gradient at a point tells us:
  - The direction of descent.
  - The steepness
- Gradient descent is using gradient at a value of W_n to update its value to decrease loss to minimum

GRADIENT DESCENT



Changing value of weight

- Adjust the value of W_n using this formula

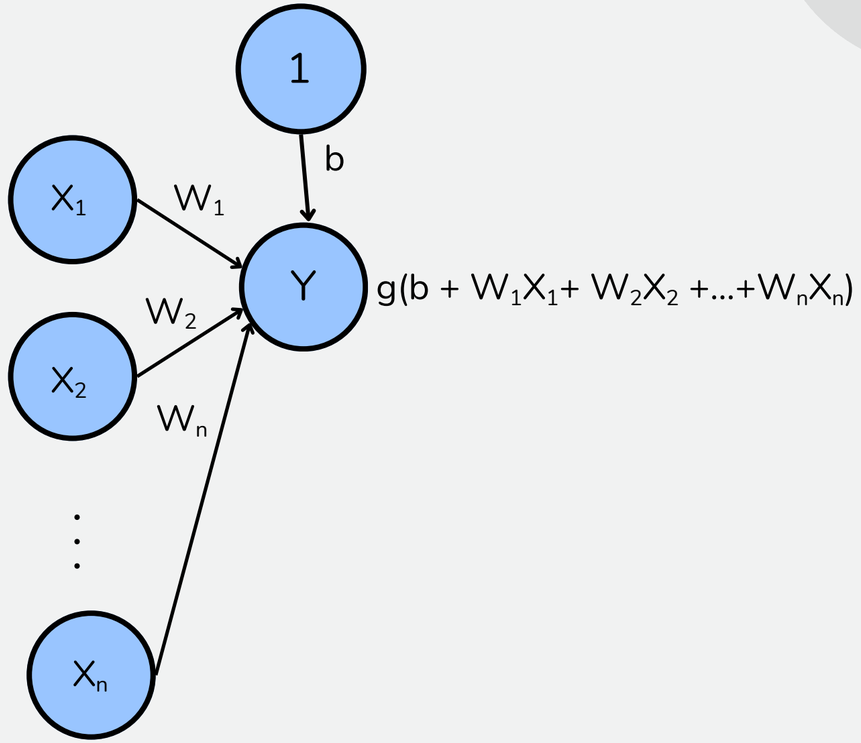
  $W_{n(new)} = W_{n(old)}$ - gradient * (learning rate)
- Learning rate is a hyperparameter to tune how fast the values of W_n are changed. Typically represented by η
- At 🔵:
  - (-gradient * learning rate) > 0
  - Formula increases W_n towards minimum loss
- At 🔴:
  - (-gradient * learning rate) < 0
  - Formula decreases W_n towards minimum loss

TRAINING ANN

Algorithm for adjusting weights and bias

For one training data point:

1. Forward passing (Making prediction with current weights)
2. Loss computed using the loss function
3. Backpropagation distributes error from the output layer to the layer after the input layer
4. Every neuron gets a portion of the error
5. Every weight in ANN obtains a gradient



TRAINING ANN

Algorithm for adjusting weights

6. Repeat 1 - 5 for all training data
7. Obtain an average of the gradient for all weights
8. Use gradient descent to update weights using the formula:

$$W_{n(new)} = W_{n(old)} - \text{gradient} * (\text{learning rate})$$

Very computationally intensive

TYPES OF GRADIENT DESCENT

Full batch gradient descent

- Compute the gradient of the loss for every sample in the dataset
- Take the average of all gradients to get one precise estimate

Accurate,stable updates

Memory intensive
Computationally intensive per update

Stochastic gradient descent

- Compute the gradient of the loss using one sample point
- Immediately update weight after each sample

Very fast updates and lightweight

High variance in updates, loss flunctuates

Mini-Batch gradient descent

- Split data into batches
- Compute gradient for each data point in the point and take average of the gradient
- Update weights once per batch

Balances speed and smoothness of convergence

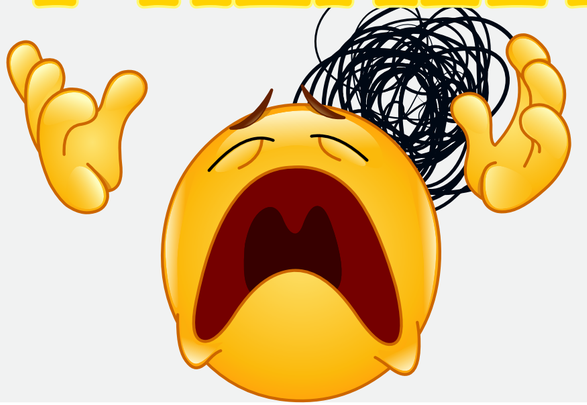
Require selection of batch size

ChatGPT, Gemini, Claude, DeepSeek, etc...

MLP is really complex and capable...

Do other fancy architectures like **Transformers** only use MLP?

NOT REALLY...



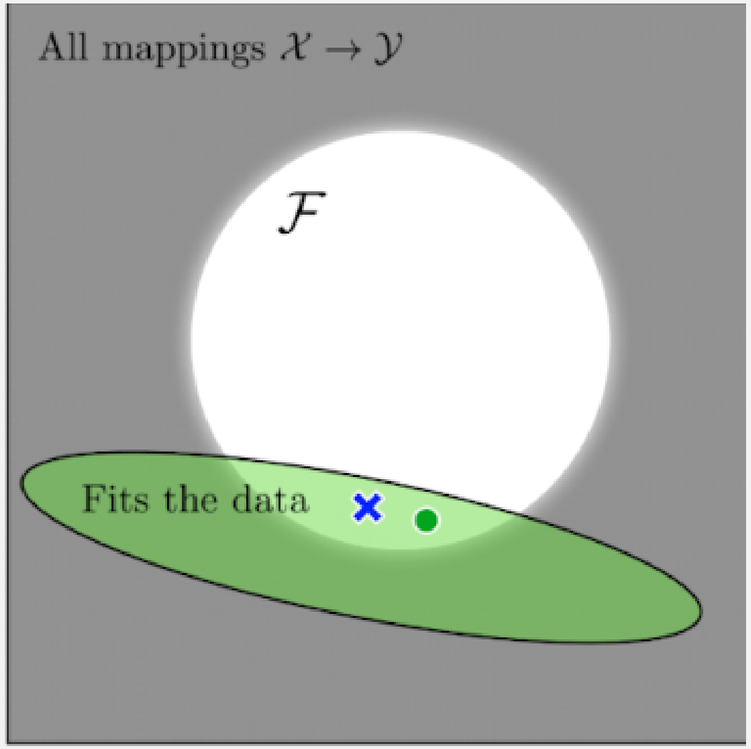
WHY NEED OTHER ARCHITECTURES?

1 **MLP is naively repetitive!**

Essentially **increasing parameters** to learn about the same data compared to other architectures **unnecessarily**.

2 **MLP has weak inductive bias[1]!**

Without inductive bias, MLP will have **a lot more different solutions** that it can find, and thus **have smaller chance to find the optimal one**!



[1] Inductive bias means assumption that a model makes about the data pattern. This will reduce the number of possible solutions that can be found.

simple

Moving from Tabular Data to... Image & Text Data

Complex

IMAGE DATA WALKTHROUGH

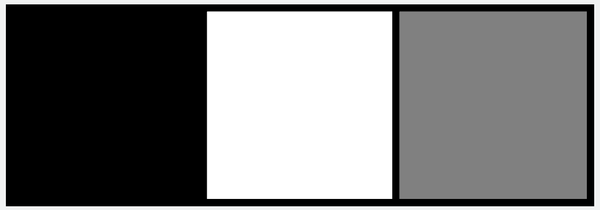
Property 1

Pixel is the **smallest unit** in image data, just like a biological cell in an organism. Pixel value **ranges from 0 (no light) to 255 (maximum light)**.

Property 2

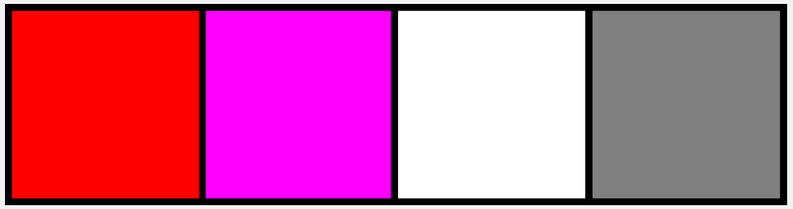
Images are made up of **channels**, commonly seen as RGB (3 channels) or grayscale (1 channel).

Grayscale



(0) (255) (?)

RGB



(255,0,0) (255,0,255) (255,255,255) ?

IMAGE DATA WALKTHROUGH

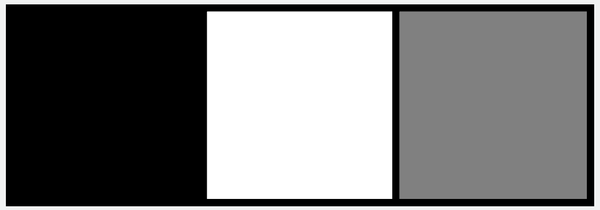
Property 1

Pixel is the **smallest unit** in image data, just like a biological cell in an organism. Pixel value **ranges from 0 (no light) to 255 (maximum light)**.

Property 2

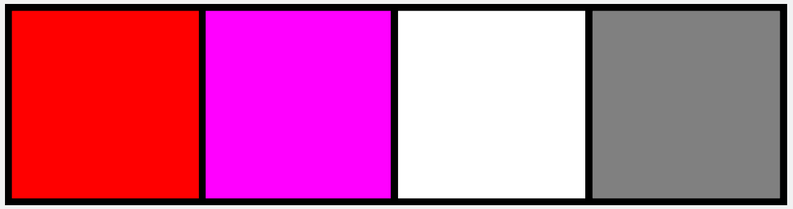
Images are made up of **channels**, commonly seen as RGB (3 channels) or grayscale (1 channel).

Grayscale



(0) (255) (128)

RGB



(255,0,0) (255,0,255) (255,255,255) ?

IMAGE DATA WALKTHROUGH

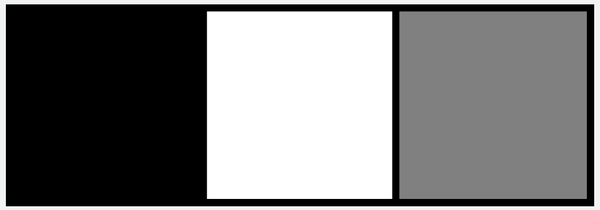
Property 1

Pixel is the **smallest unit** in image data, just like a biological cell in an organism. Pixel value **ranges from 0 (no light) to 255 (maximum light)**.

Property 2

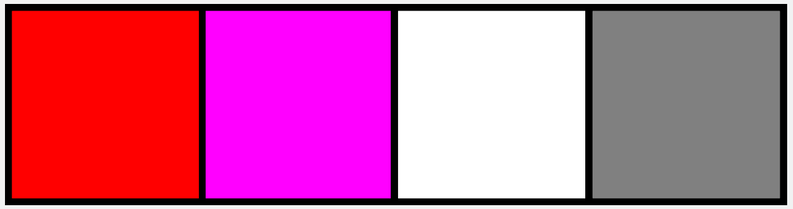
Images are made up of **channels**, commonly seen as RGB (3 channels) or grayscale (1 channel).

Grayscale



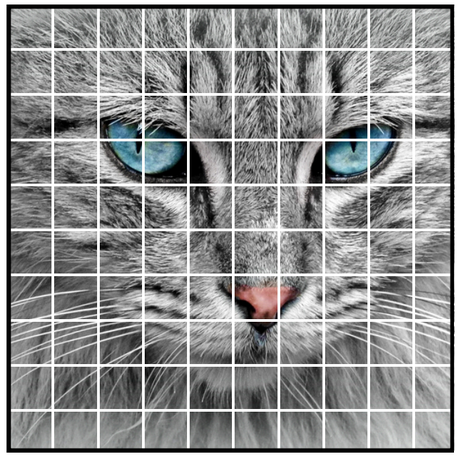
(0) (255) (128)

RGB



(255,0,0) (255,0,255) (255,255,255) (128,128,128)

RGB EXAMPLE



192	206	187	132	163	174	176	167	162	230
162	161	110	194	113	209	56	156	182	149
131	152	174	51	2	148	145	210	167	191
57	115	159	13	30	122	105	172	59	183
107	120	49	86	65	116	162	44	184	115
87	126	119	114	116	142	217	112	102	88
95	70	115	193	165	80	137	161	129	108
126	108	82	213	154	206	26	55	162	120
187	149	111	35	115	84	107	106	121	167
160	121	127	68	107	56	91	113	157	164

R

194	208	187	133	164	175	178	169	164	232
164	161	110	194	113	209	56	156	182	149
131	152	173	51	2	148	145	210	167	191
57	115	158	26	30	122	105	166	112	183
107	120	49	85	65	116	162	44	184	115
87	126	119	114	116	142	217	112	102	88
95	71	115	193	165	81	139	161	129	108
126	108	82	213	154	206	21	55	162	120
187	149	111	35	115	84	107	106	121	167
162	123	129	71	109	58	92	115	159	166

G

193	207	187	133	164	175	177	168	163	231
163	161	110	194	113	209	56	156	182	148
131	152	173	51	2	148	145	219	167	191
57	115	155	24	29	122	105	166	130	183
107	120	49	83	65	116	162	44	184	115
87	126	119	114	116	142	217	112	102	88
95	69	115	193	165	81	138	161	129	108
126	108	82	213	154	206	19	55	162	120
186	149	111	35	115	84	107	106	121	167
161	122	128	69	108	57	92	114	157	165

