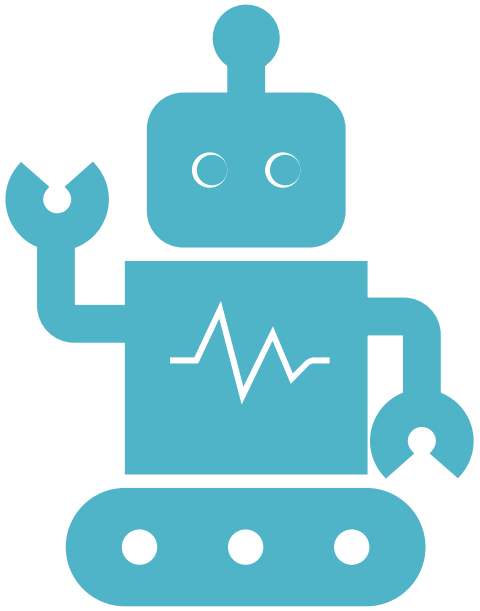
Generative AI

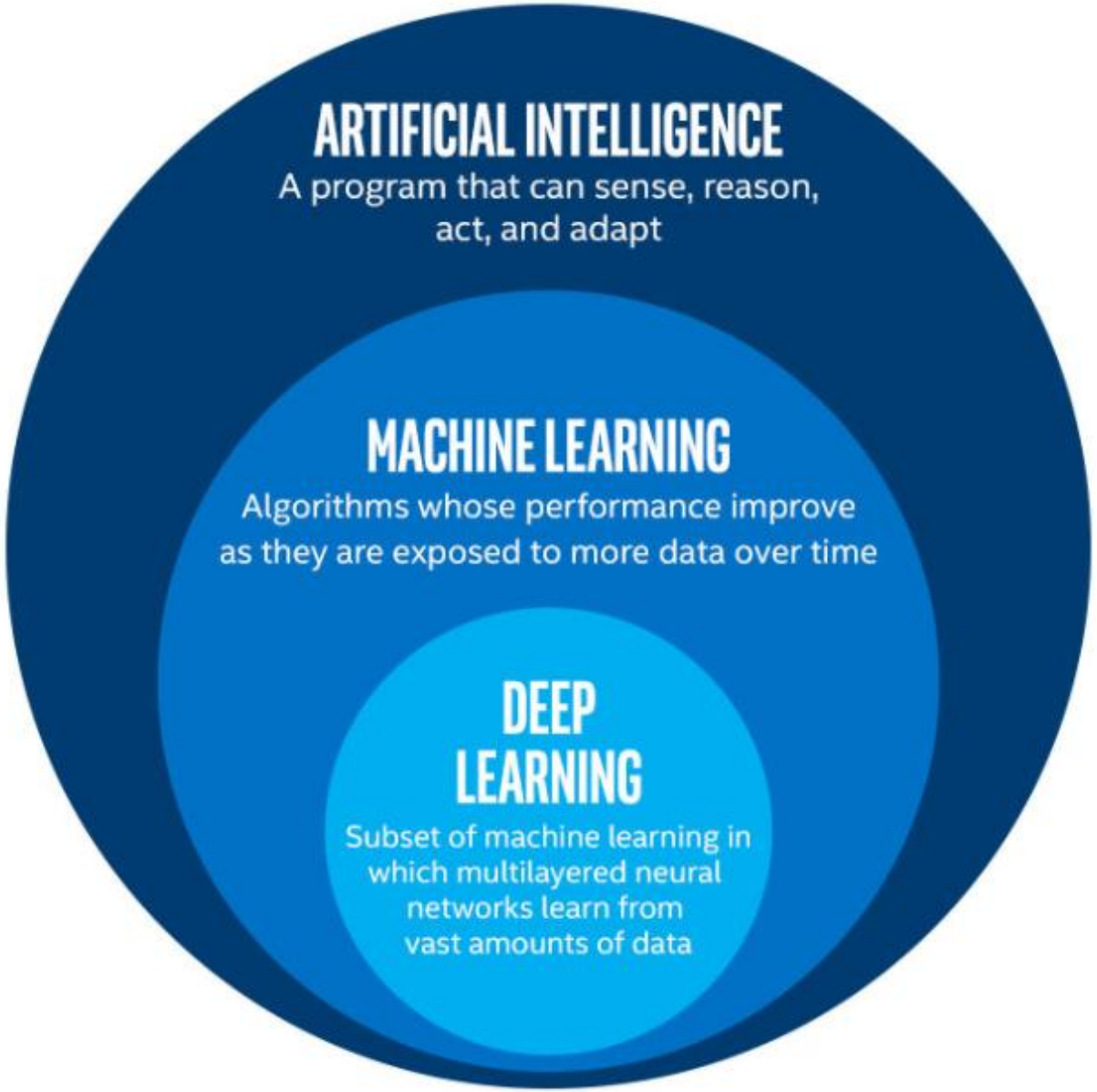
Dr. Abdulkarim M. Jamal Kanaan

Artificial Intelligence



"Artificial intelligence (AI) is the science and engineering of making intelligent machines, especially intelligent computer programs."

[John McCarthy, father of AI]



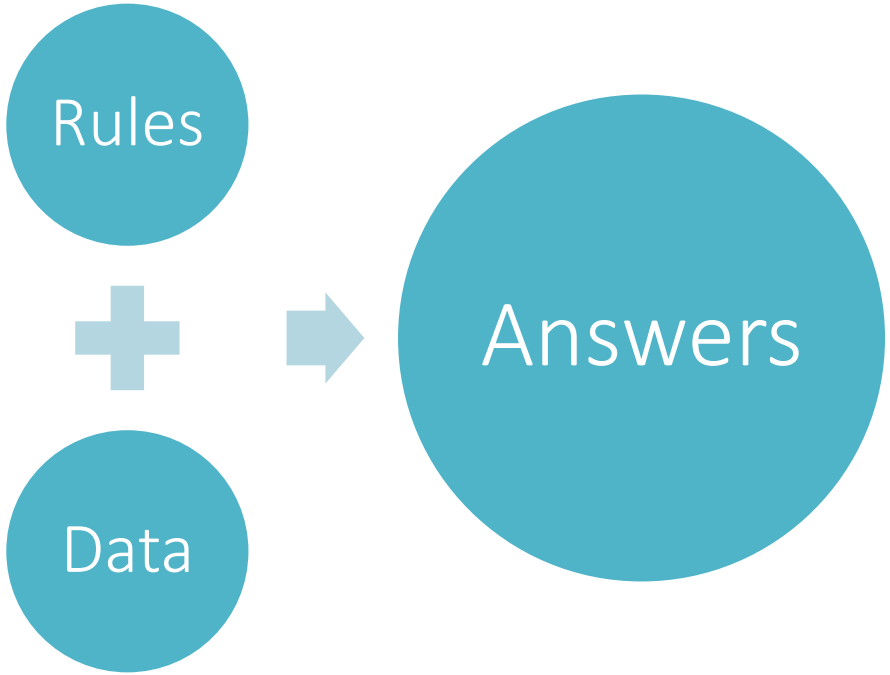
Artificial Intelligence

"Artificial intelligence (AI) is the science and engineering of making intelligent machines, especially intelligent computer programs."

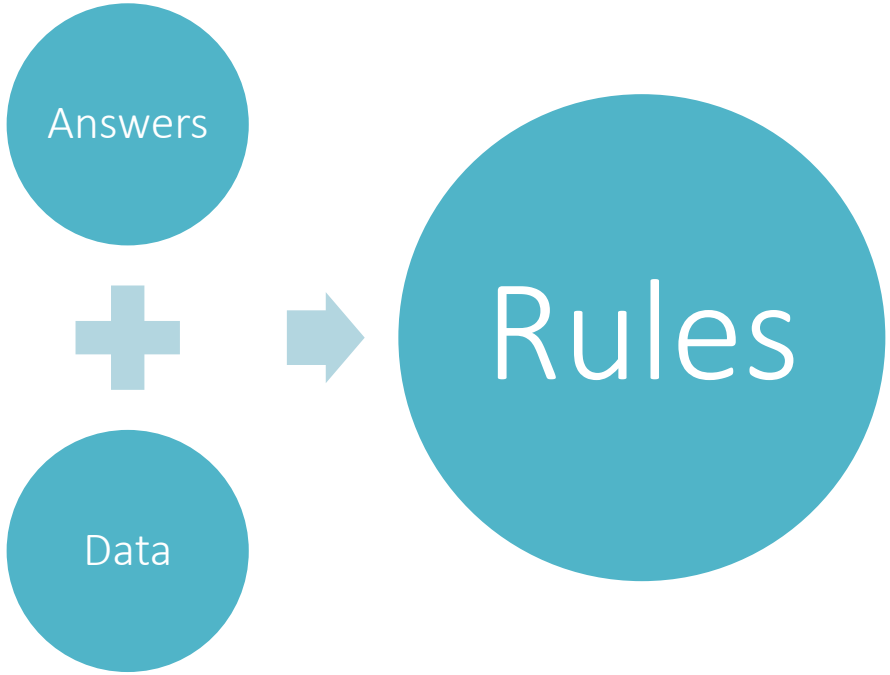
[John McCarthy, father of AI]

Management Information System vs Data Mining

TRADITIONAL PROGRAMMING



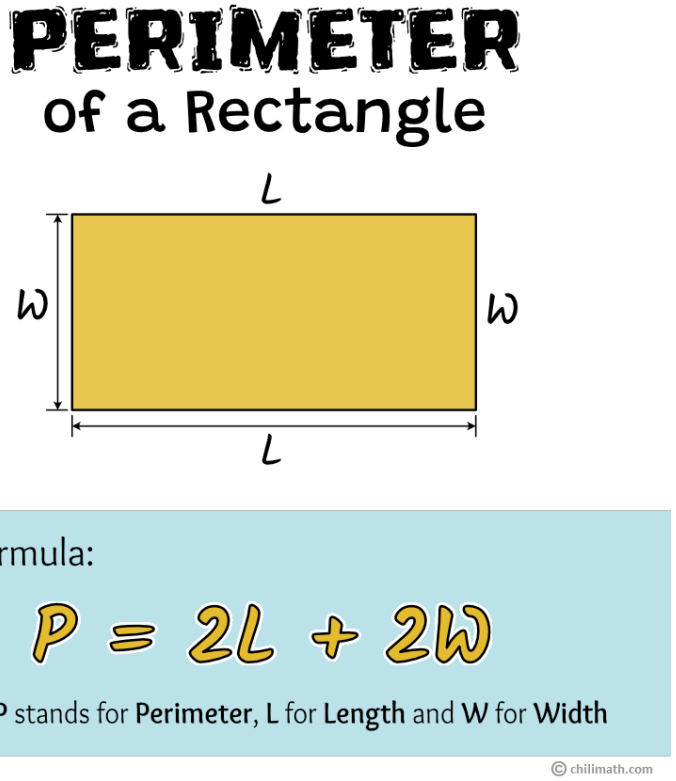
DATA MINING



Rectangle Perimeter

Management Information System (MIS)

Data Mining



Rectangle Perimeter

MANAGEMENT INFORMATION SYSTEM

Data:

Width=[1, 2, 3, 4, 5]

Height=[6, 7, 8, 9, 10]

Rule:

Area → (Width + Height) * 2

Answers:

Area: [14, 18, 22, 26, 30]

DATA MINING

Data:

Width=[1, 2, 3, 4, 5]

Height=[6, 7, 8, 9, 10]

Answers:

Area: [14, 18, 22, 26, 30]

Rule:

Area → (Width + Height) * 2

```
if (speed < 3) {
   state=WALKING
}
```

```
if (speed < 3) {
   state=WALKING
} else {
   sate=RUNNING
}
```

```
if (speed < 3) {
   state=WALKING
} else if (speed < 5) {
   sate=WALKING
} else {
   state=BIKING
}
```

```
// Oh!!!
```