B

GRAYSCALE EXAMPLE

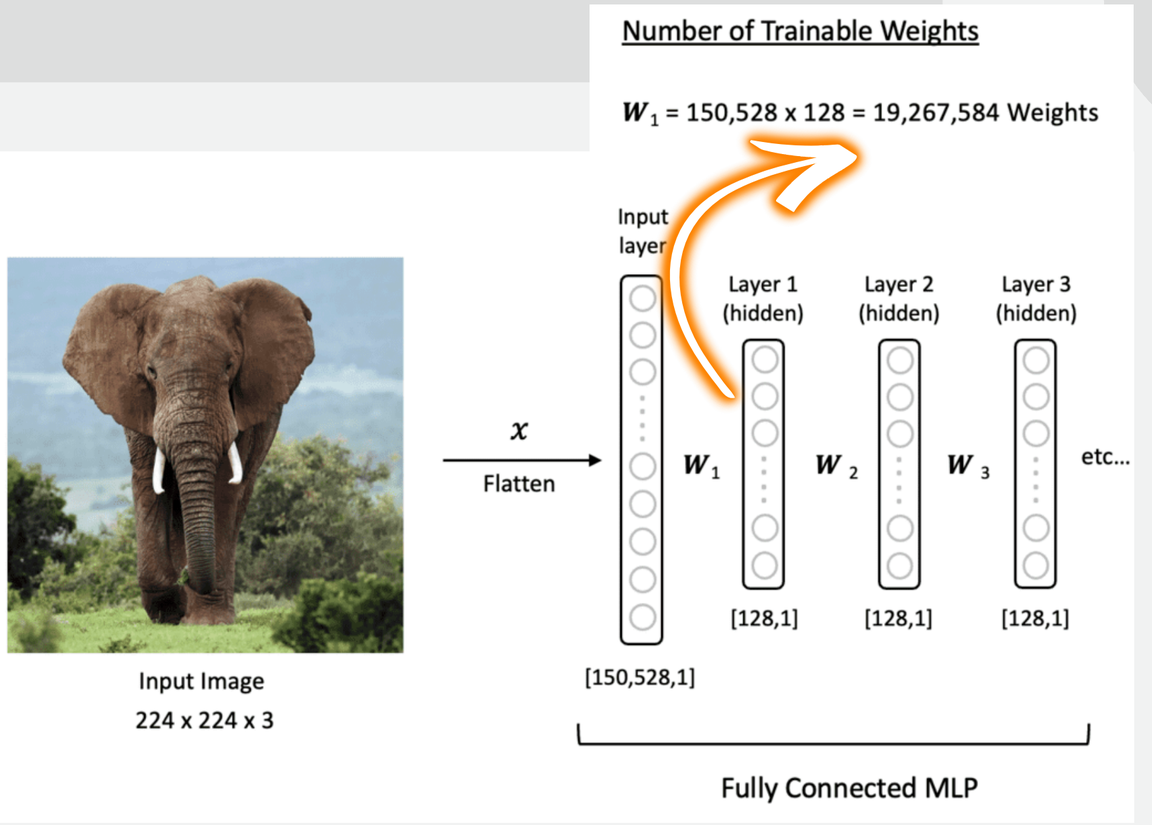
Image Credit: https://www.analyticsvidhya.com/blog/2021/03/grayscale-and-rgb-format-for-storing-images/

MLP ON IMAGE DATA

Large number of trainable parameters

MLPs use a single neuron for every input pixel in the image. So the **number of weights in the network rapidly becomes unmanageable**.

From Machine Learning, we already know that larger number of parameters will cause the model to also be highly **prone to overfitting** the training data



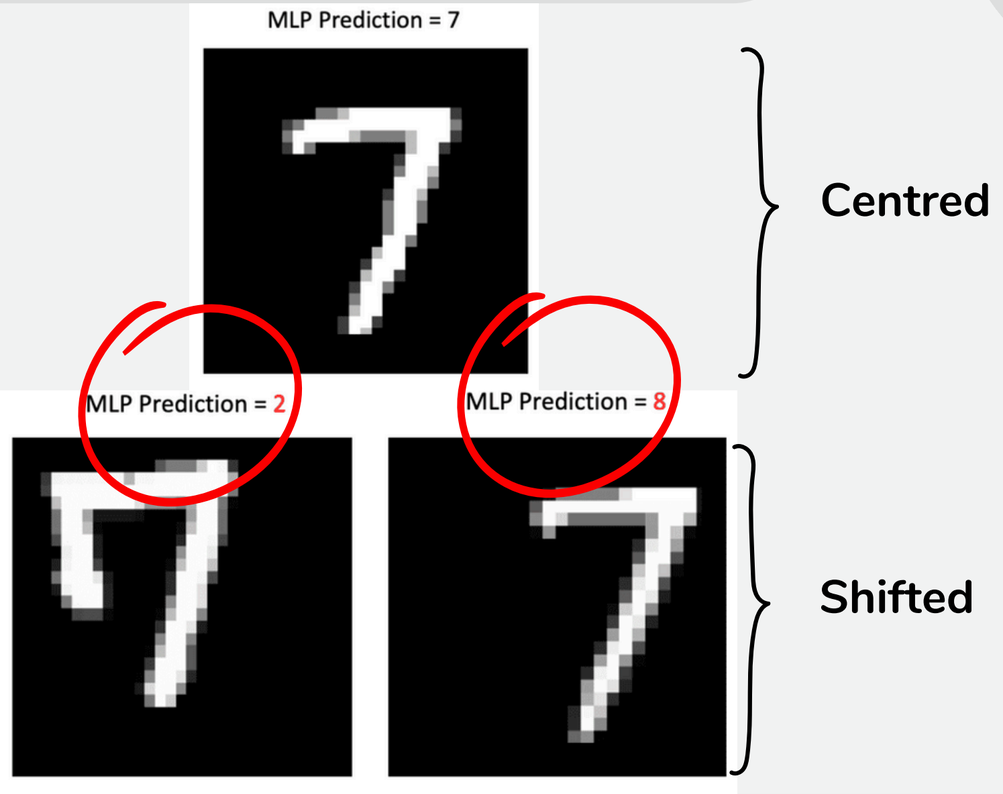
https://learnopencv.com/understanding-convolutional-neural-networks-cnn/

MLP ON IMAGE DATA

MLPs are not translational invariant

This means that the **network reacts differently** if the main content of the **image is shifted**. (Recall weak inductive bias!)

Since MLPs respond differently to shifted images, this can **complicate the model learning** process and **produce unreliable results**!



https://learnopencv.com/understanding-convolutional-neural-networks-cnn/

IMAGE PROPERTY II

Property 3

Small features repeat themselves and are often **combined into bigger features**.

NOTE!!!

Similar images are **almost always captured differently**. Our model needs to **recognise similar pattern** that exists elsewhere with **reasonable amount of parameters**.

CONVOLUTIONAL NEURAL NETWORK

Overview

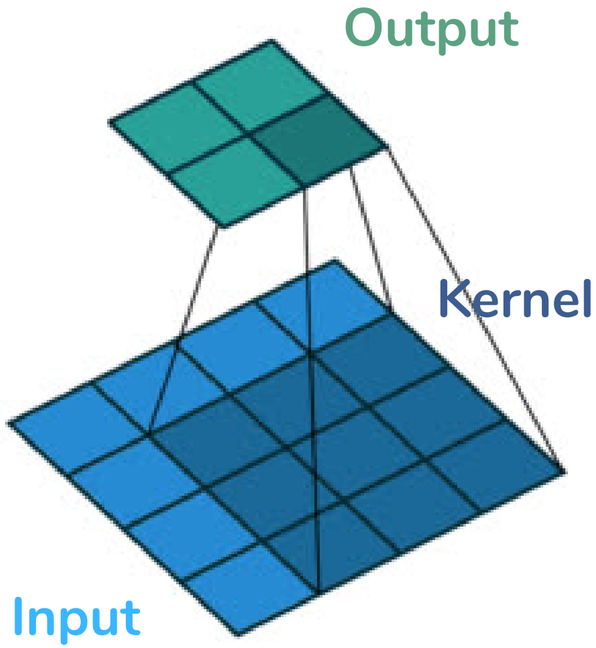
Developed to more **effectively** and **efficiently** process image data.

This is largely due to the use of **convolution operations** to extract features from images (**inductive bias of CNN**).

How does CNN work?

Learns kernels (filters) that can **exploit spatial relations in images**.

Each kernel moves across the image to process different parts of the image and detects **feature patterns** that are translational invariant.



BENEFITS OF CNN INDUCTIVE BIAS

Reduce trainable parameters

Kernel has **weight-sharing** properties, i.e., same weights are used to process different parts of the image.

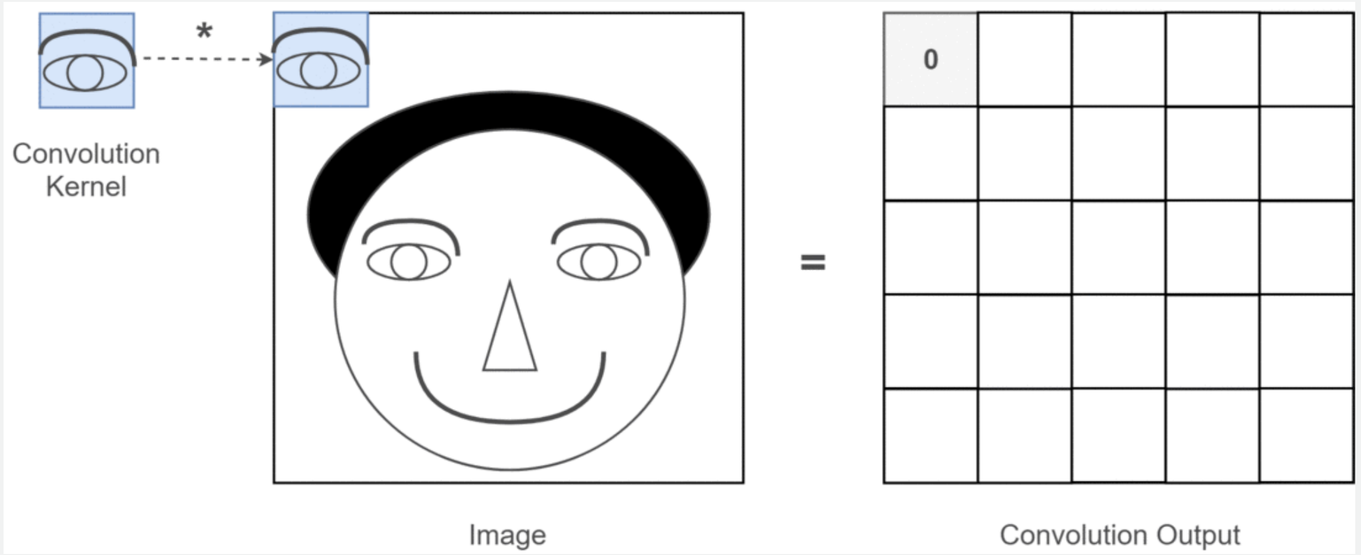


Image Credit: https://medium.com/data-science/light-on-math-machine-learning-intuitive-guide-to-convolution-neural-networks-e3f054dd5daa

BENEFITS OF CNN INDUCTIVE BIAS

Translational Invariance

Even if image is shifted, kernels can still **detect the learned pattern elsewhere**.

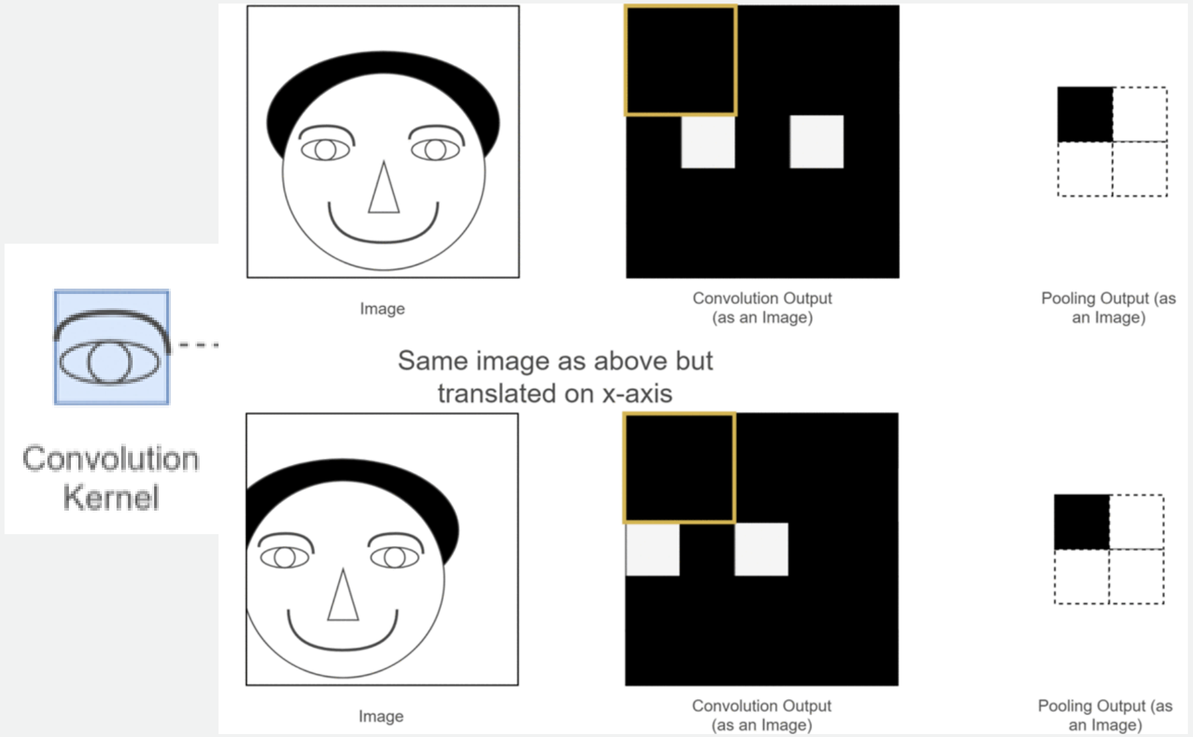


Image Credit: https://medium.com/data-science/light-on-math-machine-learning-intuitive-guide-to-convolution-neural-networks-e3f054dd5daa

CONVOLUTION OPERATION

Step by step

1. Place the **kernel** over a portion of the **input**
2. Multiply the elements of the **kernel** with the corresponding elements of the **input** + Sum
3. Repeat by sliding the **kernel** over the **input** image until the **kernel** has been placed over each **input** section.

The **resulting single number** is the **output** of the convolution operation for a given filter location.