Activity Recognition

010101111001110
001110001101001
011100011011011

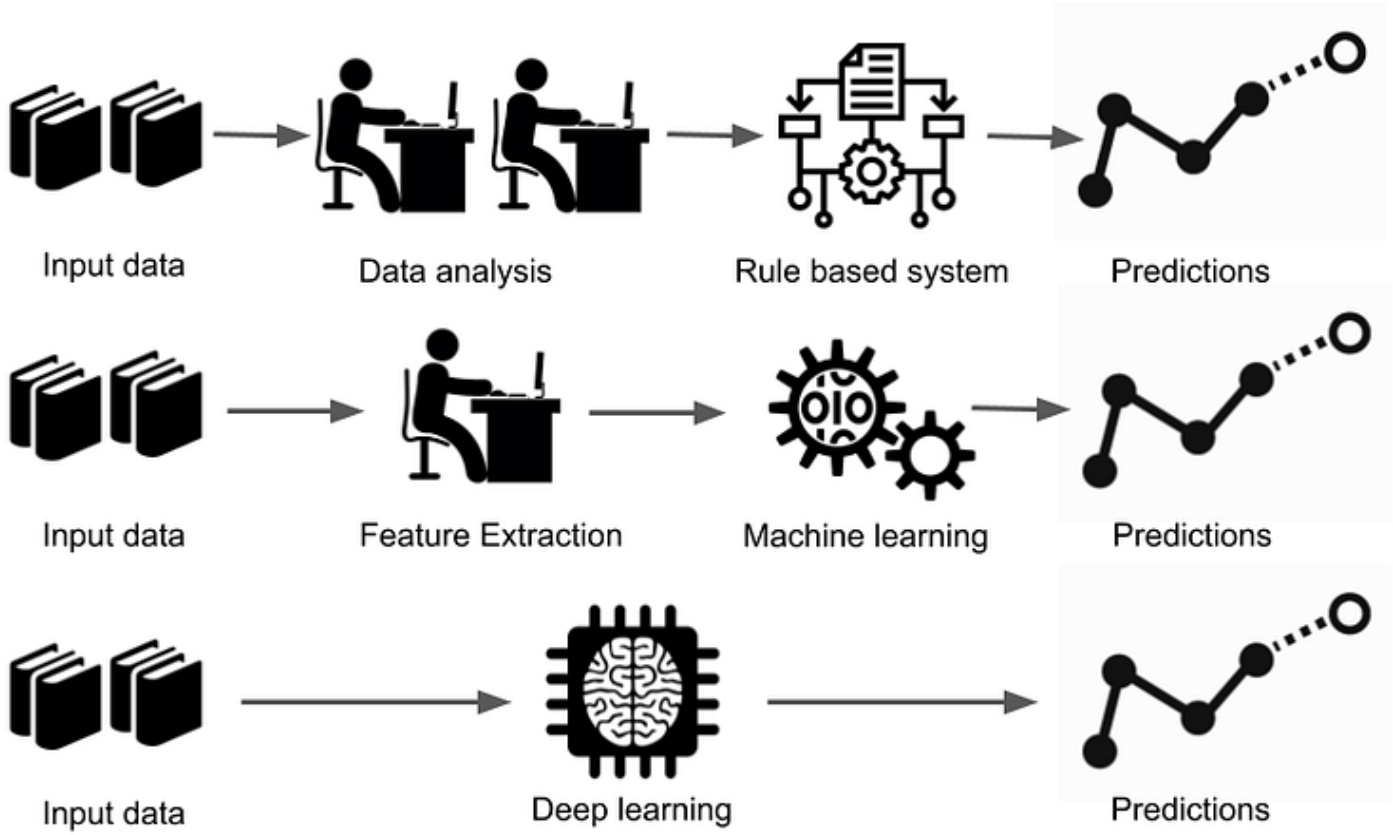
010101111001110
001110001101001
011100011011011

0101011110011100011000110101001011100011011011

010101111001110
001110001101001
011100011011011

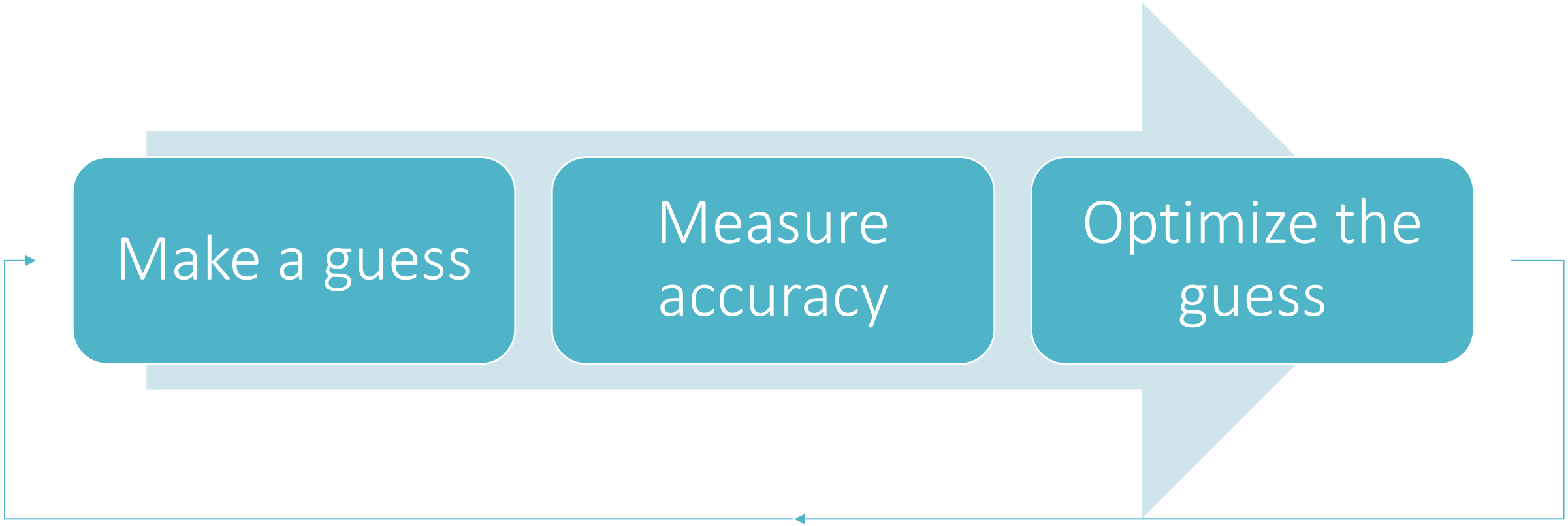
Activity Recognition

Evolution of AI Technique



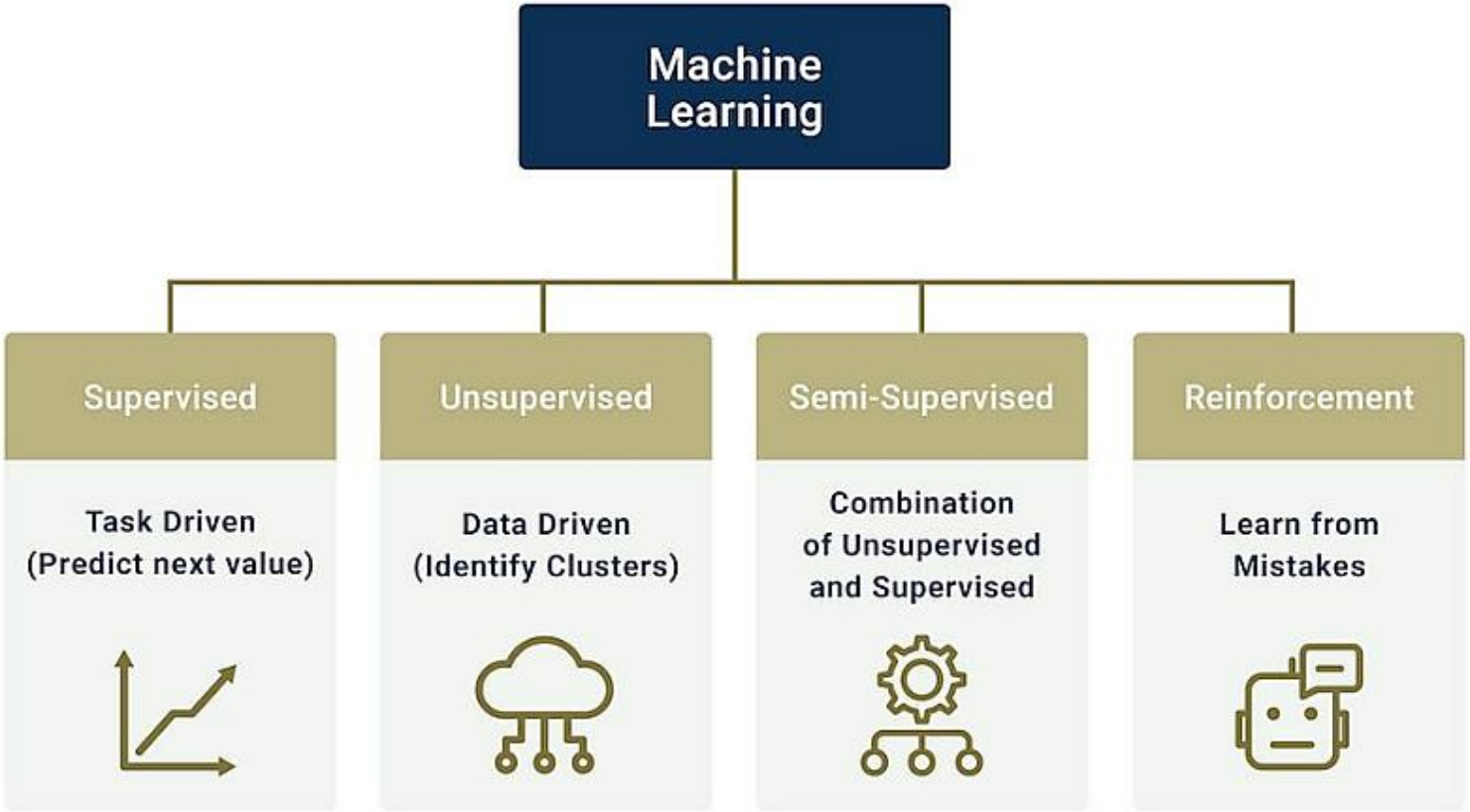
https://lotuslabs.medium.com/accurate-insurance-claims-prediction-with-deep-learning-23c575b90dd

Machine Learning Paradigm



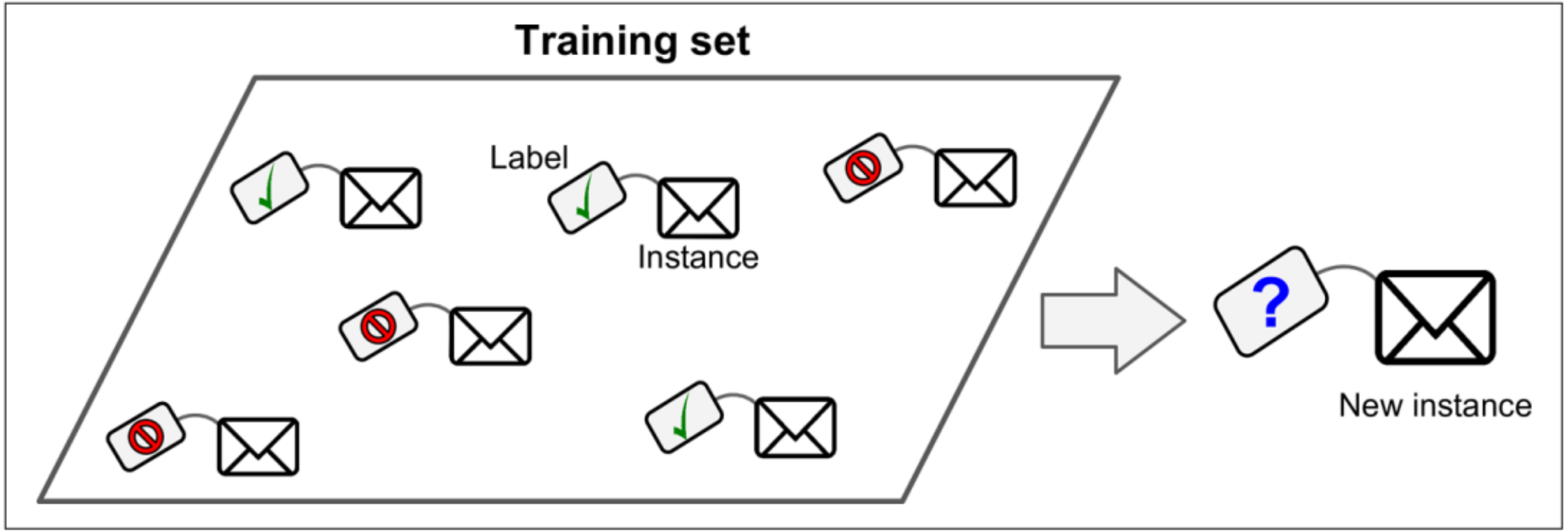
Machine Learning Paradigm

Types of Machine Learning Algorithms



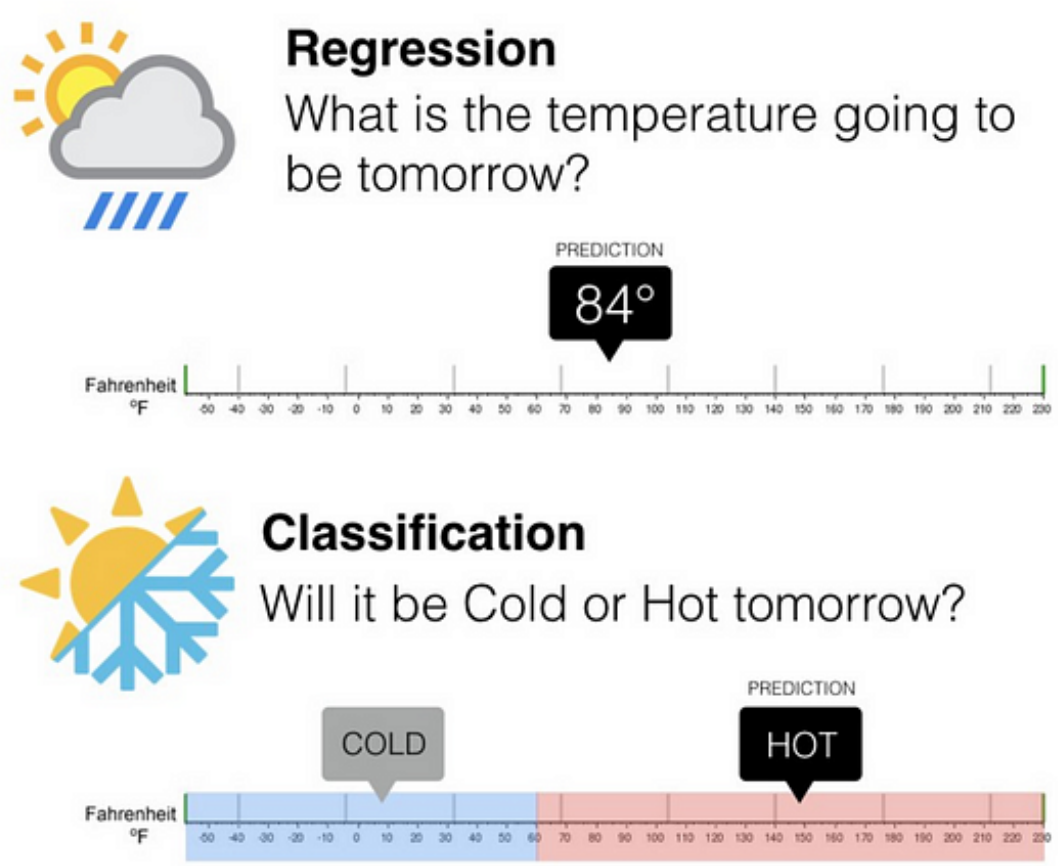
https://medium.com/@coderacheal/machine-learning-for-absolute-beginners-69ce9bb08b46

Supervised learning



Géron, A. (2019)

Supervised Learning



enjoy algorithms

The Idea of **Classification** and **Regression**

Karan, R. (2022, February 2). *Differences Between Supervised and Unsupervised Learning - Shiksha Online*. Shiksha.com; Shiksha Online. https://www.shiksha.com/online-courses/articles/differences-between-supervised-and-unsupervised-learning/

Supervised Learning - Classification

Drug Classification

- ✓ Input: numeric and categorical features
- ✓ Output: 0 (Drug A), 1 (Drug B) – discrete (nominal)
- ✓ Features: age, sex, bp, cholestero

Patient ID	Age	Sex	BP	Cholesterol	Drug
p1	Young	F	High	Normal	Drug A
p2	Young	F	High	High	Drug A
p3	Middle-age	F	Hiigh	Normal	Drug B
p4	Senior	F	Normal	Normal	Drug B
p5	Senior	M	Low	Normal	Drug B
p6	Senior	M	Low	High	Drug A
p7	Middle-age	M	Low	High	Drug B
p8	Young	F	Normal	Normal	Drug A
p9	Young	M	Low	Normal	Drug B
p10	Senior	M	Normal	Normal	Drug B
p11	Young	M	Normal	High	Drug B
p12	Middle-age	F	Normal	High	Drug B
p13	Middle-age	M	High	Normal	Drug B
p14	Senior	F	Normal	High	Drug A
p15	Middle-age	F	Low	Normal	?

Clothes Recognition

Clothes Recognition

Clothes Recognition

MANAGEMENT INFORMATION SYSTEM

DATA MINING SYSTEM

Clothes Recognition

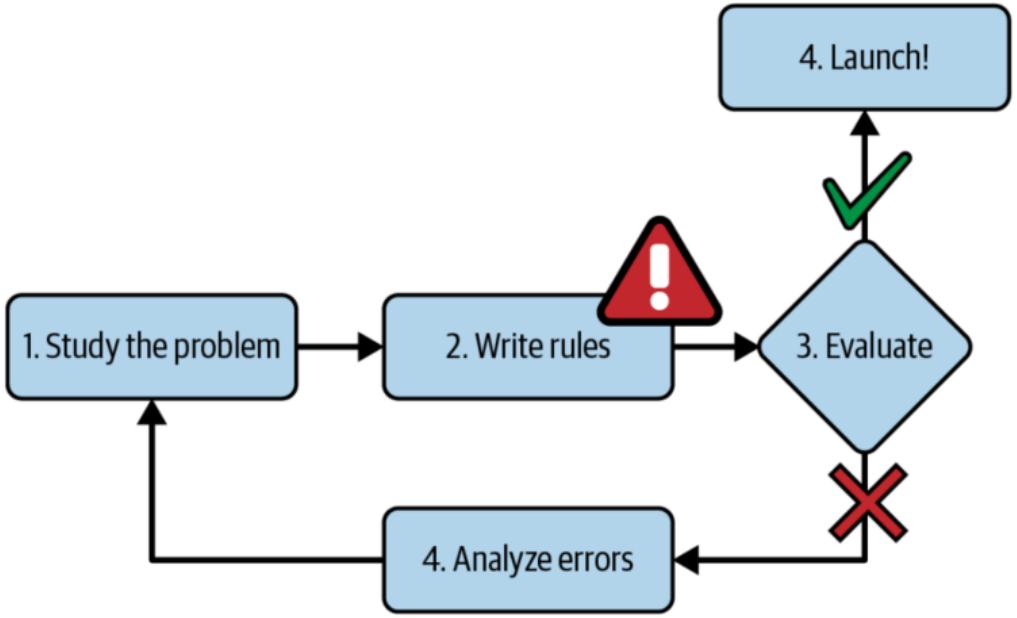
MANAGEMENT INFORMATION SYSTEM

DATA MINING SYSTEM

Role-Based Approach

Complexity

Static

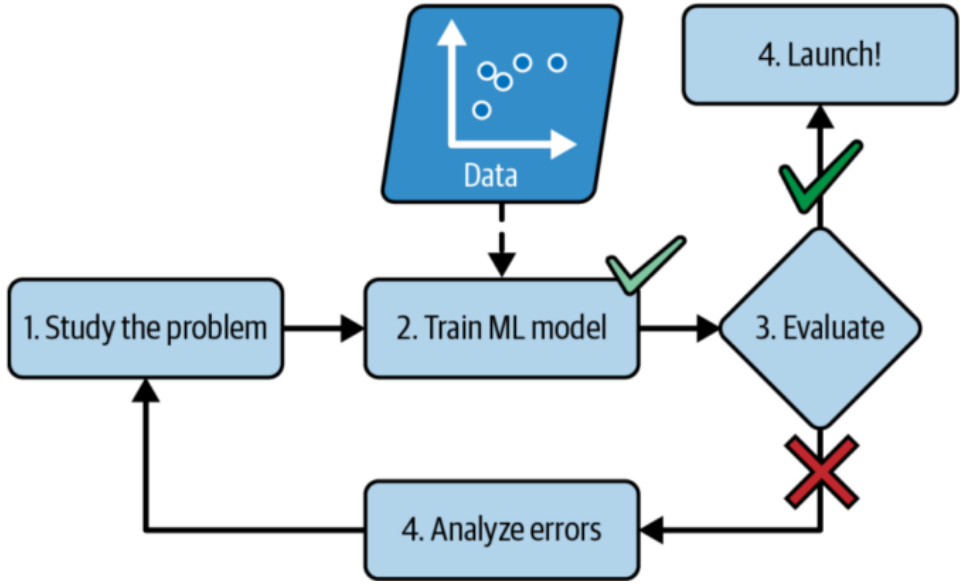


Géron, A. (2022)

Machine Learning Approach

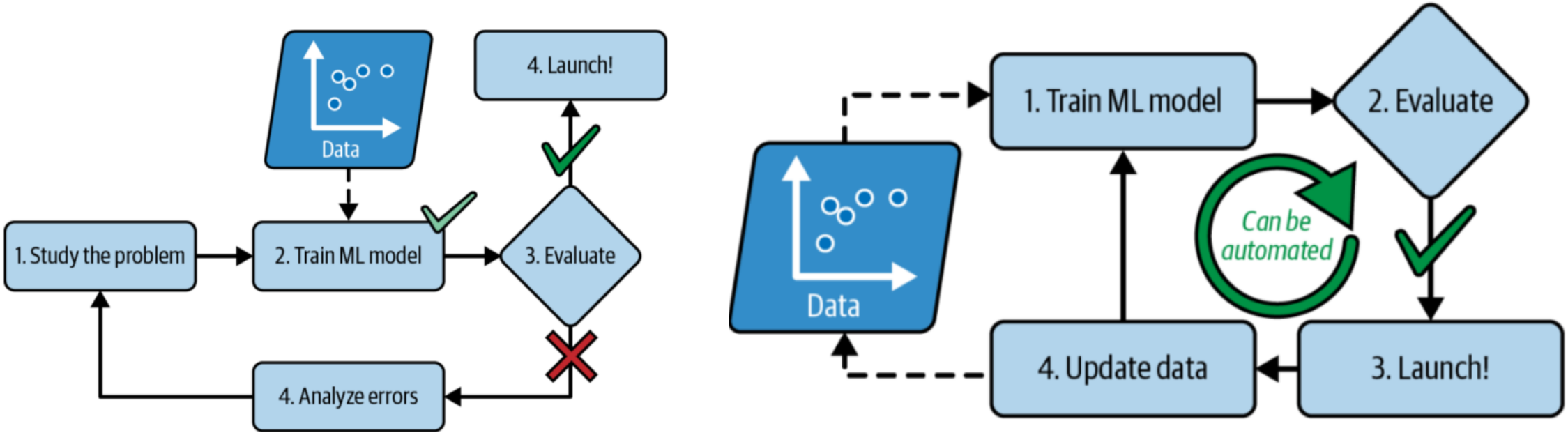
Automatically Learns

Easier to Maintain



The Machine Learning approach
Géron, A. (2022)

Machine Learning Approach

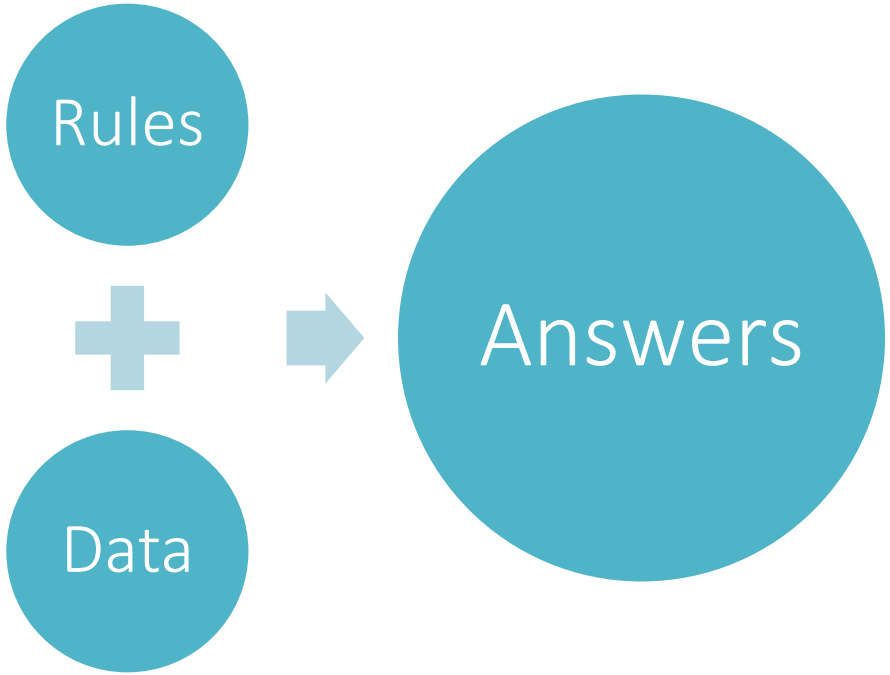


Géron, A. (2022)