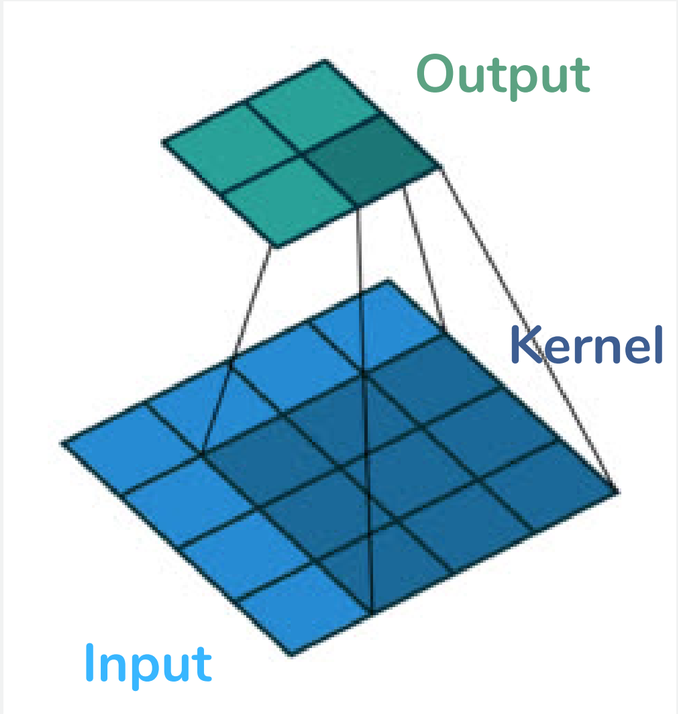
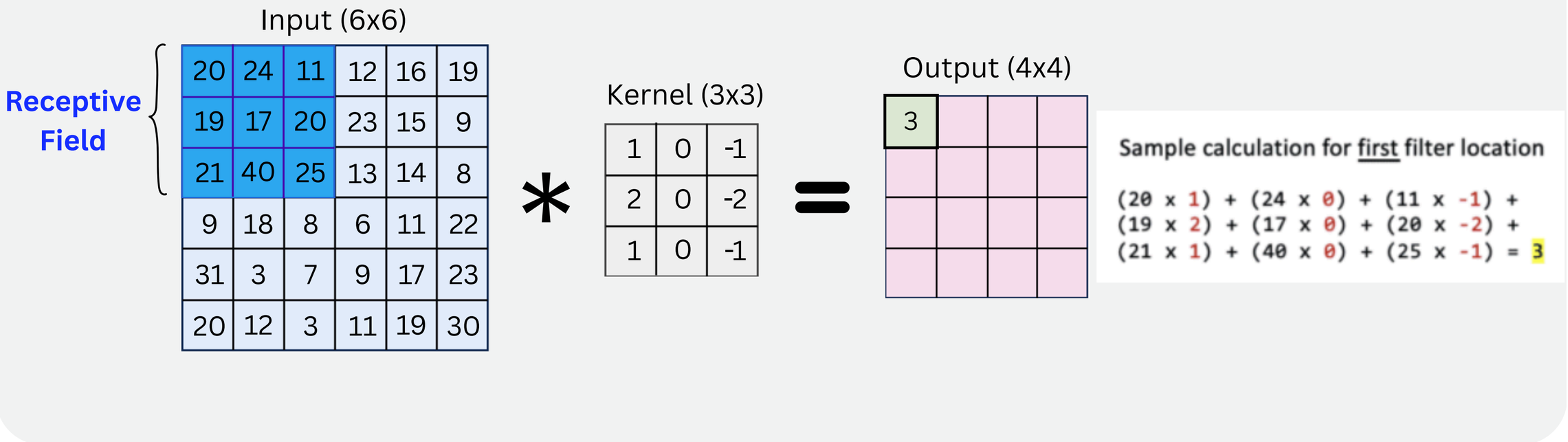


Image taken from: https://learnopencv.com/understanding-convolutional-neural-networks-cnn/

EXAMPLE WALKTHROUGH



Adapted from: https://learnopencv.com/understanding-convolutional-neural-networks-cnn/

EXAMPLE WALKTHROUGH

Input (6x6)

Receptive Field (3x3)

20	24	11	12	16	19
19	17	20	23	15	9
21	40	25	13	14	8
9	18	8	6	11	22
31	3	7	9	17	23
20	12	3	11	19	30

*

Kernel (3x3)

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

=

Output (4x4)

3	?		

Adapted from: https://learnopencv.com/understanding-convolutional-neural-networks-cnn/

EXAMPLE WALKTHROUGH

Input (6x6)

Receptive Field (3x3)

20	24	11	12	16	19
19	17	20	23	15	9
21	40	25	13	14	8
9	18	8	6	11	22
31	3	7	9	17	23
20	12	3	11	19	30

*

Kernel (3x3)

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

=

Output (4x4)

3	27		

Sample calculation for second filter location

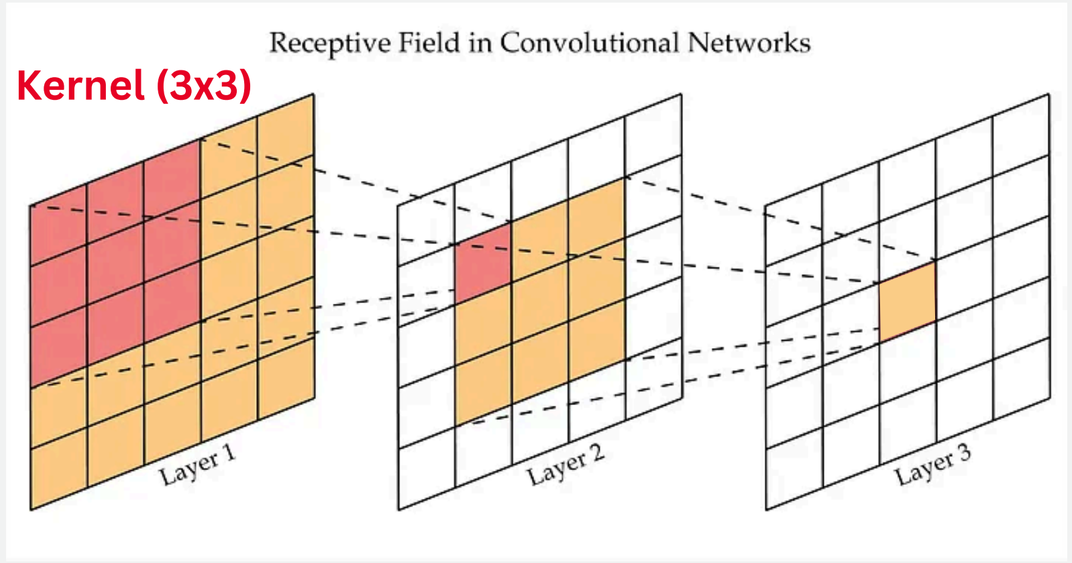
```
(24 x 1) + (11 x 0) + (12 x -1) +
(17 x 2) + (20 x 0) + (23 x -2) +
(40 x 1) + (25 x 0) + (13 x -1) = 27
```

Adapted from: https://learnopencv.com/understanding-convolutional-neural-networks-cnn/

RECEPTIVE FIELD

Overview

Receptive field refers to the **size of the input image** that a neuron (pixel) in a given layer can "see".



1 pixel in **Layer 2** can “see” **3x3 pixels in Layer 1**.
1 pixel in **Layer 3** essentially can “see” **5x5 pixels in Layer 1**.

RELATING TO CNN LEARNING

Hierarchical feature learning

In CNN, **early layers** with **small receptive fields** detect **simple features** (like edges). **Deeper layers** with progressively **larger receptive fields** combine these features to **recognise complex objects**.

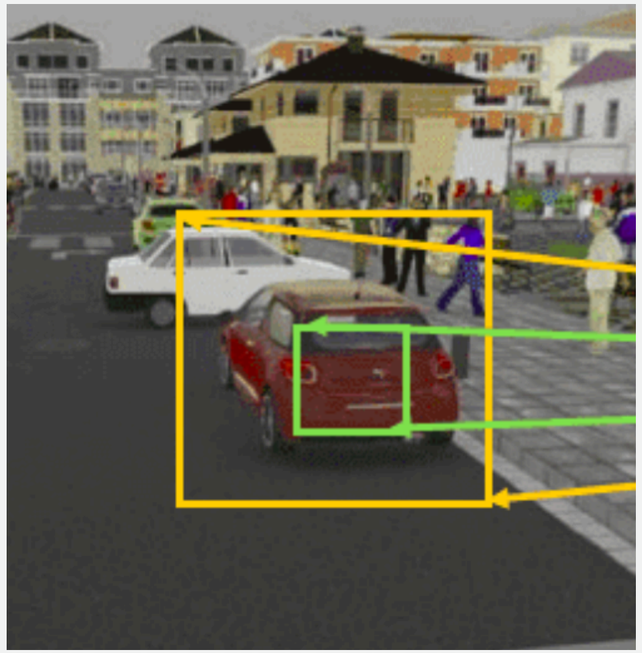
This pixel at output layer can actually “see” the details of the camel to make prediction.

Image adapted from: https://learnopencv.com/cnn-receptive-field-computation-using-backprop/

RECEPTIVE FIELD

Why is RF important?

We want to ensure the pixel at deep layer (output layer) to have **big receptive field**, so as to ensure that **no crucial information** was **not taken into account**.



We prefer **orange** receptive field as it captures more information about the car.

Image adapted from: https://theaisummer.com/receptive-field/

STRIDING

Overview

Stride is the number of pixels by which we slide our kernel matrix over the input. The operation happens from left-to-right, top-to-bottom across an image.

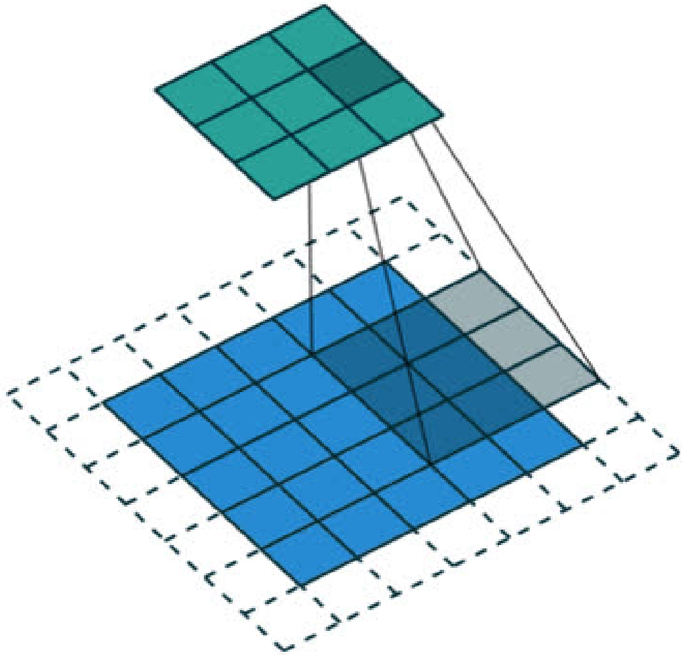
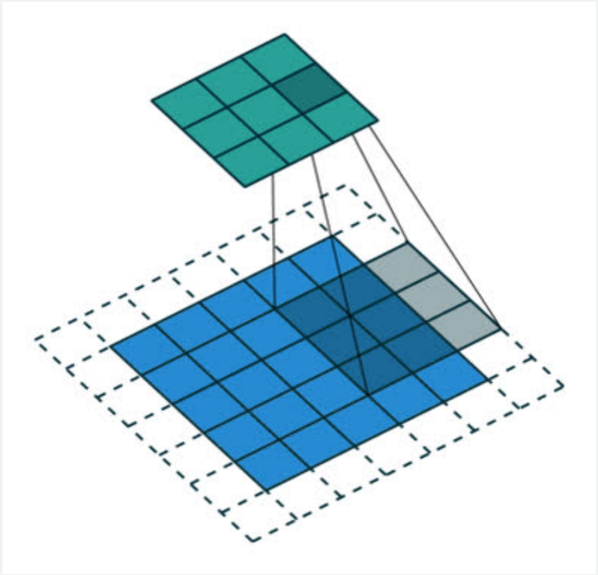


Image Credit: https://commons.wikimedia.org/wiki/File:Convolution_arithmetic_-_Padding_strides.gif

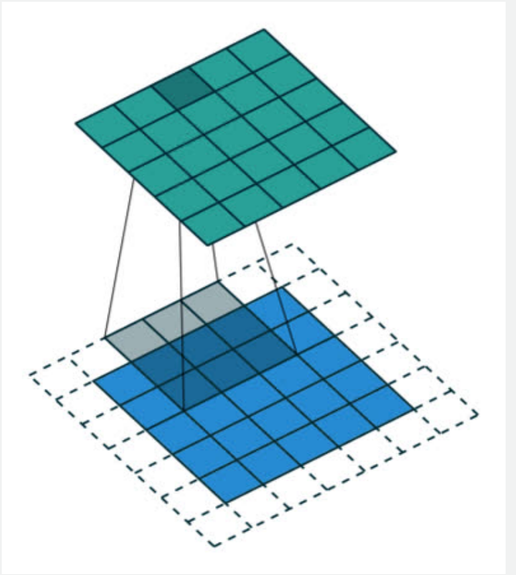
STRIDING

Key Note

Smaller strides will lead to **overlapping receptive fields** and **larger output** size. Conversely, **larger strides** will result in **less overlapping** receptive fields and **smaller output** size.