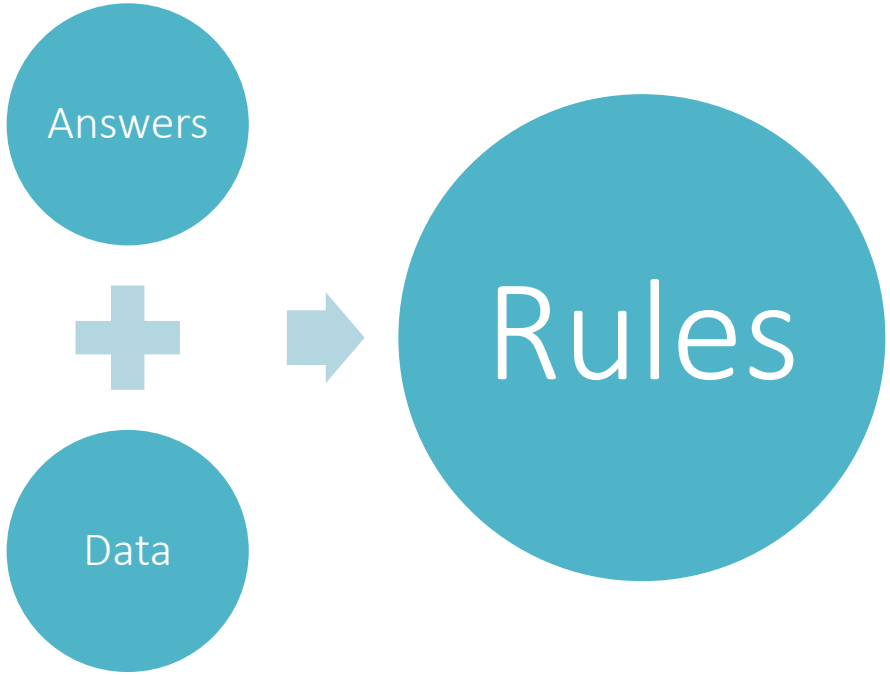
Artificial Neural Network

Management Information System vs Data Mining

MANAGEMENT INFORMATION SYSTEM



DATA MINING



Data Mining

Data (X)	:	[-1, 0, 1, 2, 3, 4]
Answers (Y)	:	[-3, -1, 1, 3, 5, 7]
Rules	:	?

Data Mining

Data (X) : [-1, 0, 1, 2, 3, 4]

Answers (Y) : [-3, -1, 1, 3, 5, 7]

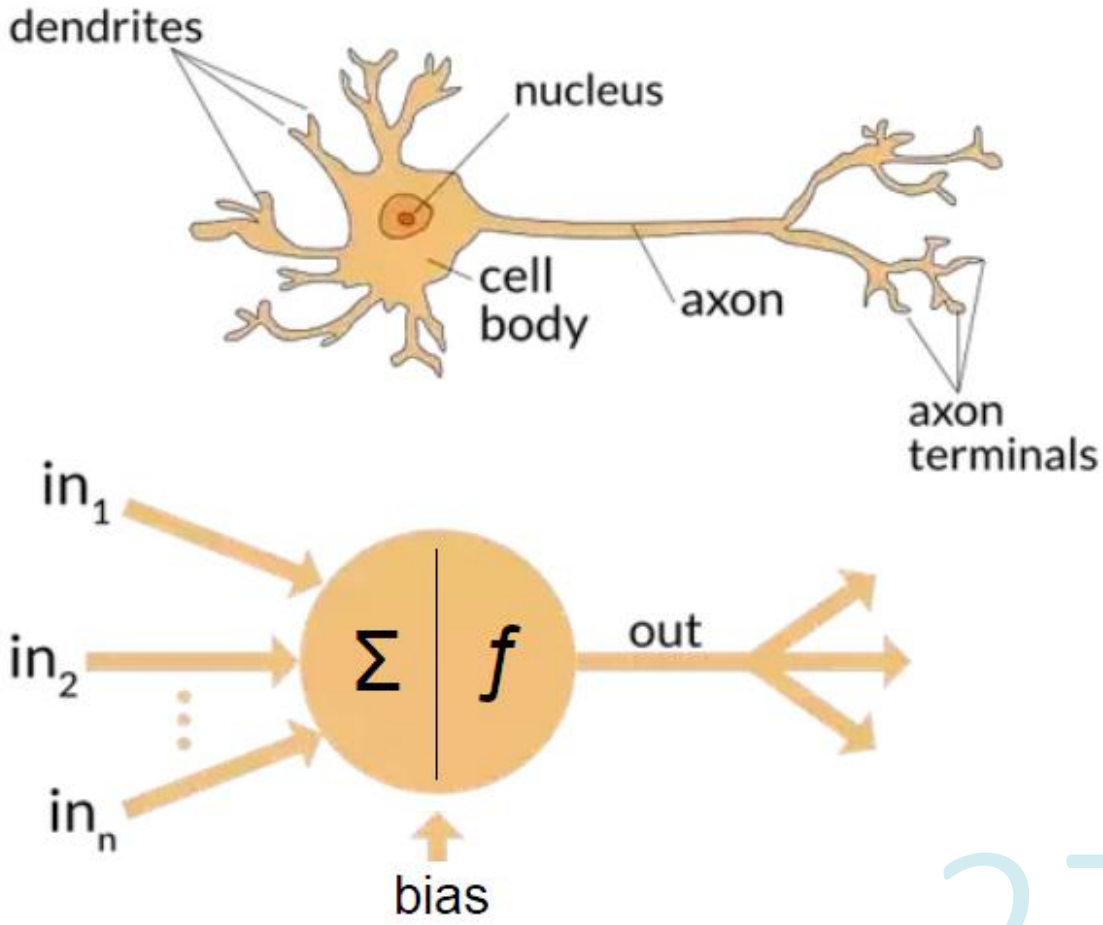
Rules : $Y = 2*X - 1$

Artificial Neural Networks (ANN)

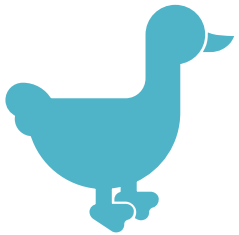
A neural network is a computational model inspired by the structure and functioning of biological neural networks in the human brain. It is a powerful machine learning algorithm capable of learning complex patterns and relationships in data.

Artificial Neural Network (ANN)

The concept of perceptron, which are precursors to artificial neuron, is founded on the possibility of emulating specific components of biological neurons, including dendrites, cell bodies, and axons.



Inventions



Birds

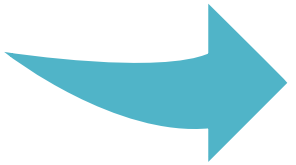
Honeycomb

Whale Fins

Inventions

airplanes

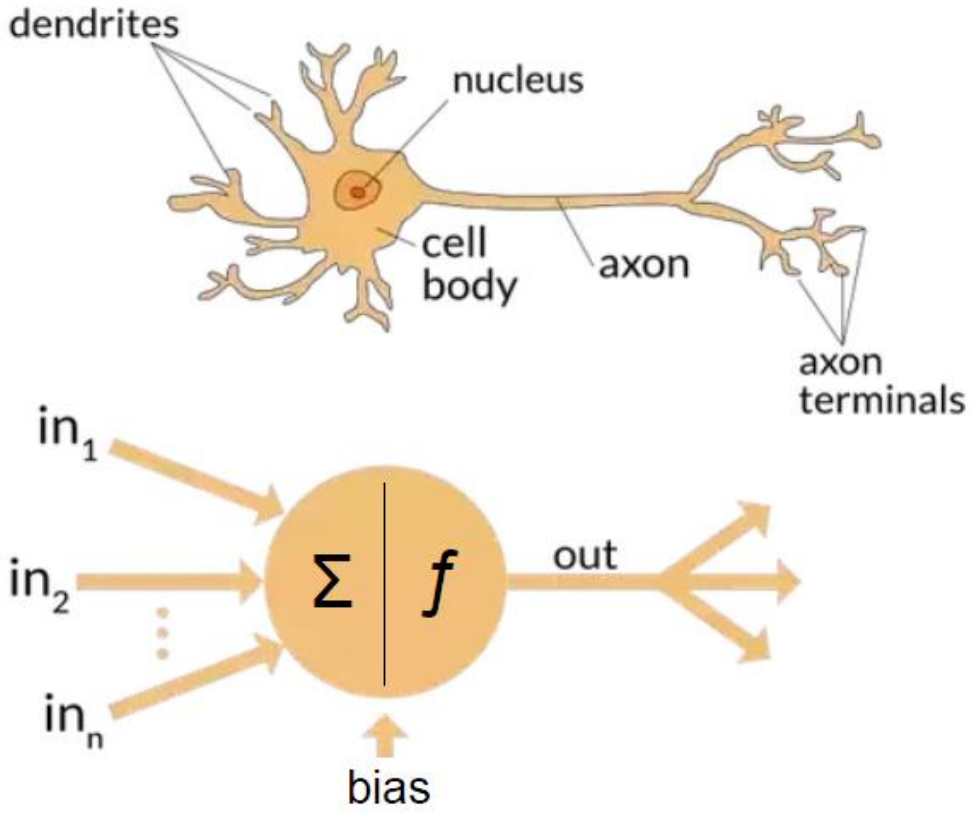
bridge

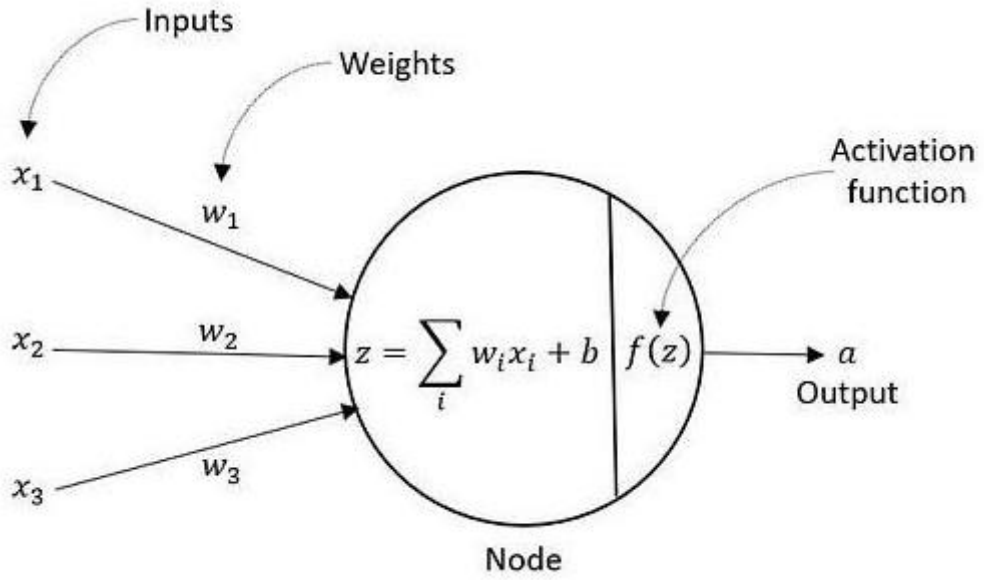


turbines

```
model = tf.keras.Sequential(
    [tf.keras.layers.Dense(units=1, input_shape=[1])]
)
model.compile(optimizer='sgd', loss='mean_squared_error')

data_x = np.array([-1.0,  0.0, 1.0, 2.0, 3.0, 4.0], dtype=float)
answers_y = np.array([-3.0, -1.0, 1.0, 3.0, 5.0, 7.0], dtype=float)

model.fit(data_x, answers_y, epochs=500)

model.predict([10.0])
```