Stride = 2



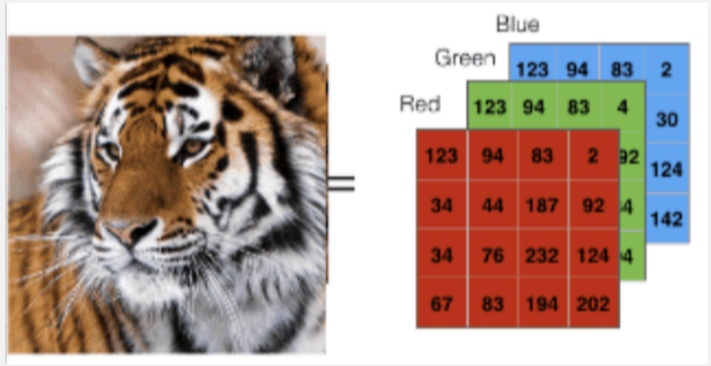
Stride = 1

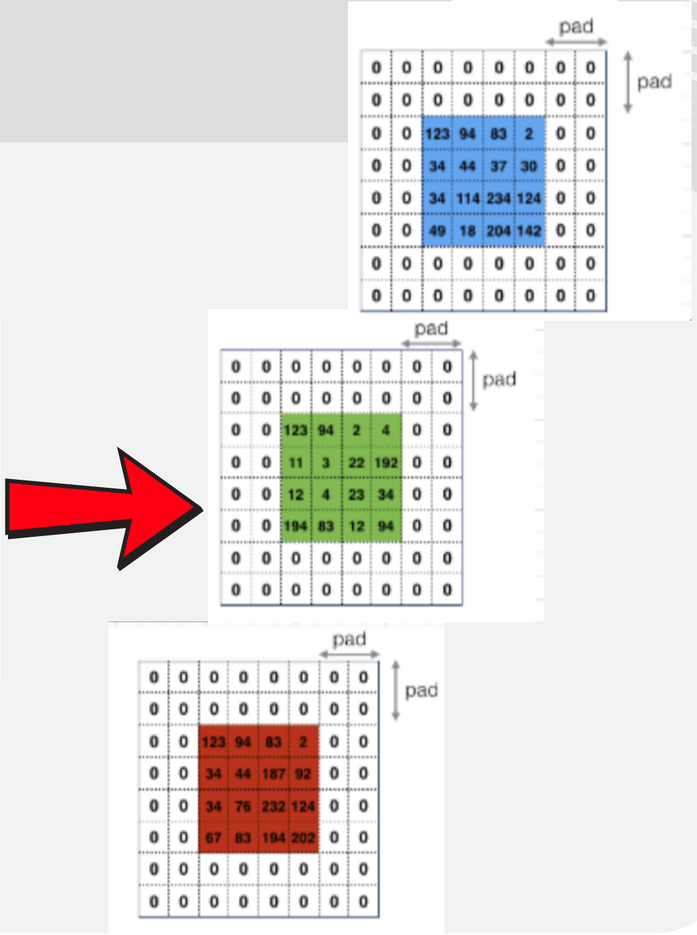
Image Credit: https://commons.wikimedia.org/wiki/File:Convolution_arithmetic_-_Padding_strides.gif, https://learnopencv.com/understanding-convolutional-neural-networks-cnn/

PADDING

Overview

Refers to the process of **adding extra pixels** (usually zeros) around the border of the images.





PADDING

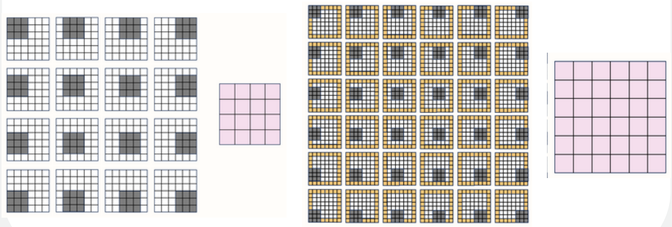
Why is it important?

As network becomes deeper, **convolution output will get smaller**.

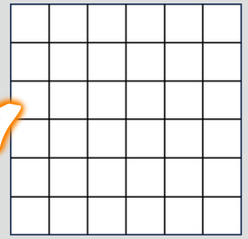
Only the **middle part** of the **image gets more focus**, and we lose information at the border.

But... We can tackle...

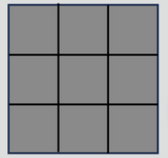
By adding **sufficient pixels**[2] around the border, we can maintain the feature map's dimension after convolution!



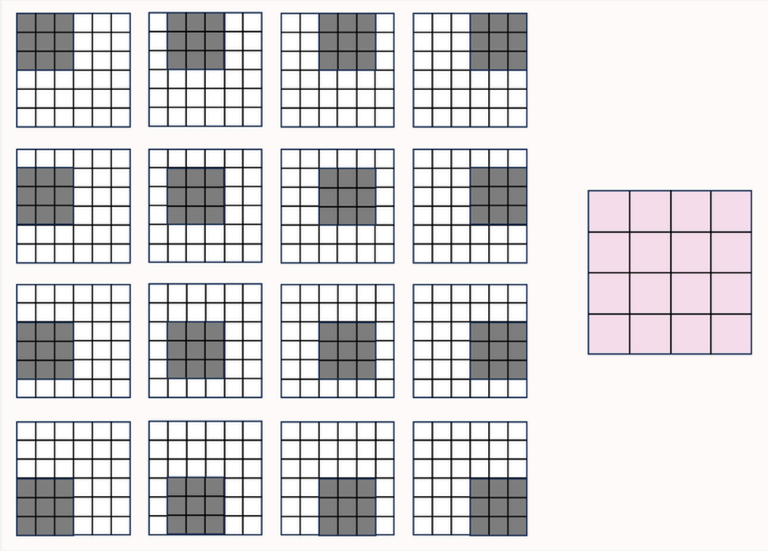
[2] Number of pixels to be padded can be calculated using the general formula, P = ((O - 1) * S - I + F) / 2, where P is the padding applied to one side, O is the output size, S is the stride value, I is the input size and F is the filter (kernel) size. For "Same" padding (i.e., output size = input size), P = (F - 1) / 2.



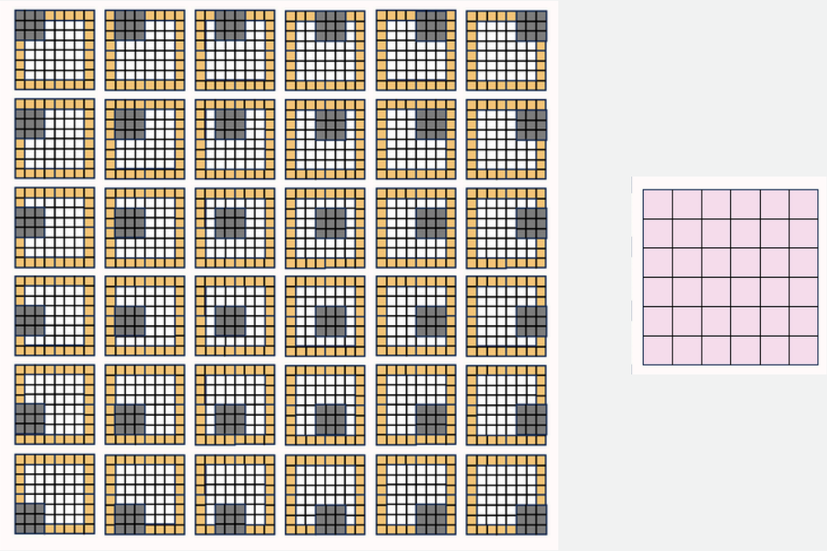
PADDING EXAMPLE



Let’s say our image has a 6x6 dimension, and the current Convolutional Layer is working on a 3x3 kernel.



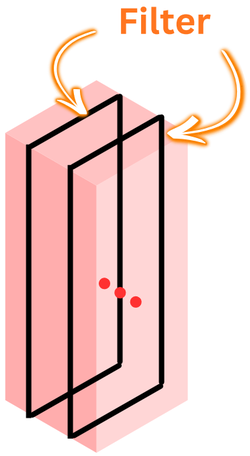
Without padding, we will get a 4x4 output after the convolution.



By adding one layer of padding to all sides, we will first get a 8x8 input. After the convolution, we will get a 6x6 output, which has the same dimension as before the convolution.

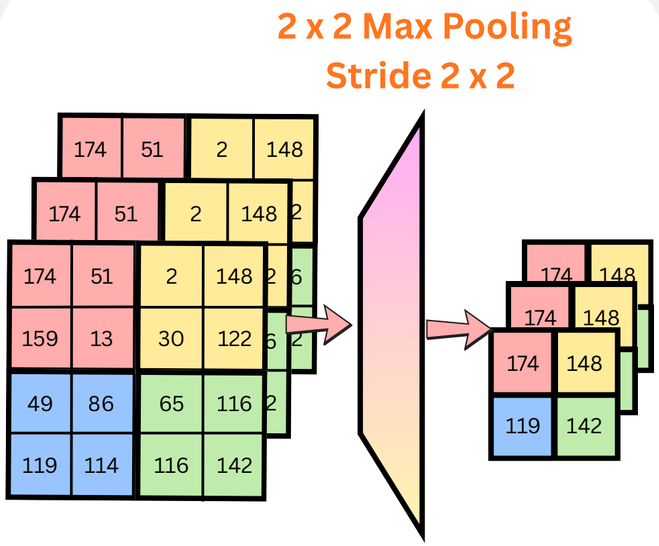
TYPES OF CNN LAYER

Convolutional Layer

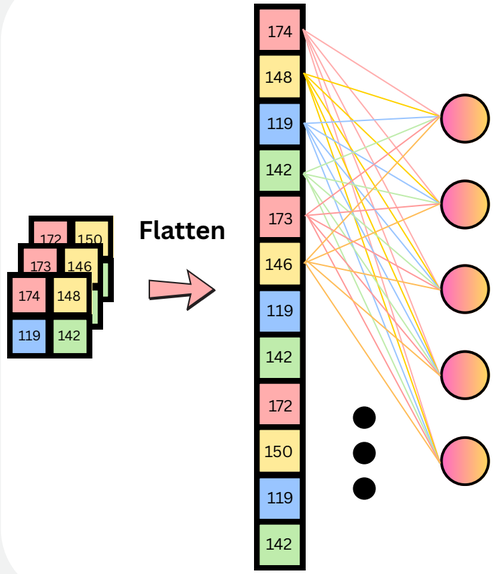


Each convolutional layer can have **different number of filters** which learn different feature.

Pooling Layer

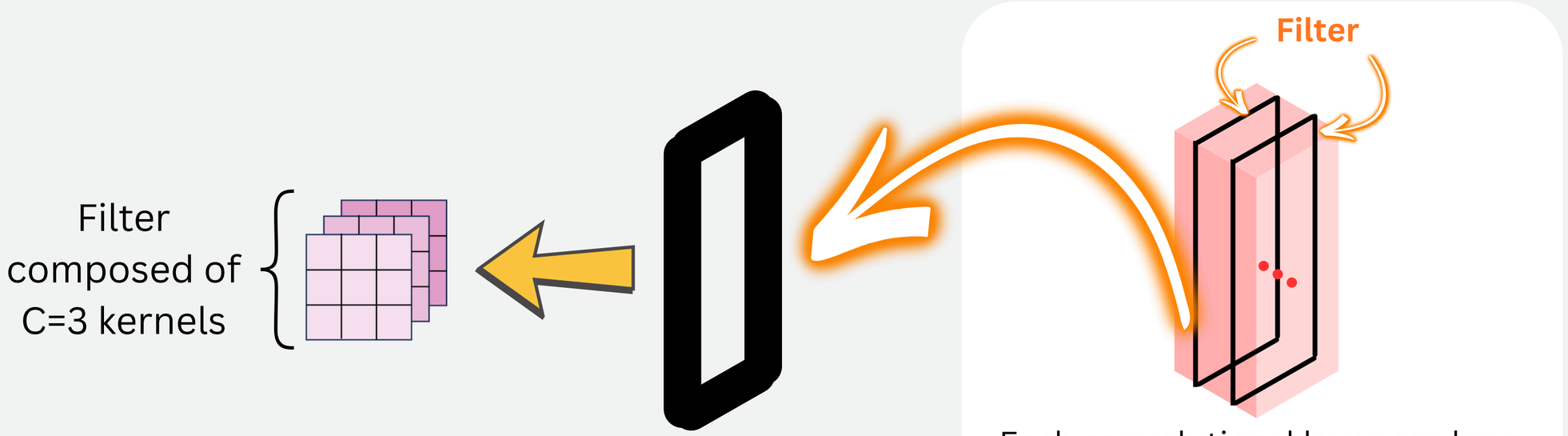


Fully Connected Layer



Convolutional Layer

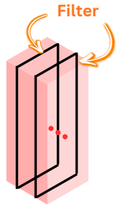
CONVOLUTIONAL LAYER



Remark:

Kernels refer to the 2D weight matrix, while filters refer to the 3D structure of multiple kernels stacked together.

CONVOLUTIONAL LAYER



Each convolutional layer can have **different number of filters** which learn different feature.

Overview

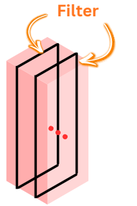
Consists of multiple **learnable** filters, say **K**. Each filter convolves on **all channels**, **C**, of the input. Hence, **C** depends on the number of channels from the input to this layer.

Key Features: Parameter Sharing

The **same set of weights** in **each filter** is used to process different parts of the input image, hence **significantly reducing the total number of trainable parameters** compared to MLP.

Summary: Conv. layer has K (3D) filters of C * H * W dimension!

CONVOLUTIONAL LAYER



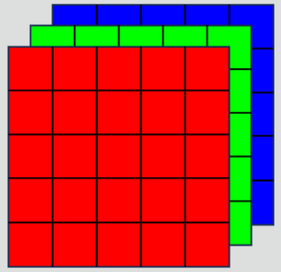
Each convolutional layer can have **different number of filters** which learn different feature.

Output: Feature Map

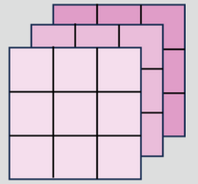
The **output** of the **convolution operation** is known as 2D **feature map**. Convolutional layers typically contain **many filters**, meaning each convolutional layer produces **multiple feature maps**.

Small Note

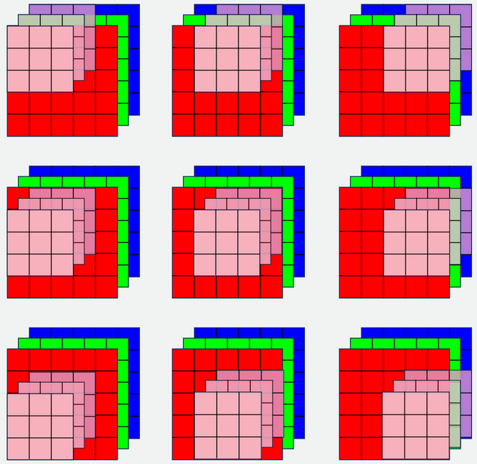
The **number of filters in a convolutional layer** is a **design choice** and dictates the number of feature maps produced in the output.



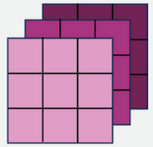
WALKTHROUGH



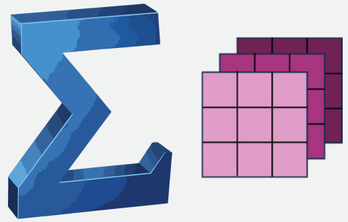
Filter composed of C=3 kernels



=

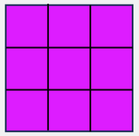


3D map



Pixel sum

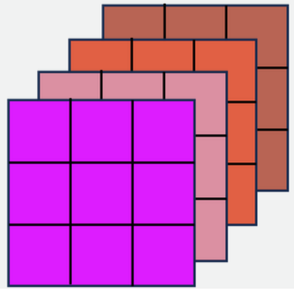
=



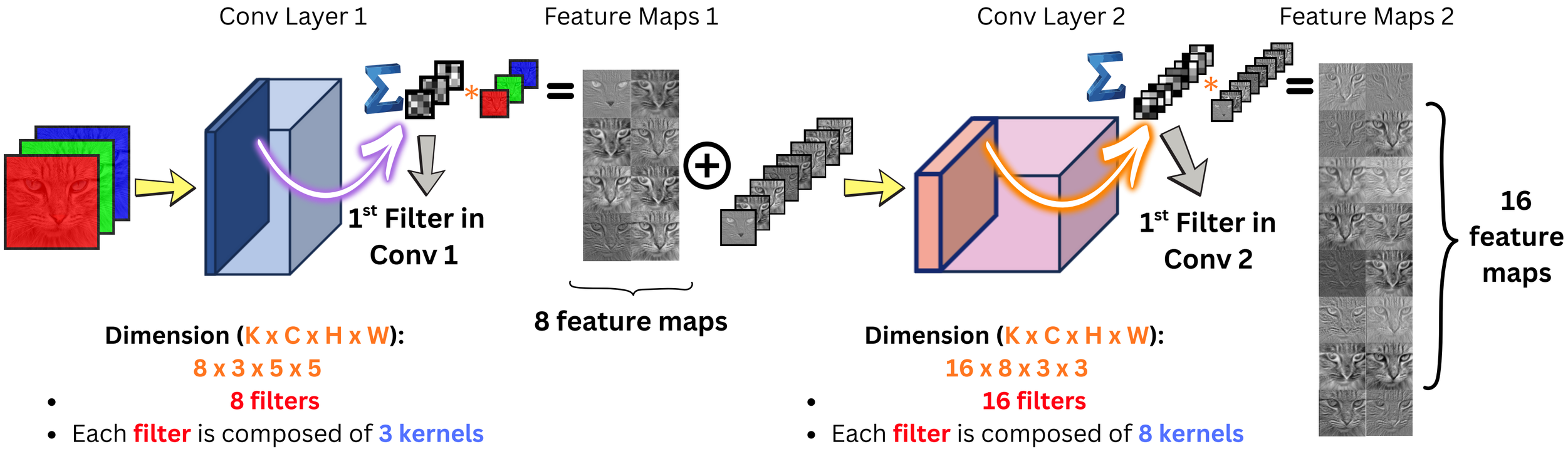
Feature map

Result from one filter only!

Say I have 4 filters in this convolutional layer



EXAMPLE



Dimension (K x C x H x W):

8 x 3 x 5 x 5

- **8 filters**
- Each **filter** is composed of **3 kernels**
- Each **kernel** is a 5 x 5 matrix

Dimension (K x C x H x W):

16 x 8 x 3 x 3

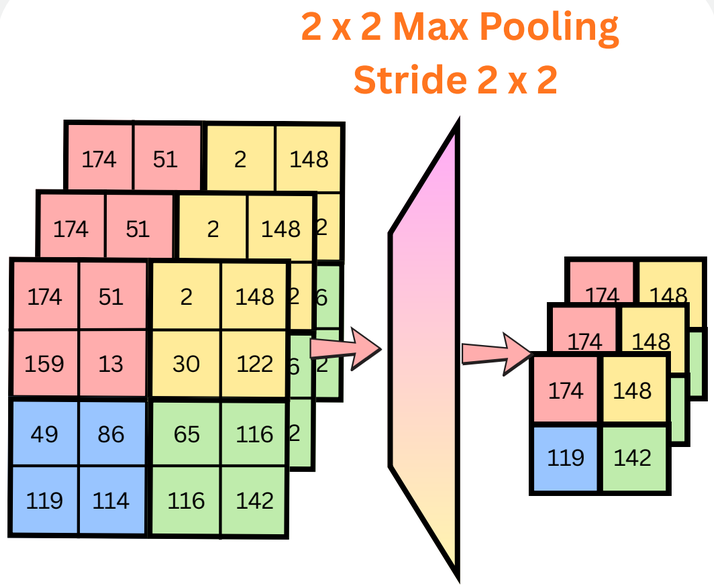
- **16 filters**
- Each **filter** is composed of **8 kernels**
- Each **kernel** is a 3 x 3 matrix

Pooling Layer

POOLING LAYER

Overview

A layer that **does not have learnable parameters**, mainly used to **downsample + aggregate dispersed information** into smaller vectors while **retaining important information**.



PURPOSE OF POOLING

1 **Reduce the spatial dimension!**

This means we are “distilling” the information from the image, eventually leading to **smaller kernel size needed** (i.e., **less** number of total **parameters to learn**) → **Reduce overfitting**.

2 **Provide certain level of translational invariance!**

By **maintaining** the most **prominent feature information regardless** of their **exact** pixel **position**, it helps CNN recognise patterns/objects elsewhere in the image.

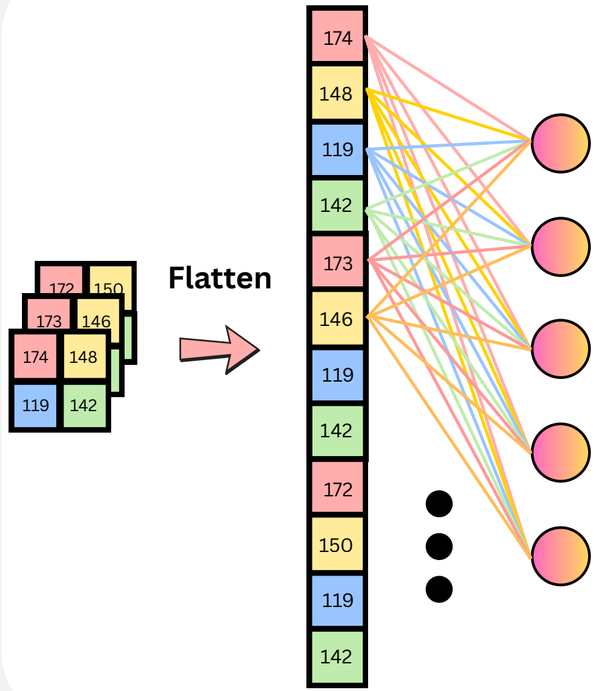
Fully Connected (FC) Layer

FULLY CONNECTED (FC) LAYER

Overview

FC layer is a layer where **every neuron in the previous layer is connected to every neuron in the current layer**, placed **right before the output layer**.

FC layers will **combine features** learned by the kernels to let the network **form a global understanding about the image**.



Output Layer

OUTPUT LAYER

Final layer[3] that produces the predictions of the model.

Multiclass Classification

Number of neurons:
N (depending on the number of classes)

Activation Function:
Softmax, so each neuron represents the probability of the instance belongs to that class.

Binary Classification

Number of neurons:
1

Activation Function:
Sigmoid, the output represents the probability of the instance belongs to the positive class (Class 1).

Regression

Number of neurons:
1

Activation Function:
None, the neuron just outputs the scalar value obtained from the linear combination in FC layer.

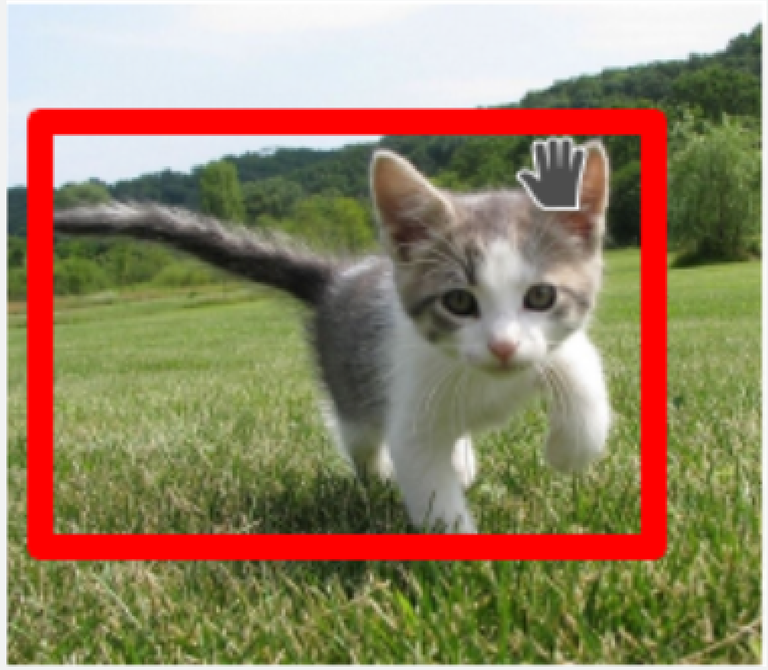
[3] The output layer is just another FC layer, with the number of neurons depending on the output of the task.

QUICK QUIZ

Object Detection

Is this a Classification or a Regression?

How many neurons will the output layer have?



SHORTCOMINGS

1 **CNNs are not naturally rotational invariant![4]**

While training **more kernels** tends to allow CNNs to **learn** the **rotated versions of features**, which indirectly makes them rotational “invariant”, we are **not guaranteed** it will happen!

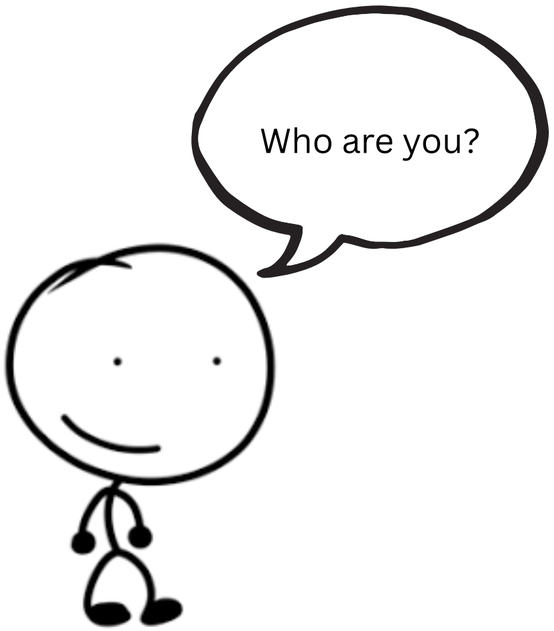
2 **CNNs might have hard time capturing the global pattern!**

Each **kernel** only **operates locally** (i.e., only some pixels are convoluted at a time). Though deeper CNN can access the entire image, it might still **fall short in learning global patterns**!

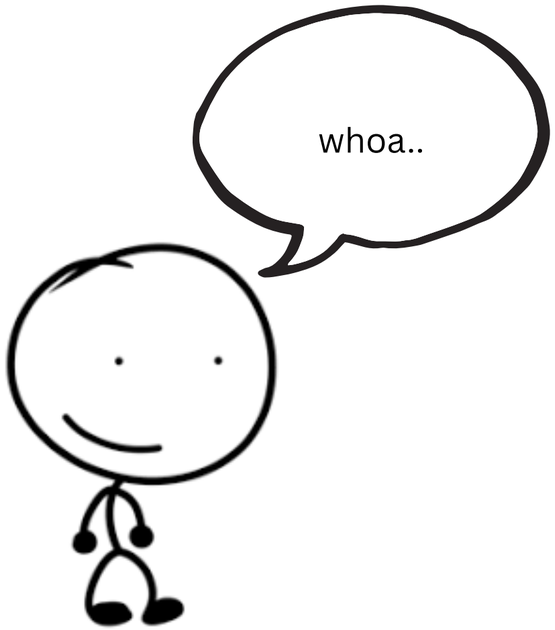
[4] Use Rotation-Invariant Coordinate CNN (RIC-CNN) if you really need both invariance in CNN. Or look into Vision Transformer for better capture of global context.

TRANSFORMERS

TRANSFORMERS



TRANSFORMERS



Why Transformers changed the game

Traditional Models:
Traditional models (like RNNs) processed text sequentially
(Word 1 → Word 2 → Word 3).
This step-by-step method was very slow and created a major processing bottleneck.

Transformers:
Transformers process the entire sentence at once (Word 1 | Word 2 | Word 3).
This "parallel processing" is much faster.