The Perceptron





https://www.linkedin.com/pulse/neural-network-eeswar-chamarthi/

The Perceptron

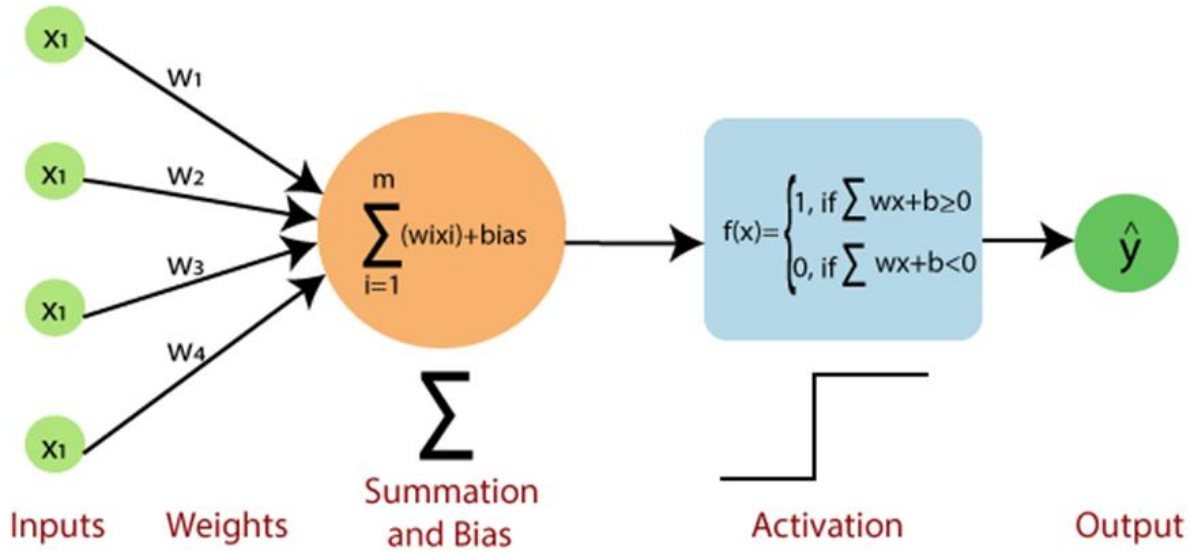
Inputs

Weights

Summation

Activation

Output



https://www.nomidl.com/deep-learning/difference-between-perceptron-and-neuron/

Activation Function

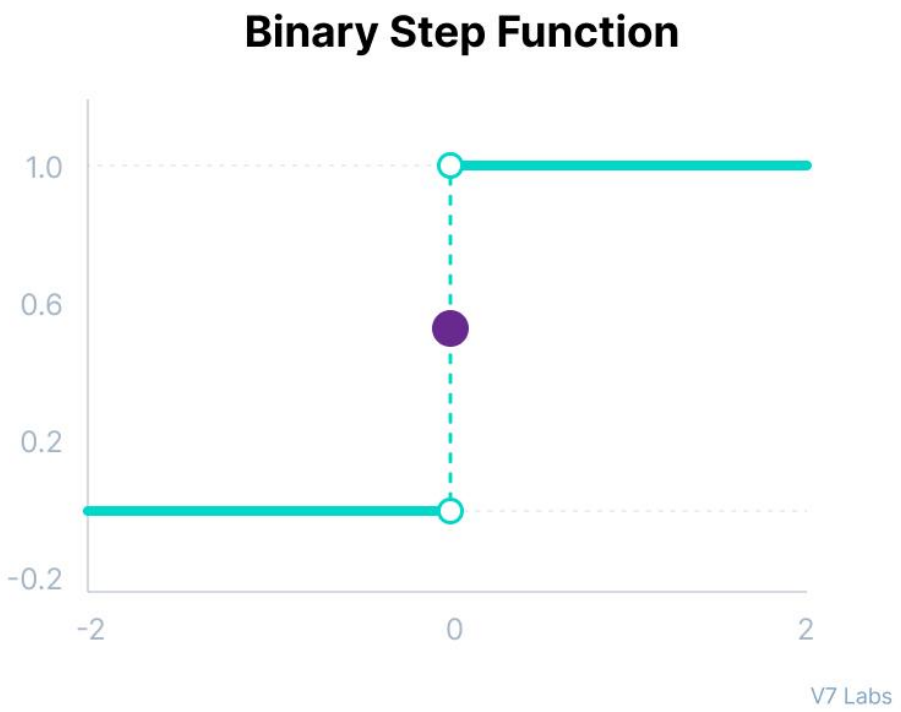
$$\text{heaviside}\,(z) = \begin{cases} 0 & \text{if } z < 0 \\ 1 & \text{if } z \geq 0 \end{cases}$$

$$\text{sgn}\,(z) = \begin{cases} -1 & \text{if } z < 0 \\ 0 & \text{if } z = 0 \\ +1 & \text{if } z > 0 \end{cases}$$

Binary Step Function

Binary step

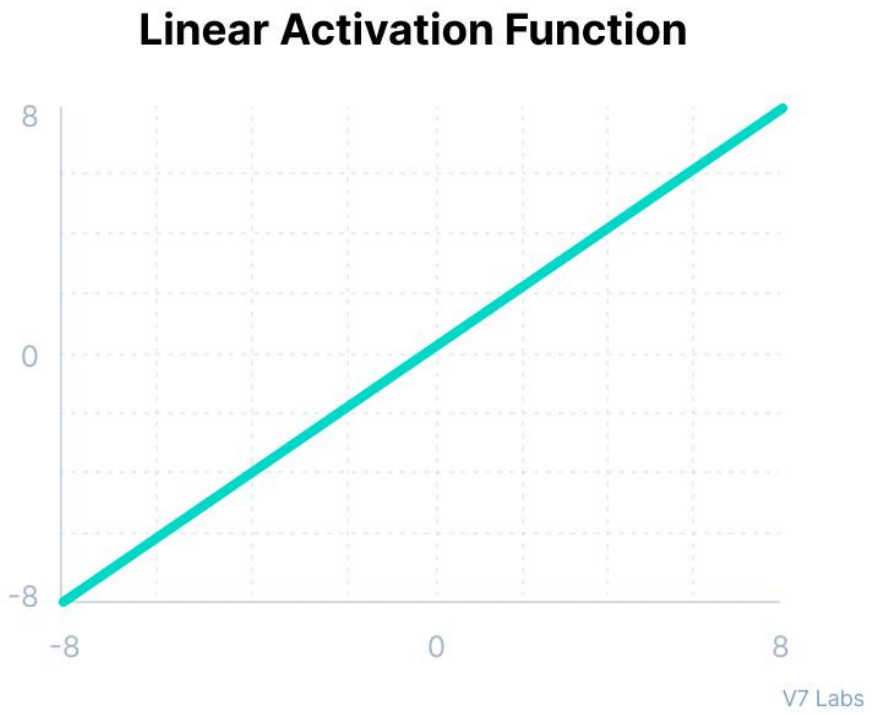
$$f(x) = \begin{cases} 0 & for\ x < 0 \\ 1 & for\ x \geqslant 0 \end{cases}$$



Linear Activation Function

$$Linear$$

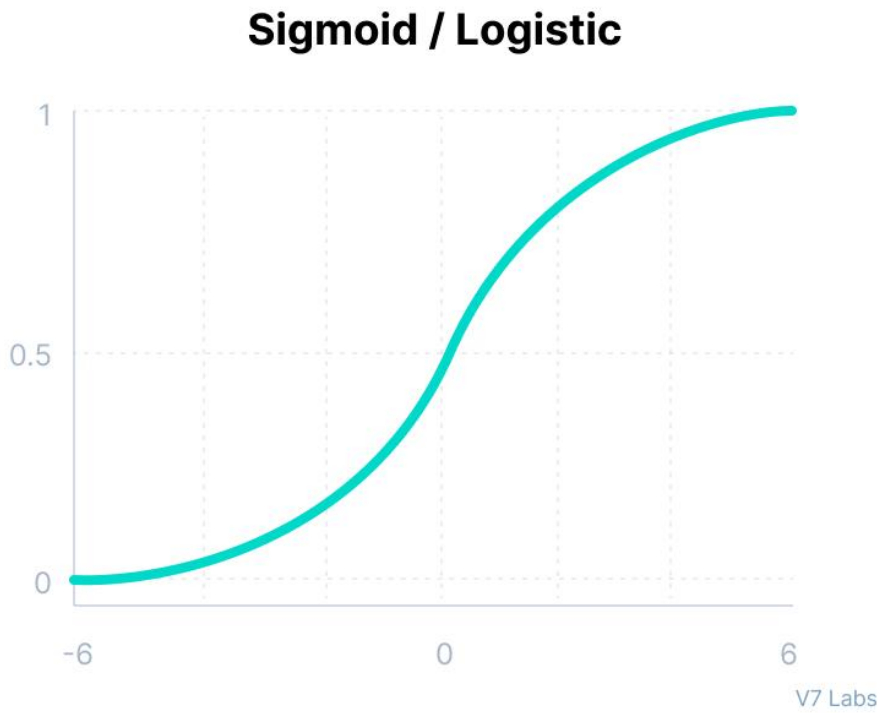
$$f(x) = x$$



Sigmoid / Logistic Activation Function

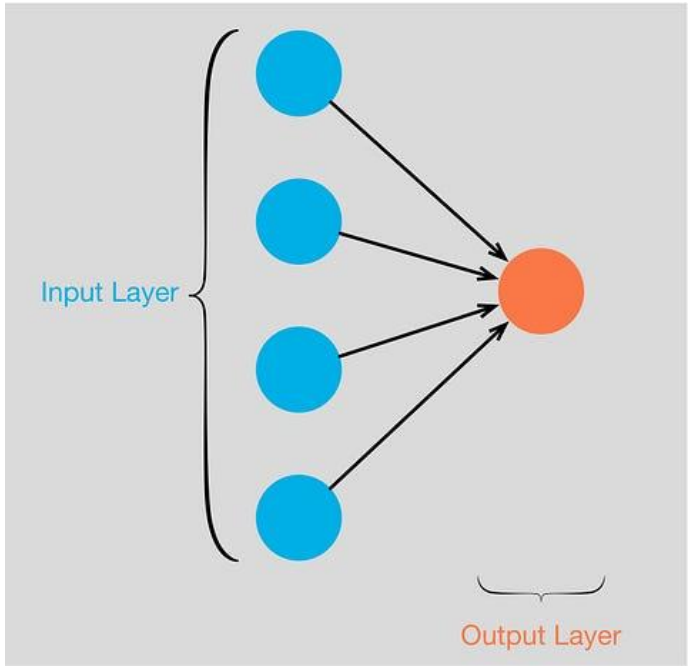
Sigmoid / Logistic

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



The Perceptron (Single Layer Perceptron)

A Perceptron consists of just one layer of TLUs (sometimes, it refers to a tiny network with a single TLU), with each TLU linked to all the inputs. When every neuron in a particular layer is connected to each neuron in the preceding layer (that is, its input neurons), this configuration is referred to as a fully connected layer or a dense layer.

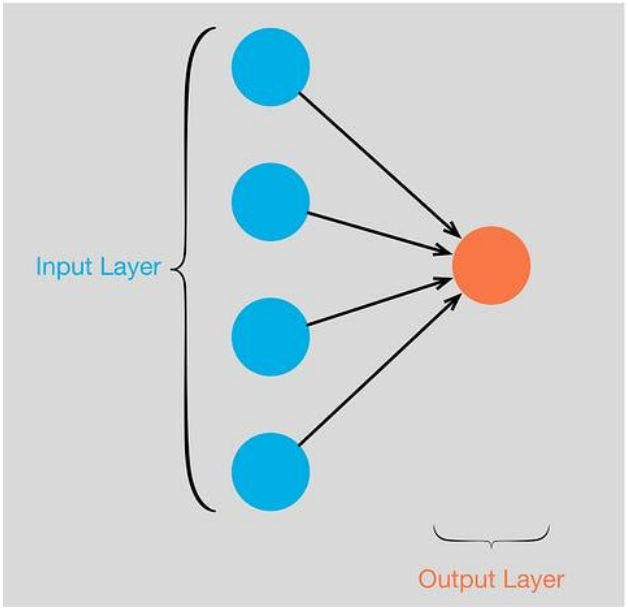


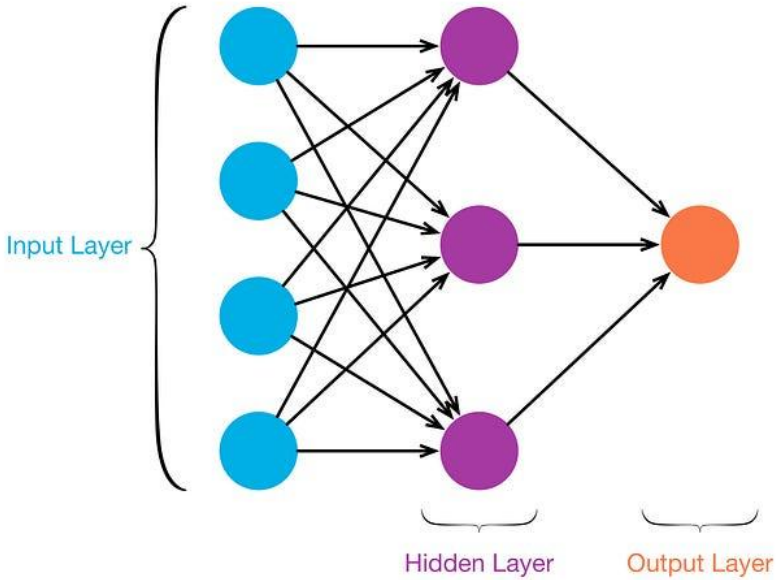
https://towardsdatascience.com/multi-layer-neural-networks-with-sigmoid-function-deep-learning-for-rookies-2-bf464f09eb7f