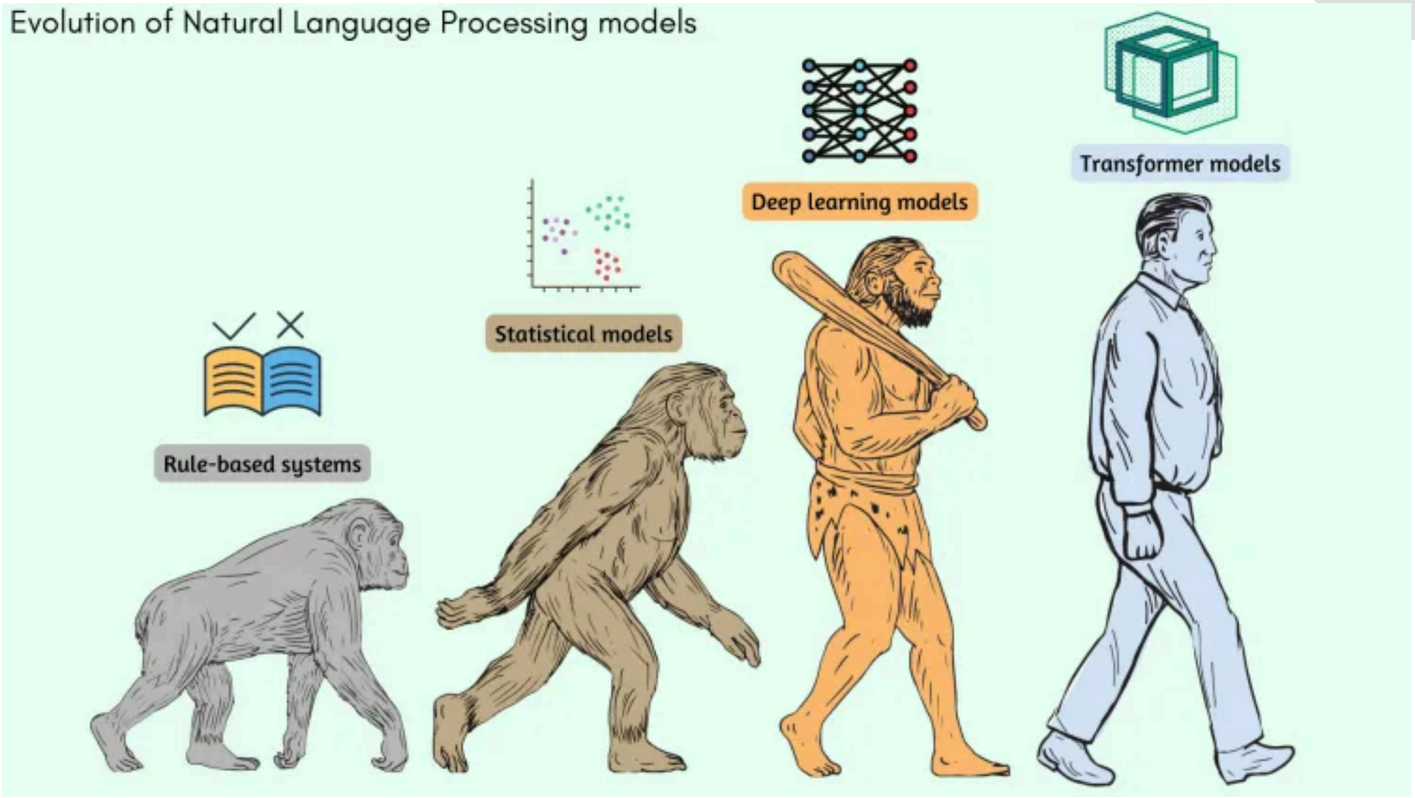


Image credit: https://medium.com/@srechakra/from-rulesets-to-transformers-a-journey-through-the-evolution-of-sota-in-nlp-43033d8422c5

Why Transformers changed the game

Traditional Models:
Older models had short memory and "forgot" context in long sentences, failing to grasp complex relationships.

Transformers:
Transformers use Self-Attention, where every word interacts with every other word at once. This provides "global context," letting words from the start and end of a sentence connect easily to understand complex meaning.

Traditional Models:

Transformers:

Why Transformers changed the game

Traditional Models:
Adding more layers to older models often led to diminishing returns or made them worse at their job.

Transformers:
Transformers scale exceptionally well.
This means you can make them more powerful by:

- Adding more data
- Adding more layers
- Adding more compute power

Why Transformers changed the game

Traditional Models:
Previously, AI models were highly specialised. The best model for text was very different from the best model for images or audio.

Transformers:
The Transformer architecture proved to be incredibly flexible. The same core design that "attends" to words in a sentence can be adapted to "attend" to:

- Patches of an image (e.g., Vision Transformers, ViT)
- Snippets of an audio waveform (e.g., Whisper)
- Segments of code (e.g codex)
- And more...

Tokenization

Input sentence is broken into pieces called **tokens**.

You can think of these as smart words or sub-words.

Input: I am a transformer"
Tokens: ["I", "am", "a", "transform", "##er"]

(Note: "transformer" might be split to handle variations of the word)

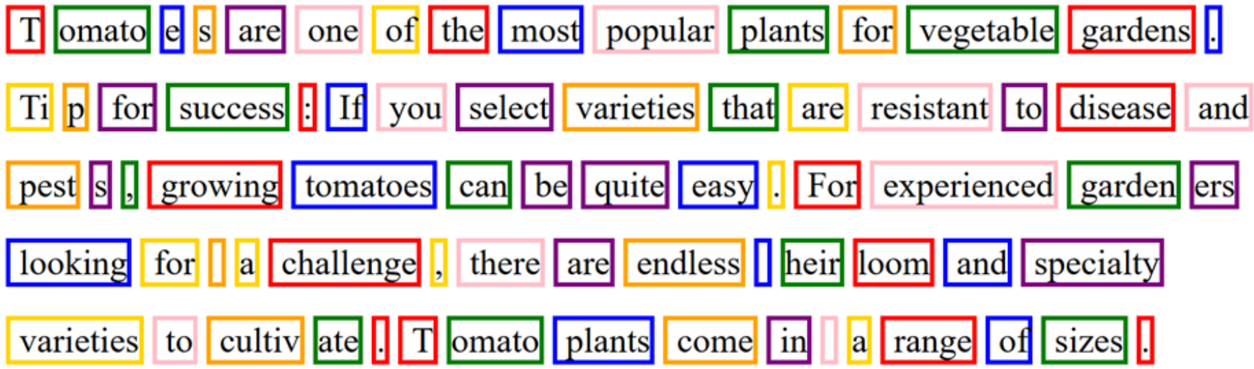


Image credits: https://www.ibm.com/think/topics/tokenization

Embedding

Computers can't read words.

So, we convert each token into a list of numbers called an embedding.

This list (or vector) represents the token's meaning.

For example, the vector for "dog" would be numerically similar to the vector for "puppy."

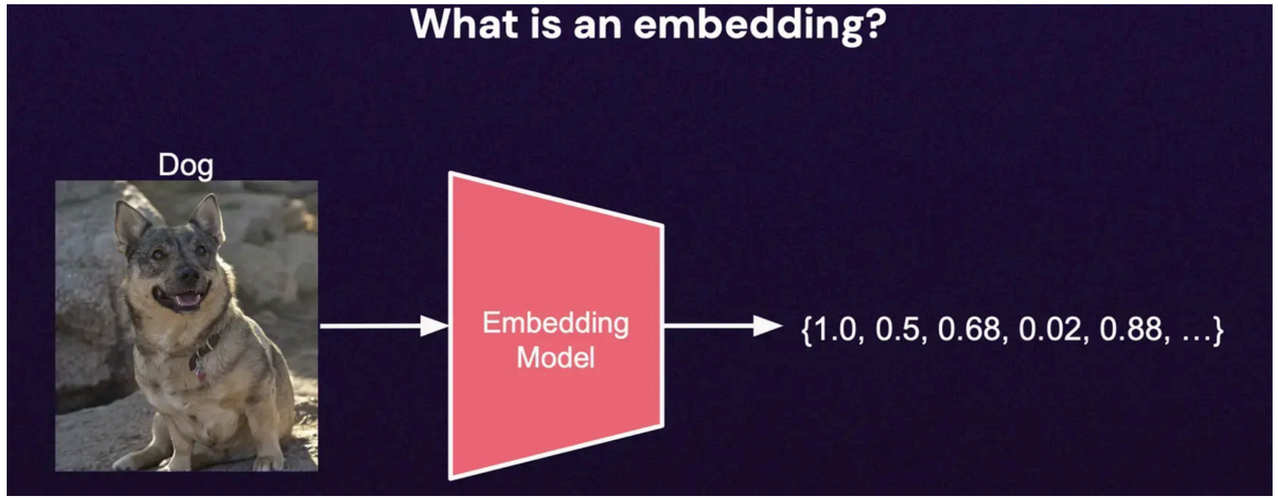


Image credit: https://www.superannotate.com/blog/embeddings-in-ml

Problem!!!!

Computers are **Order-Blind**

- When a Transformer processes words in parallel (all at once), it loses information about their **original order**.

- To the model, the words in "The dog bites the man" and "The man bites the dog" might look identical if we only use their meaning (embeddings).

- Their meanings are the same (dog, bites, man), but the order changes everything!

Positional Encoding

For each word's embedding, we create a **unique** position vector.

We simply add the position vector to the word's embedding vector.

Example:
"dog" (meaning) + Position 2 (order) = **Modified Vector** for "dog" at Position 2
"man" (meaning) + Position 4 (order) = **Modified Vector** for "man" at Position 5

Now, the final vector for each word holds both its meaning **AND** its specific place in the sentence.

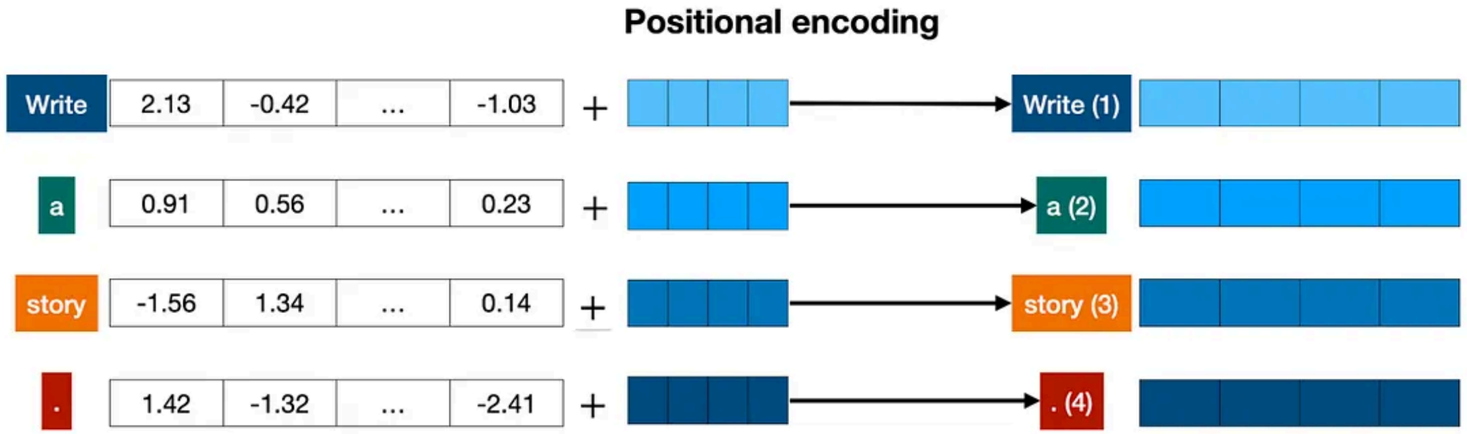


Image credits: https://medium.com/@lordmoma/the-transformer-model-revolutionizing-natural-language-processing-a16be54ddb1e

Encoder Block

The Encoder's main job is to understand the context of the input sentence.

INPUT:
The list of number-vectors (embeddings) for each token. [Embedding for "I"] [Embedding for "am"] [Embedding for "a"]...

PROCESS:
We'll cover these in the next few slides, but the Encoder uses:

- Positional Encoding (Tells the model where each word is)
- Multi-Head Self-Attention
- A Feed-Forward Network

OUTPUT:
A new list of number-vectors. These are "context-rich" representations.

The original vector for "transformer" just meant "transformer." The new vector for "transformer" now also understands it's related to "I" and "am" and "a" in this specific sentence.

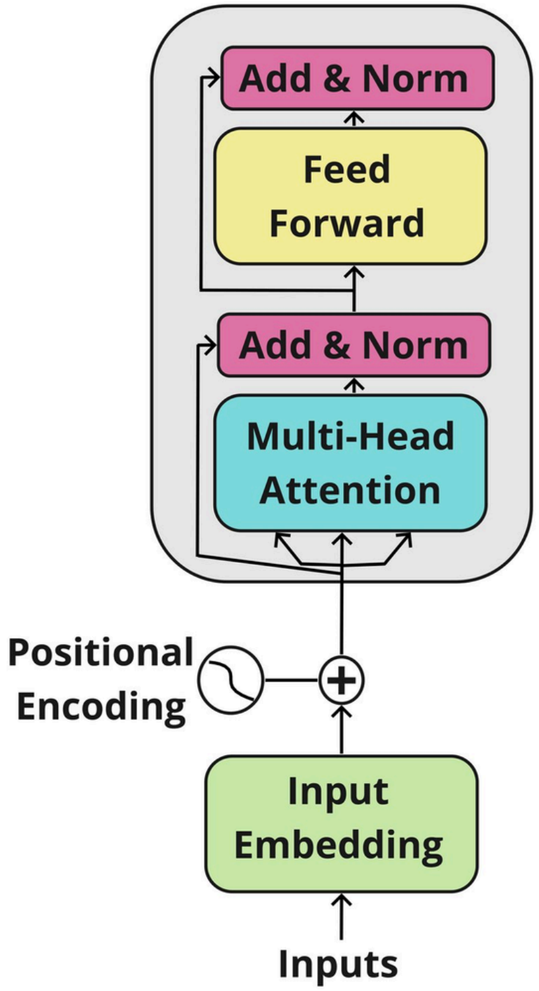


Image credits:https://www.researchgate.net/figure/Detailed-view-of-a-transformer-encoder-block-It-first-passes-the-input-through-an_fig1_352239001

Decoder block

Core Job

- Decoder: Uses that understanding to generate the output.

How it Works (One Word at a Time)

1. INPUTS: Tokenised and Positionally embedded
2. PROCESS:
    - Masked Self-Attention: each position attends only to earlier tokens (no look-ahead).
    - Feed Forward Neural Network: "Thinks" about the information gathered from the self-attention step.
3. OUTPUT:
    - Probabilities for every possible next word in the dictionary.
    - The model picks the one with the highest score (e.g., "un"), which becomes part of the input for the next loop.
    - This process repeats, generating one token at a time until it produces an <EOS> (End of Sentence) token or hits its max length.