The Perceptron Weaknesses

Problem: the inability to solve nonlinear problems

Solution: stacking multiple Perceptrons resulting in Multilayer Perceptron (MLP)

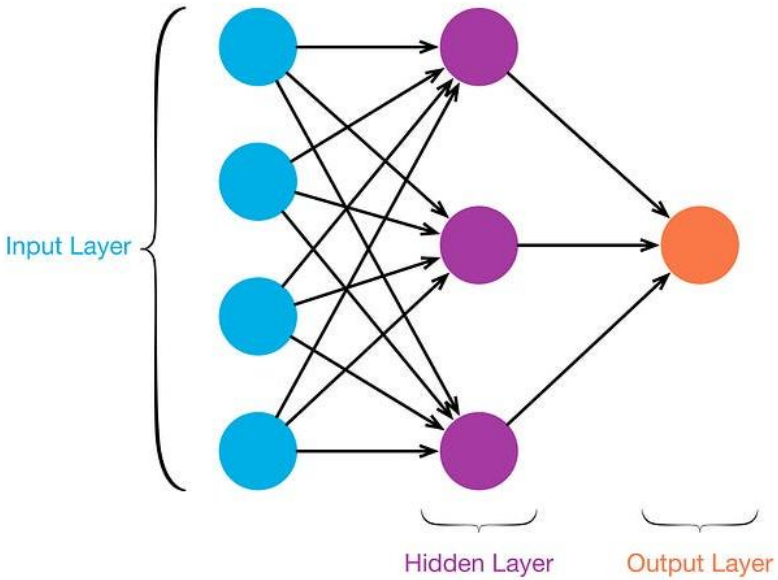




https://towardsdatascience.com/multi-layer-neural-networks-with-sigmoid-function-deep-learning-for-rookies-2-bf464f09eb7f

Multilayer Perceptron (MLP)

- Input layer
- Hidden layers
- Output layer



https://towardsdatascience.com/multi-layer-neural-networks-with-sigmoid-function-deep-learning-for-rookies-2-bf464f09eb7f

Perceptron

Scikit-Learn provides a Perceptron class that implements a single-TLU network.

```python
import numpy as np
from sklearn.datasets import load_iris
from sklearn.linear_model import Perceptron
iris = load_iris()
X = iris.data[:, (2, 3)] # petal length, petal width
y = (iris.target == 0).astype(np.int) # Iris setosa?

per_clf = Perceptron()
per_clf.fit(X, y)

y_pred = per_clf.predict([[2, 0.5]])
```

Multilayer Perceptron Training

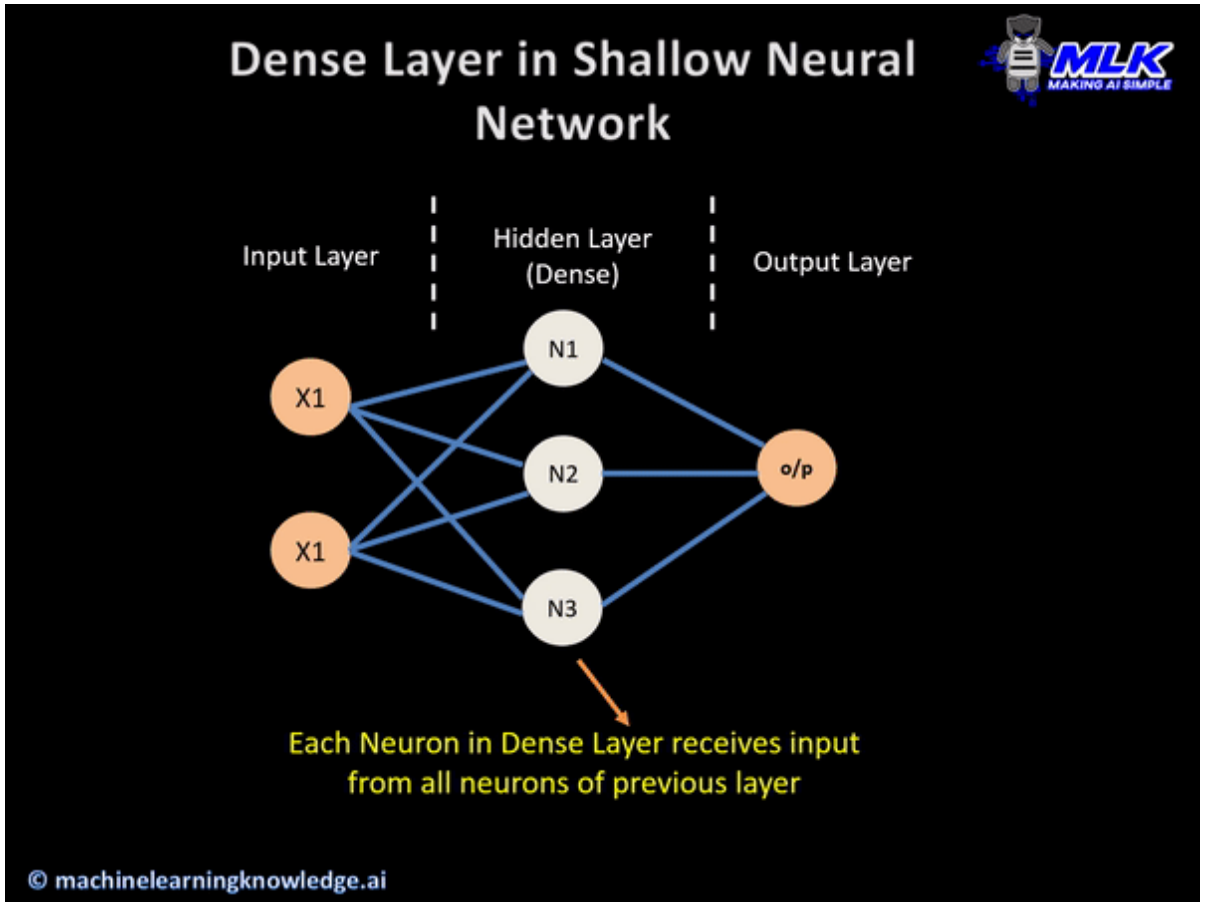
Multilayer Perceptron Training

Feedforward

Backpropagation

Feedforward Neural Network (FNN)

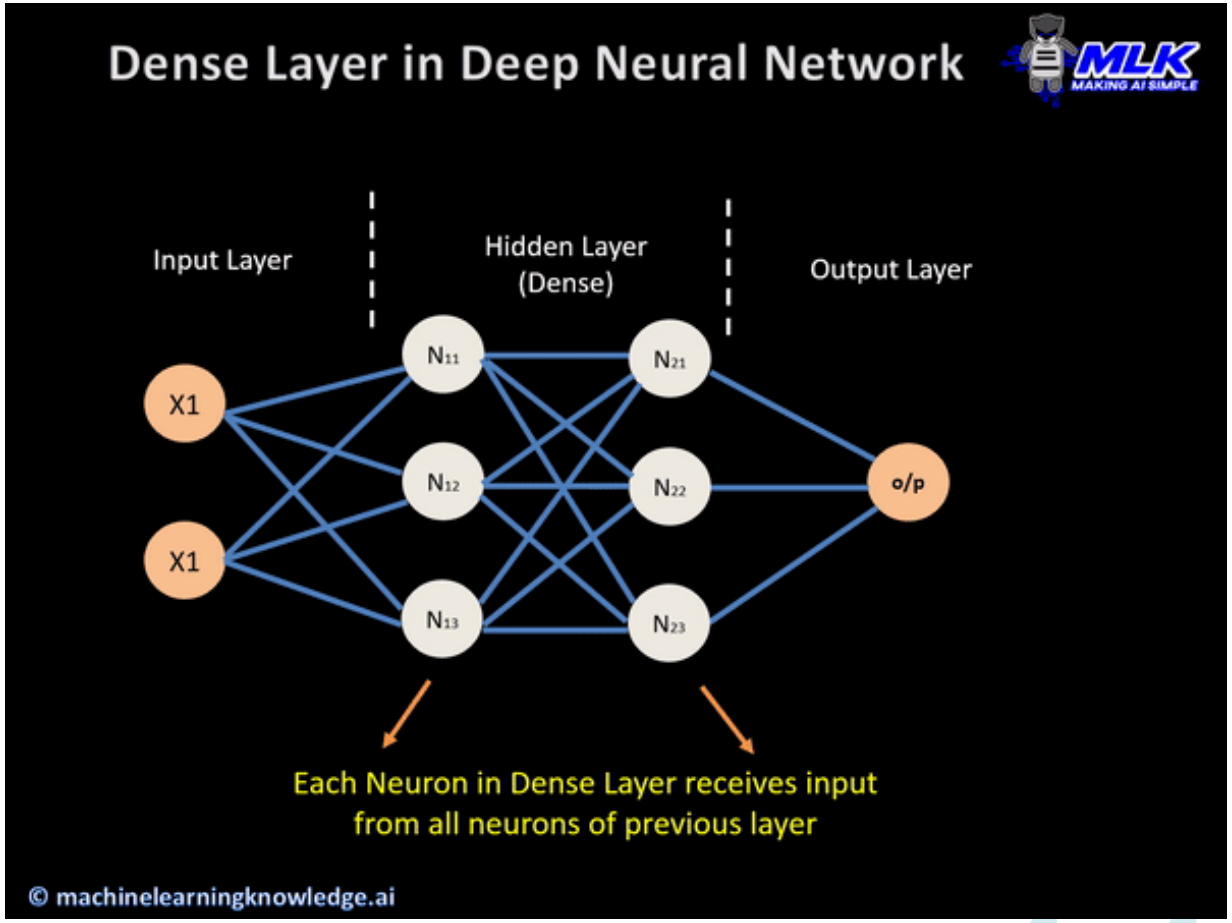
- Input to output
- One-way flow
- Simple structure



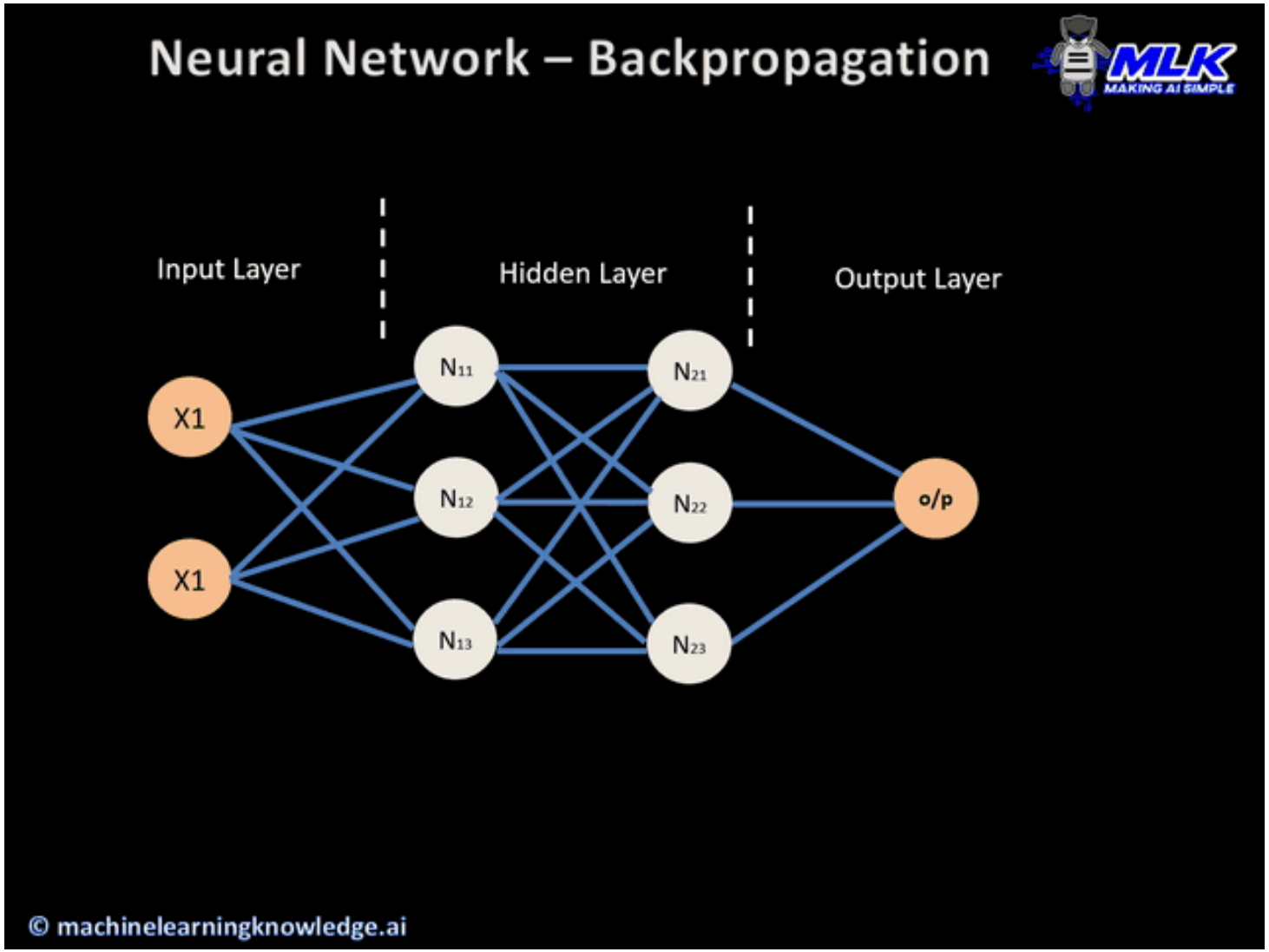
https://machinelearningknowledge.ai/keras-dense-layer-explained-for-beginners/

Feedforward Neural Network (FNN)

- Simplest type
- One-way flow
- Layered structure



https://machinelearningknowledge.ai/keras-dense-layer-explained-for-beginners/

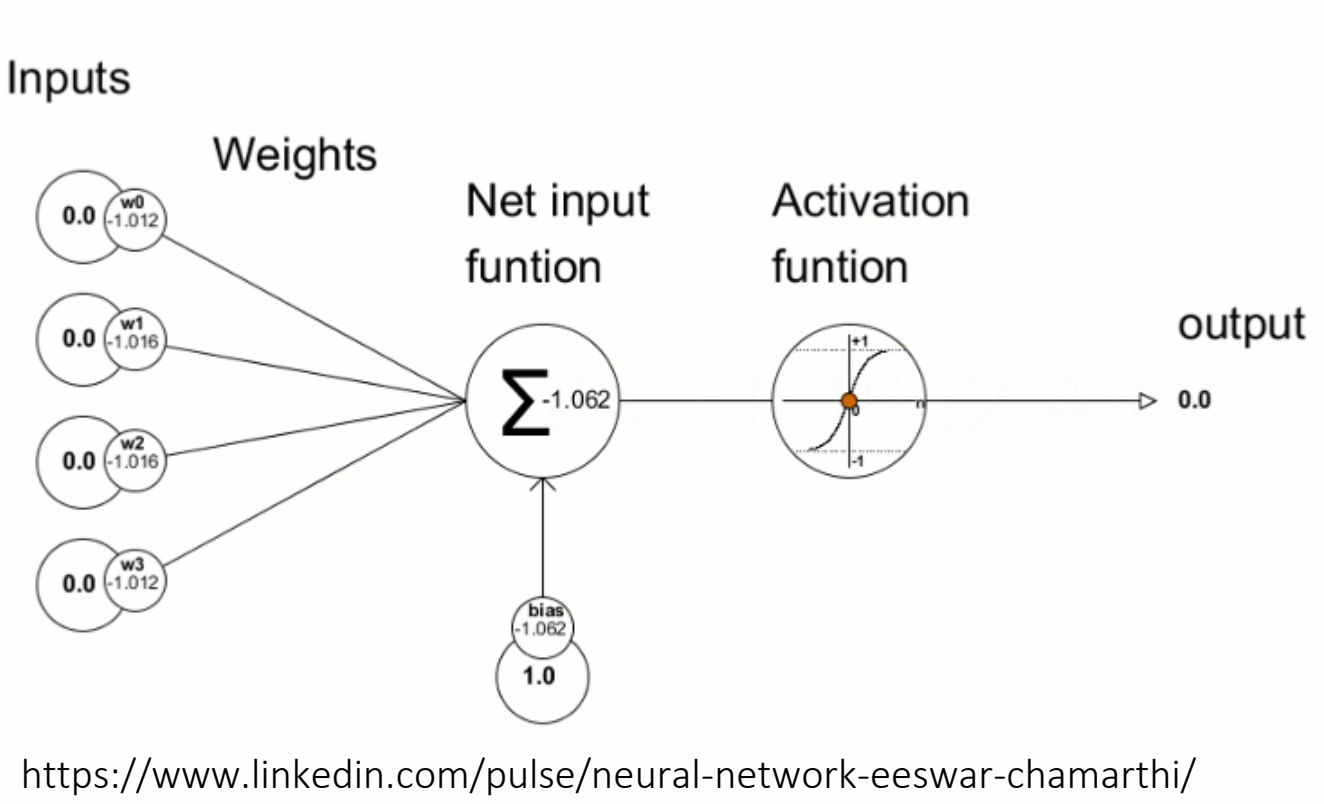


https://medium.com/analytics-vidhya/backpropagation-for-dummies-e069410fa585

Backpropagation

In 1986, David Rumelhart, Geoffrey Hinton, and Ronald Williams:

1. Forward Pass
2. Compute Loss
3. Backward Pass
4. Update Weights
5. Repeat 1-4 with each batch of inputs until the network performs satisfactorily.



https://www.linkedin.com/pulse/neural-network-eeswar-chamarthi/

Deep Neural Network (DNN)

When an artificial neural network (ANN) is constructed with a deep stack of hidden layers, it is referred to as a deep neural network (DNN).

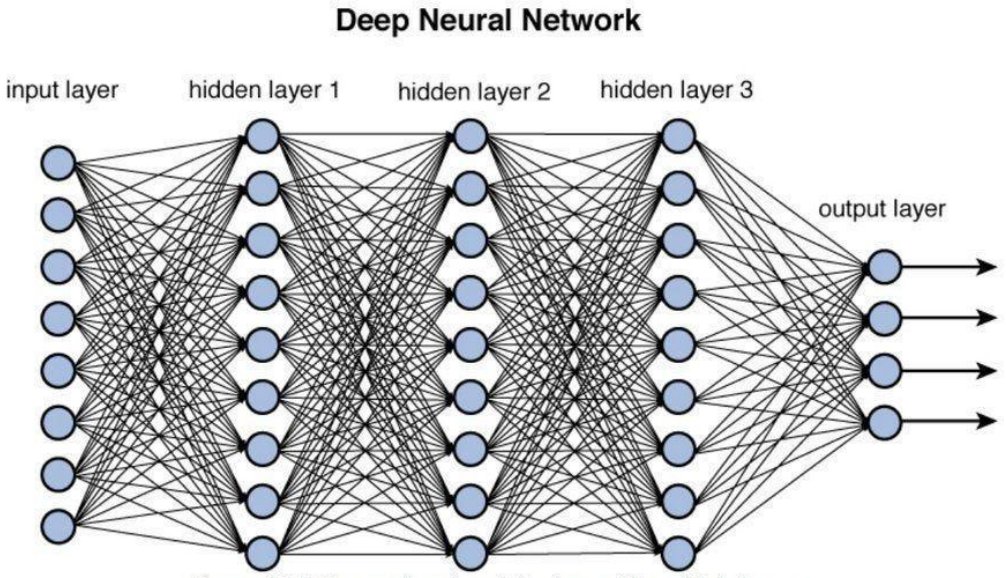


Figure 12.2 Deep network architecture with multiple layers.

https://towardsdatascience.com/training-deep-neural-networks-9fdb1964b964

problems could face when training a deep learning (10 layers or more) network

- ✓ During the training of deep neural networks (DNNs), you could encounter the challenge of gradients diminishing or expanding significantly as they propagate back through the network. This issue complicates the training of earlier layers.
- ✓ The network's large size might pose a challenge if there's not enough data available for training, or if the cost of data labeling is too high.
- ✓ The process of training can be significantly slow.
- ✓ A model with a vast number of parameters faces a substantial risk of overfitting the training data, especially when the data is scarce, noisy, or both.

Deep Learning Trends

Training **deep** neural networks (more than 5-10 layers) could only be possible in recent times with:

- Faster computing resources (GPU)
- Larger labeled training sets

Algorithmic Improvements in Deep Learning

- Responsive activation functions (e.g., RELU)
- Regularization (e.g., Dropout)
- Supervised pre-training
- Unsupervised pre-training (auto-encoders)

Specialized ANN Architectures:

- Convolutional Neural Networks (for image data)
- Recurrent Neural Networks (for sequence data)
- Residual Networks (with skip connections)

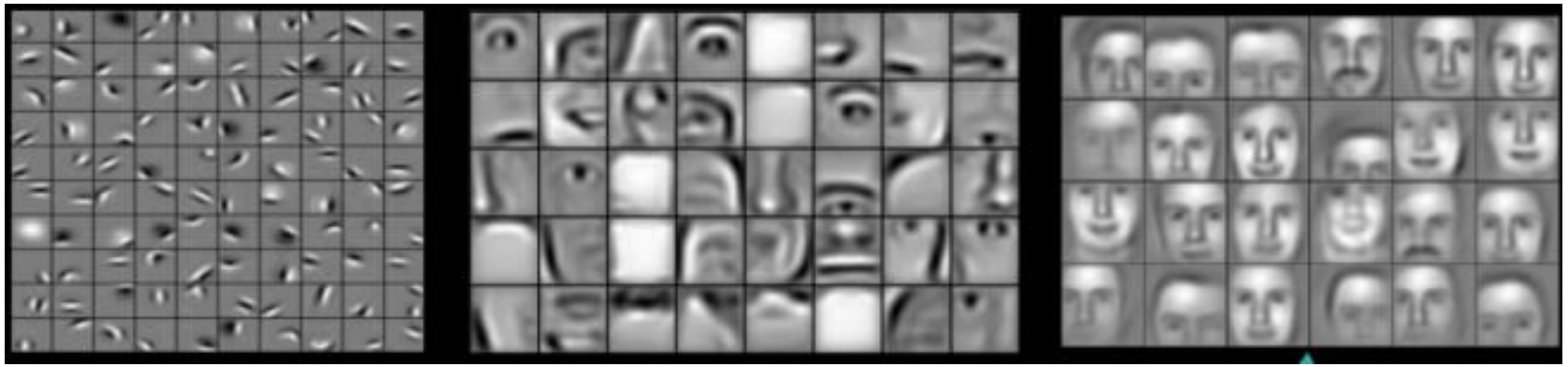
Generative Models: Generative Adversarial Networks

Why Multiple Hidden Layers?