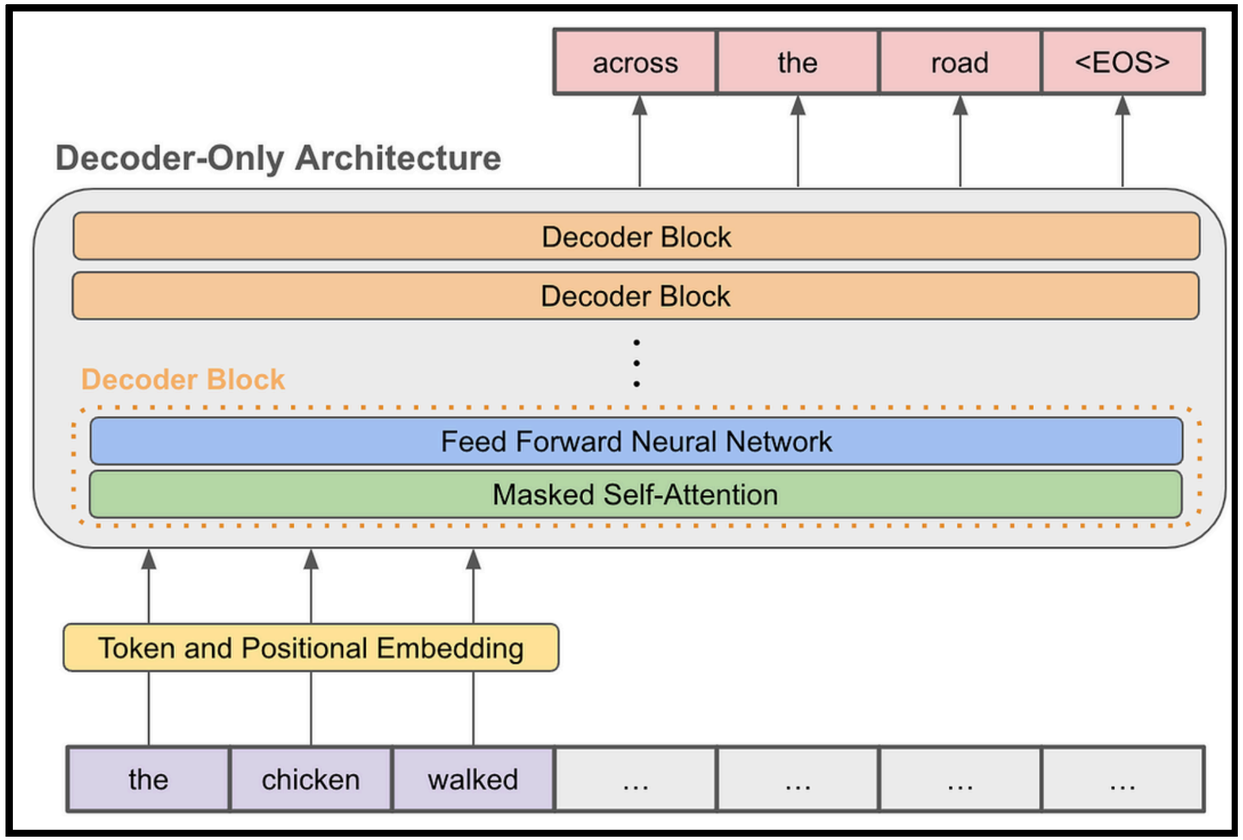


Image credits:https://www.researchgate.net/figure/Detailed-view-of-a-transformer-encoder-block-It-first-passes-the-input-through-an_fig1_352239001

Attention Mechanism

What is Attention?

Think: "Which words should this token care about?"

- Improves text generation by focusing on relevant context
- Captures relationships between tokens

Self Attention mechanism

The Job:
Allows a word to get its meaning by looking at all other words in the same sentence.

Example: "The cat chased the mouse until it got tired"

- Self-Attention links "it" to "cat".

The Result: The model learns which surrounding words are relevant (and updates the vector for "it" to represent the correct meaning.

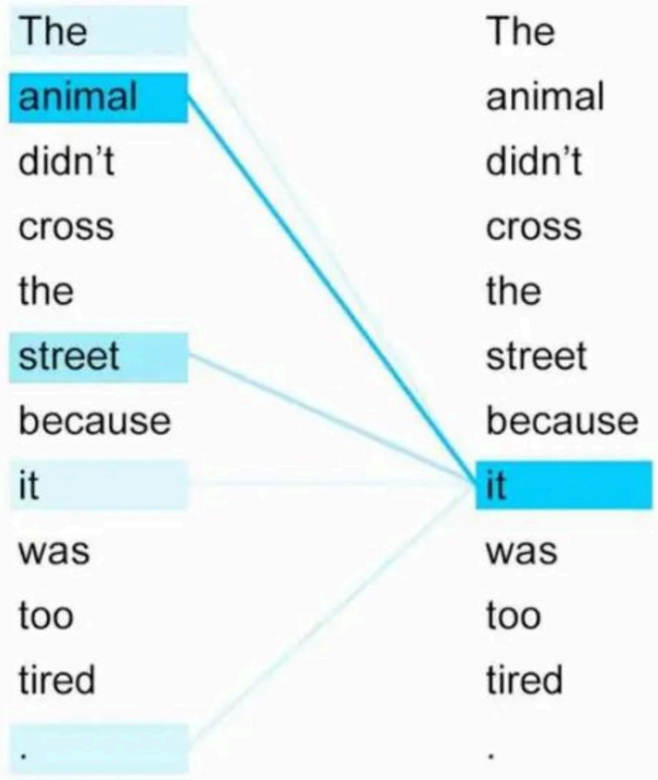


Image credits: https://spotintelligence.com/2024/02/29/bert-nlp/

Self Attention mechanism

Query(Q): The word we're focusing on (e.g., "it") forms a "Query."

Key(K): Every word in the sentence has a "Key"

Values(V): Actual content to be aggregated or weighted

Attention Scores: Raw relevance numbers from dot-products of Queries with Keys.

- (Q "it") -> (K "cat") = 9.5/10 (High Score)
- (Q "it") -> (K "mouse") = 1.5/10 (Low Score)

Attention Weights: Scores are converted into probabilities (using softmax) that add up to 1.

- "cat" -> 0.9 (Gets almost all the attention)
- "mouse" -> 0.1

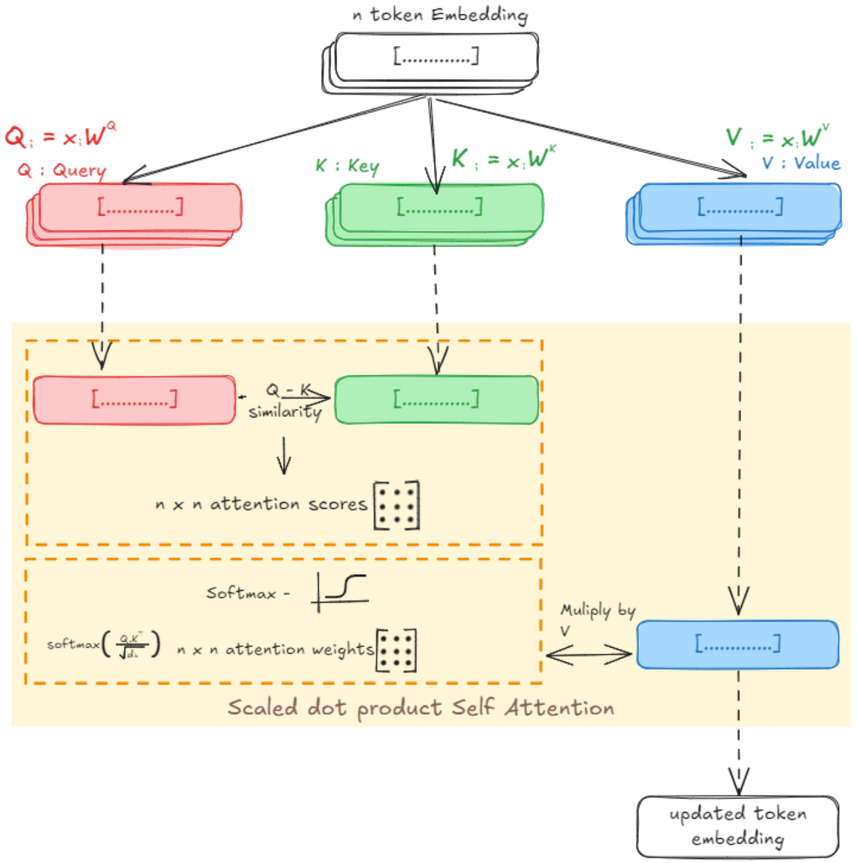


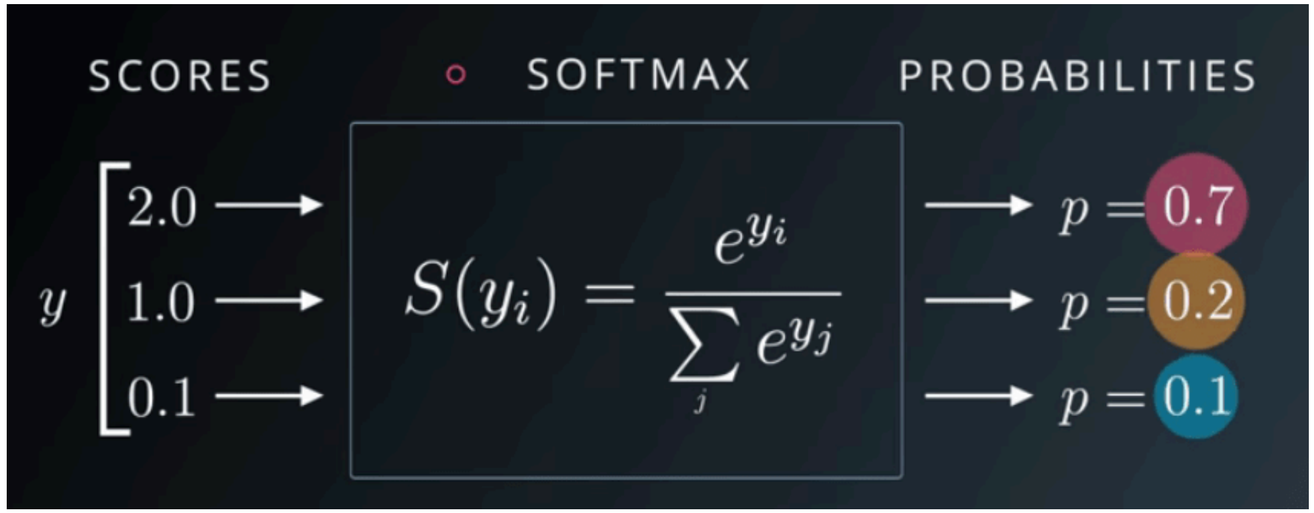
Image credits: https://medium.com/@anjilakshetri/build-your-own-transformer-a-complete-step-by-step-implementation-guide-4680443df83b

softmax

Softmax converts raw "Attention Scores" into "Attention Weights" (

Example:

- Raw Scores:
  - “cat”: -0.105
  - “mouse”: -2.659
  - “chased”: -3.507
- After Softmax (Weights):
  - "cat": 0.9
  - "mouse": 0.07
  - "chased": 0.03



Why? It forces the model to make a clear choice, telling it to pay almost all of its attention to "cat."

Image credits: https://medium.com/@shashank.1842000/the-softmax-activation-function-4a184310b3b4

Multi-Head Attention Mechanism

Instead of one, the model uses multiple "heads" in parallel.

Each head looking for different patterns:

- Head 1: Finds grammatical links (e.g., subject-verb).
- Head 2: Finds pronoun links (e.g., "it" -> "cat").
- Head 3: Finds action links (e.g., "chased" -> "mouse").

Result: Combining these different perspectives gives the model a much richer, deeper understanding of the sentence.

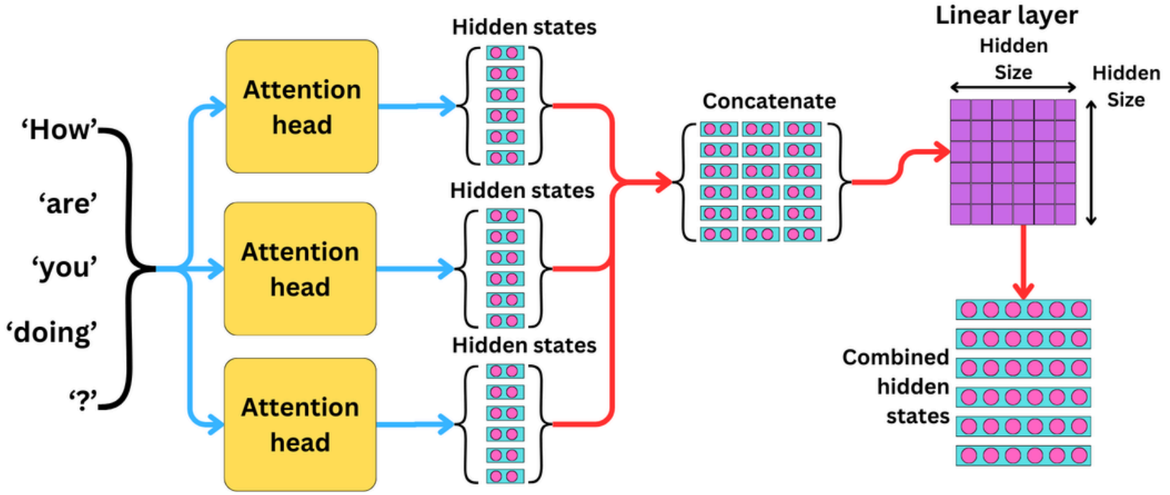


Image credits:https://newsletter.theaiedge.io/p/the-multi-head-attention-mechanism

Position-wise Feed-Forward Networks (FFN)

Core Idea:

- Multi-Head Attention is the "Relational" step: It mixes information between all tokens to build context.
- Feed-Forward Network (FFN) is the "Processing" step: It processes each token independently to add computational depth.

Why?: The FFN transforms the context-rich vectors from the attention step, allowing the model to derive more complex and abstract features from that information.

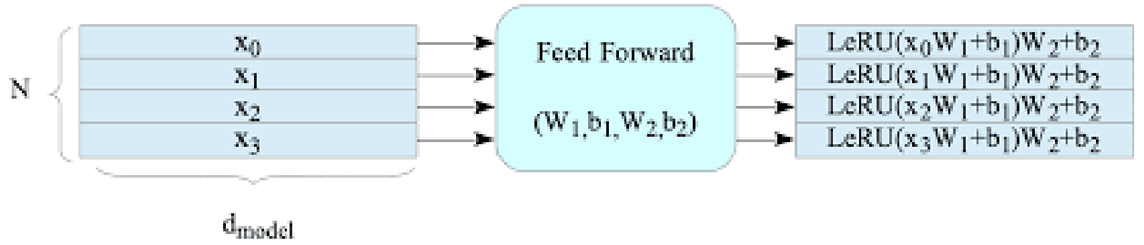


Image credits: https://www.interdb.jp/dl/part04/ch15/sec04.html

Position-wise Feed-Forward Networks (FFN)

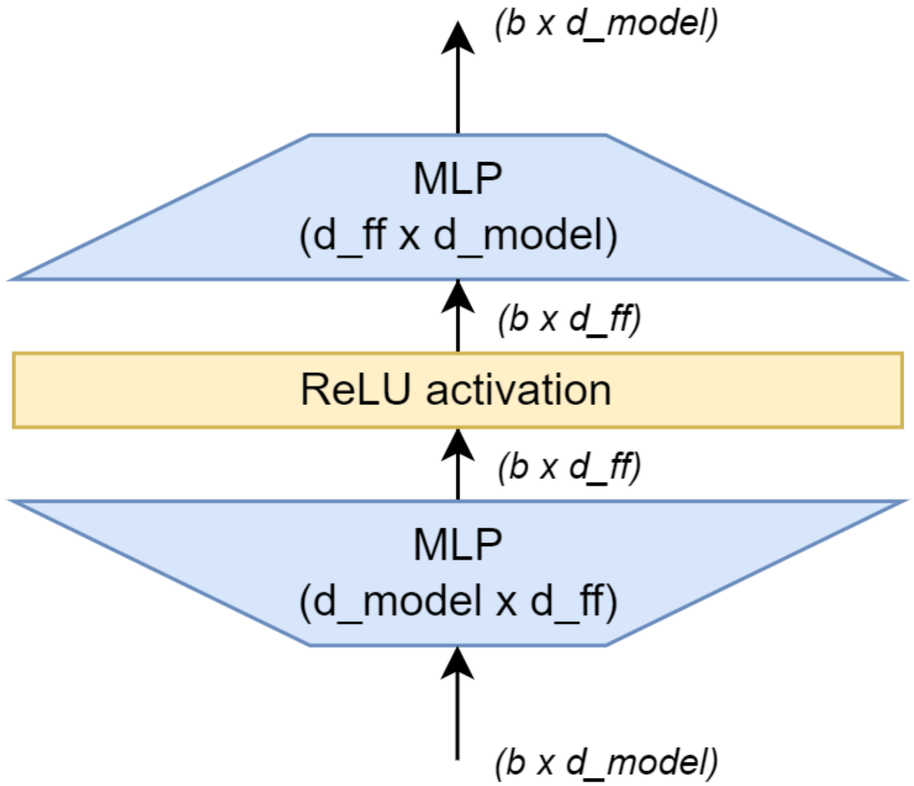
The Feed-Forward Network (FFN) is a two-layer neural network applied "position-wise."

"Position-wise" Application

- Identical, shared network which is applied independently to every token's vector representation.

Internal Transformation:
This sub-layer transforms each vector to learn complex, non-linear patterns:

1. Expand (Layer 1): Projects the vector to a much larger dimension (e.g., 512 -> 2048).
2. Activate (ReLU): Applies a non-linear activation function.
3. Project (Layer 2): Projects the vector back to the original dimension (e.g., 2048 -> 512).

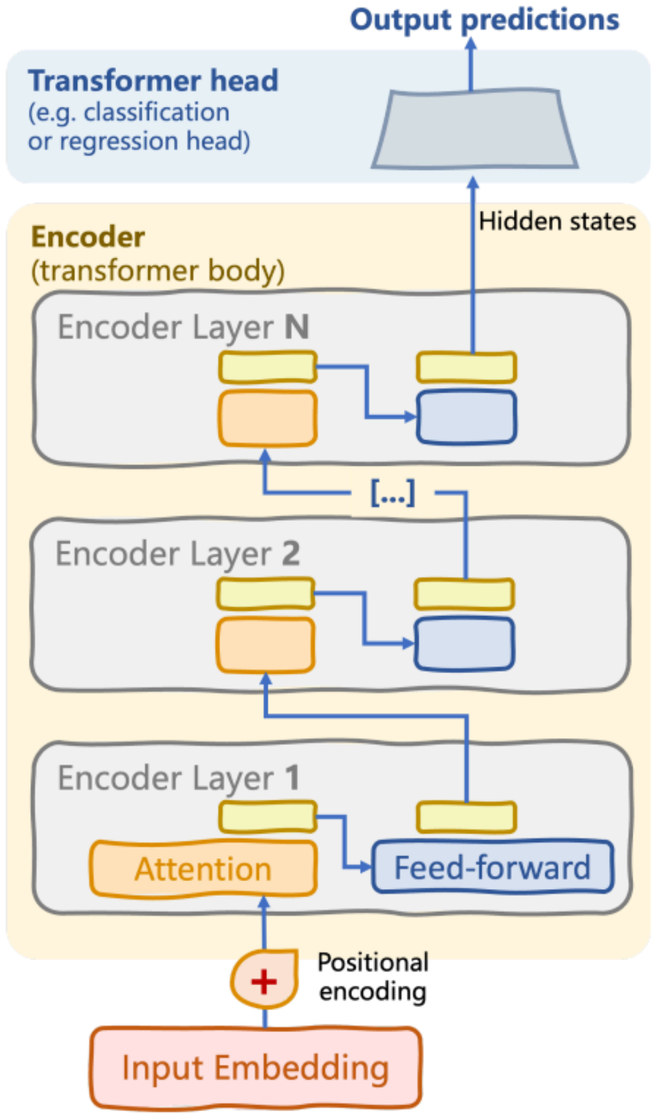


Add & Norm

https://molgorithm.medium.com/what-is-add-norm-as-soon-as-possible-178fc0836381

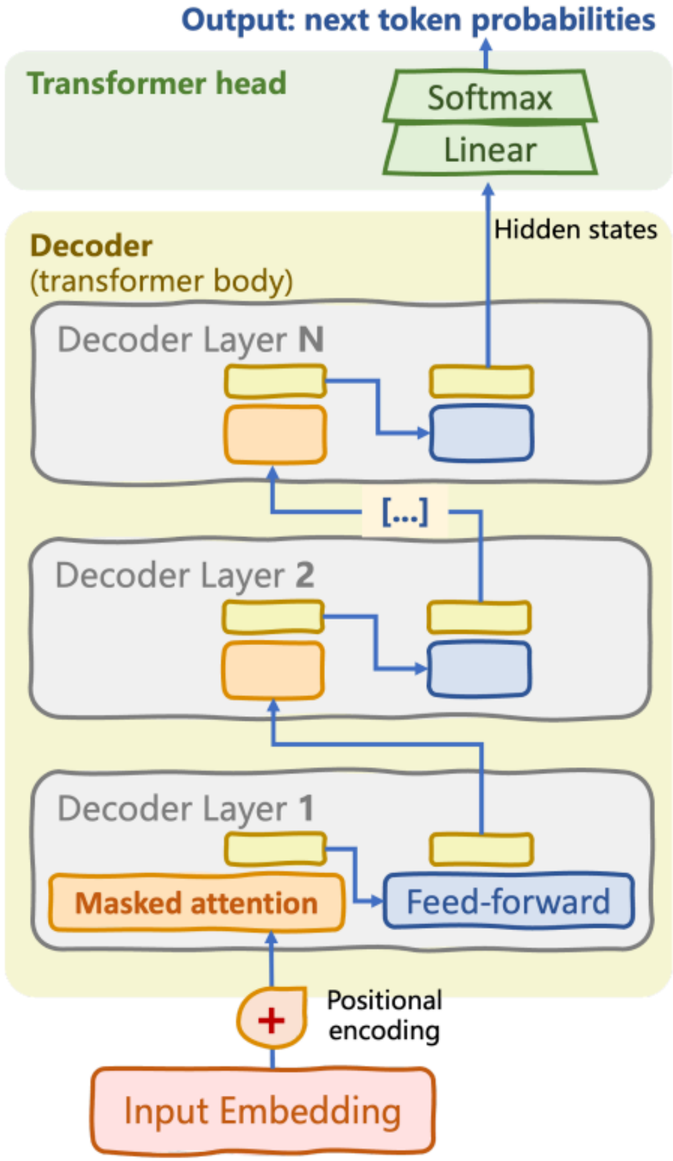
Encoder only

- Best for: Understanding tasks (e.g., sentiment analysis, text classification, question answering).
- Examples: BERT, ALBERT, DistilBERT



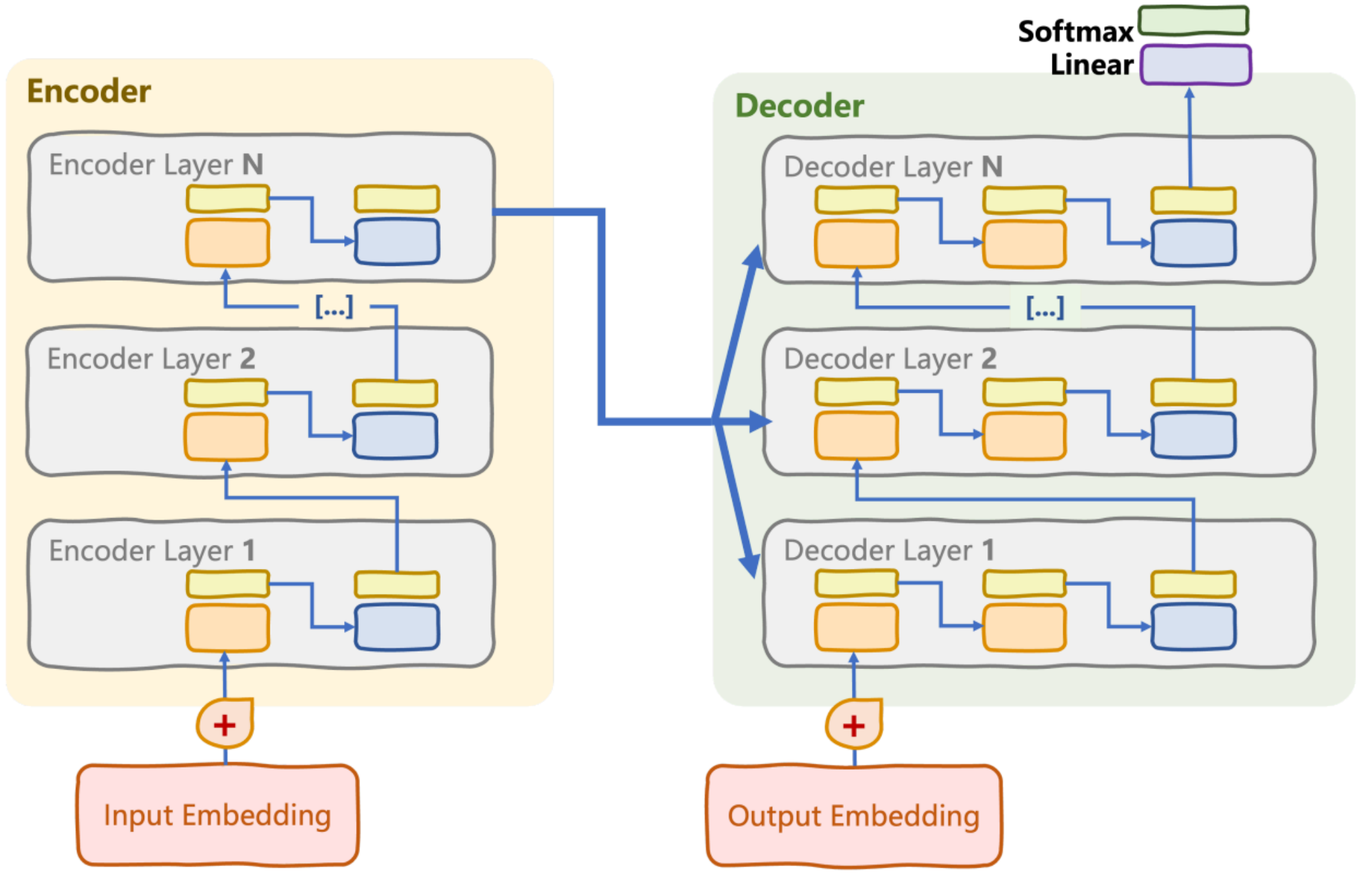
Decoder only

- Best for: Generative tasks (e.g., writing essays, chatbots, code generation).
- Examples: ChatGPT, Claude



Encoder-Decoder Transformer

- Best for: Transformation tasks (e.g., translation, summarization).
- Other Examples: BART, mBART, "Vanilla" Transformer, T5.



Parameters in a Transformer

- d_model (Model Dimension):
  - This is the "size" of the vector used for each token.
- nheads (Number of Heads):
  - How many parallel "attention heads" to use in the Multi-Head Attention step.
  - This number must divide d_model evenly (e.g., if d_model=512, nheads=8 is a perfect fit).
- num_encoder_layers:
  - Number of Encoder Blocks to stack on top of each other.
  - More layers = a "deeper" model that can capture richer, more abstract context (but is slower).
- num_decoder_layers:
  - Number of Decoder Blocks to stack. This is often the same as the number of encoder layers.

```
#Example 1
import torch.nn as nn
model = nn.Transformer(d_model=512,
                       nhead=8,
                       num_encoder_layers=6,
                       num_decoder_layers=6)
print(model)
```

MATERIALS ARE ALSO ON GITHUB

https://github.com/NUS-SDS-Workshops/AY-25-26-Public

THIS SEMESTER

SDS WORKSHOPS COMPLETED

OUR 3-PART BUILD-UP BEGINNER SERIES

STAY TUNED FOR EVEN MORE WORKSHOPS NEXT SEMESTER!

@NUSSDS

NUS.SDS

FEEDBACK FORM

FILL IN SO WE CAN IMPROVE FUTURE WORKSHOPS FOR YOU!

THANK YOU

AND WE HOPE TO SEE YOU AGAIN NEXT SEMESTER!