Activations at hidden layers can be viewed as features extracted as functions of inputs

Every hidden layer represents a level of abstraction

Complex features are compositions of simpler features



Number of layers is known as **depth** of ANN

Deeper networks express complex hierarchy of features

Create a Test Set

- Looking at the test set can lead to biased decisions (**data snooping**)
- You just apply **simple random sampling** to select some examples, usually around 20% of the data set, and keep them separate. If your data set is exceptionally large, you might choose less than 20%.

```
from sklearn.model_selection import train_test_split

train_set, test_set = train_test_split(housing, test_size=0.2, random_state=42)
```

Business Understanding

Data Understanding

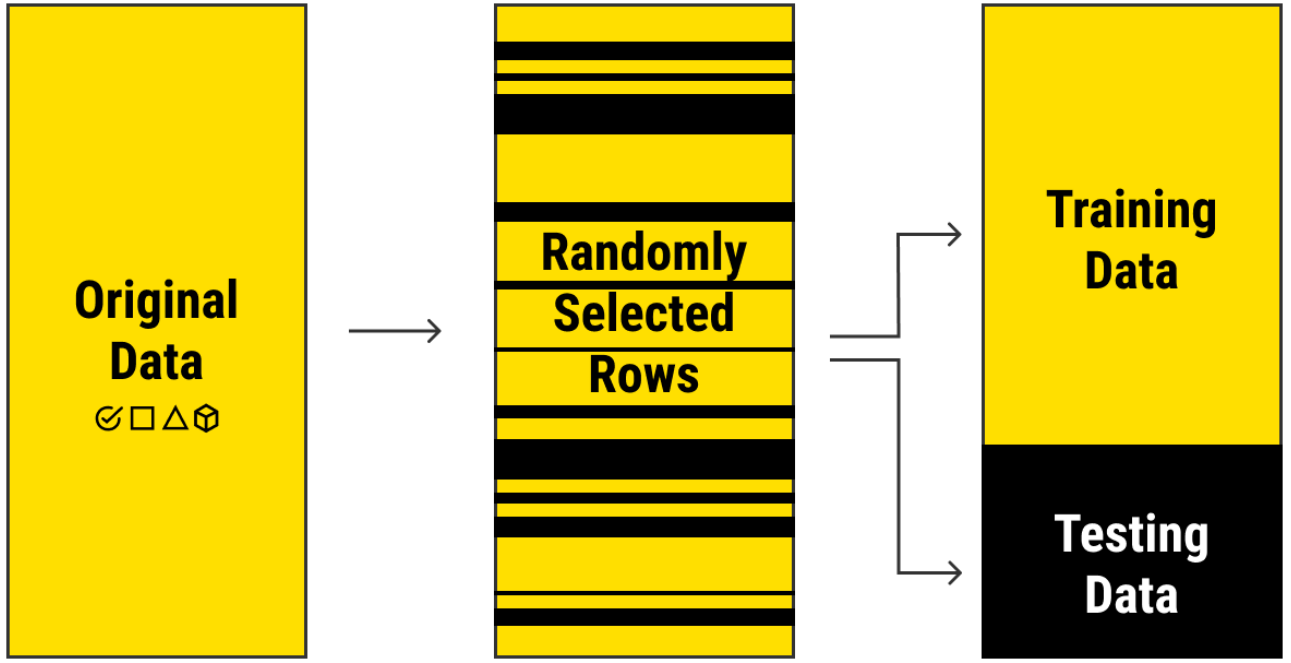
Data Preparation

Modelling

Evaluation

Deployment

Create a Test Set



Karyna Naminas. (2021, March 18). *Machine Learning and Training Data: What You Need to Know*. Labelyourdata.com; Label Your Data. https://labelyourdata.com/articles/machine-learning-and-training-data

```
from sklearn.model_selection import train_test_split

train_set, test_set = train_test_split(housing, test_size=0.2, random_state=42)
```

Types of Sampling

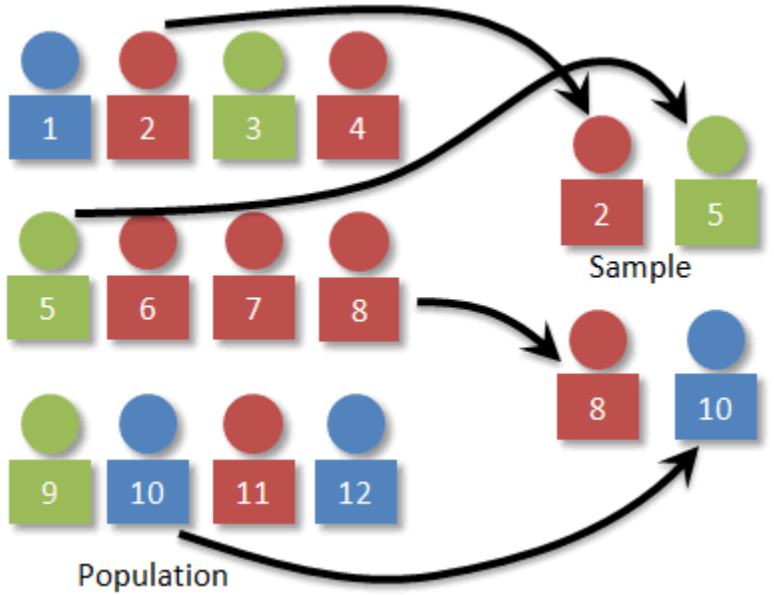
- Simple Random Sampling
- Stratified Sampling

(Simple) Random Sampling

Each individual in the population has an identical probability of being chosen or selected.

Frequently applied when the population is vast, and it is challenging to pinpoint a particular subgroup (the population is homogeneous)

E.g., in a study of public opinion, a group of people chosen at random from the phone book of a city could be selected



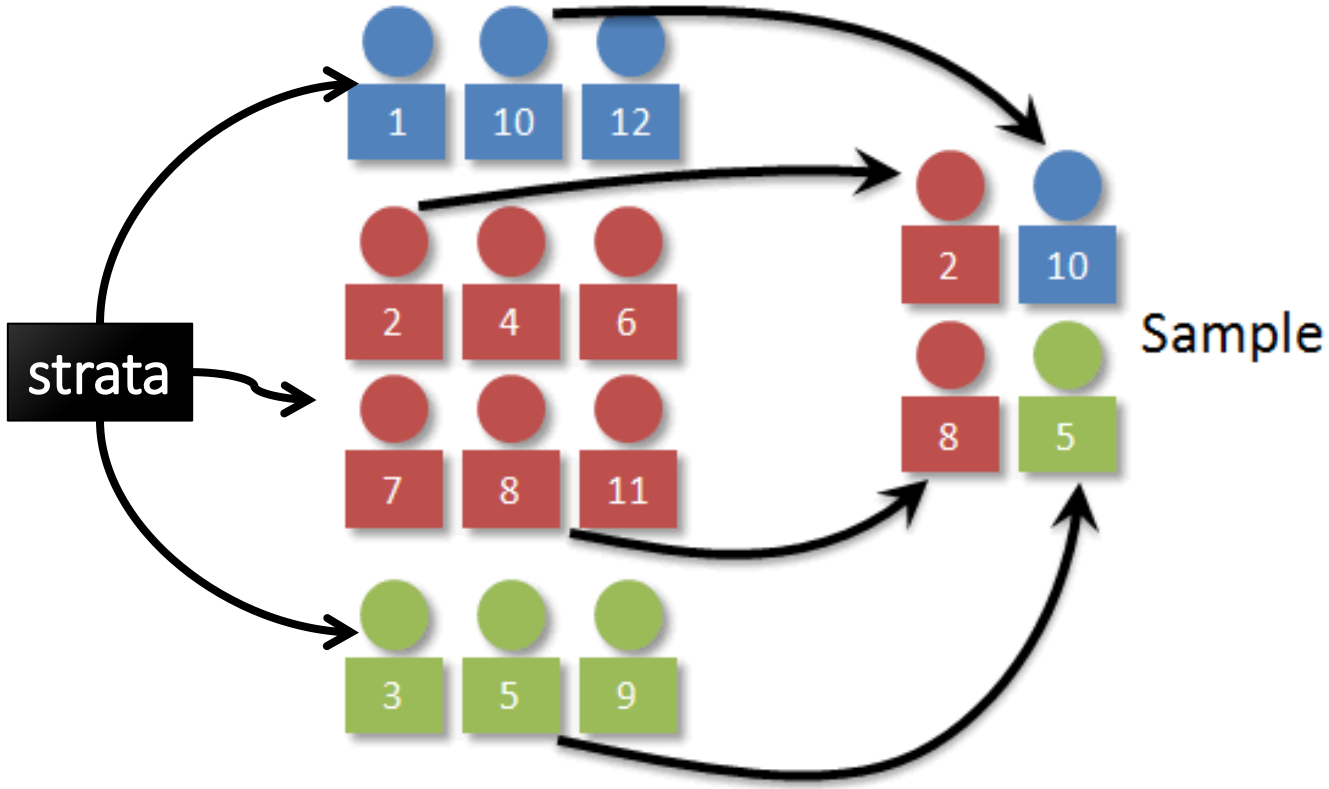
Source: faculty.elgin.edu

Stratified Sampling

dividing the population into subgroups, or strata, and then selecting a **proportional simple random sample** from each stratum.

used when the population can be divided into identifiable subgroups

more effective when researchers want to ensure that they have a representative sample from each subgroup (the population is heterogeneous)



Source: faculty.elgin.edu

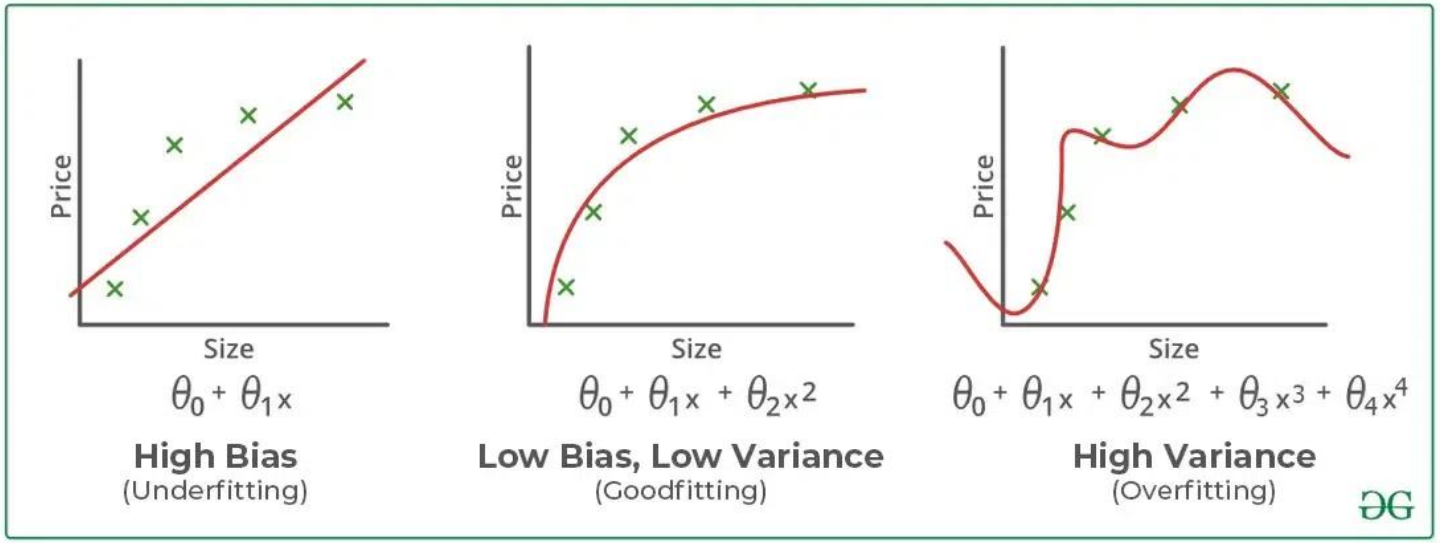
Create a Test Set (Stratified Sampling)

StratifiedShuffleSplit ensures that the test set is representative of the overall distribution of the income_cat variable in the dataset.

```
from sklearn.model_selection import StratifiedShuffleSplit

split = StratifiedShuffleSplit(n_splits=1, test_size=0.2, random_state=42)
for train_index, test_index in split.split(housing, housing["income_cat"]):
    strat_train_set = housing.loc[train_index]
    strat_test_set = housing.loc[test_index]
```

Underfitting vs Overfitting

- Underfitting (high bias) may happen when our model is over-simplified or not expressive enough (both training error and test error are high).
- Overfitting (high variance) may happen when our model is too complex and fits too specifically to the training set, but it does not generalize well to new data (low training error but high testing error).



Reasons of Underfitting

the model is too simple to capture the underlying patterns

The model is highly constrained

Sign of Overfitting

training score is good

Regression: training error is low

test score is bad

Regression: test error is high

Underfitting Remedy

- Select a more powerful model, with more parameters.
- Feed better features to the learning algorithm (feature engineering).
- Reduce the constraints on the model (e.g., reduce the regularization hyperparameter)

Reasons of Overfitting

Model Complexity	• *Example: Decision tree with high depth, Polynomial regression with high-degree polynomials*
Limited training data	• *Small amount of data as fewer data points, the model is more likely to overfit.*
Noisy data	• *Data with high amount of noise, can make it challenging to the model to capture underlying patterns. Consequently, the model may end up overfit*
Low Regularization	• *Low (insufficient) regularization can result in models that are too complex and prone to overfit*

Sign of Overfitting

training error is low

generalization error is high

Overfitting Remedy

simplify or regularize the model	For example, choose a linear regression model over a high-degree polynomial regression, or reduce the depth of a decision tree
get more training data	More data points can help the model generalize better and reduce overfitting.
clean up the training data (noise).	This can help the model focus on the underlying patterns in the data, rather than fitting the noise

Overfitting vs Underfitting

cross-validation to get an estimate of a model's generalization performance

Model performance on training data vs cross-validation

- Underfitting?
- Overfitting?

Generative AI

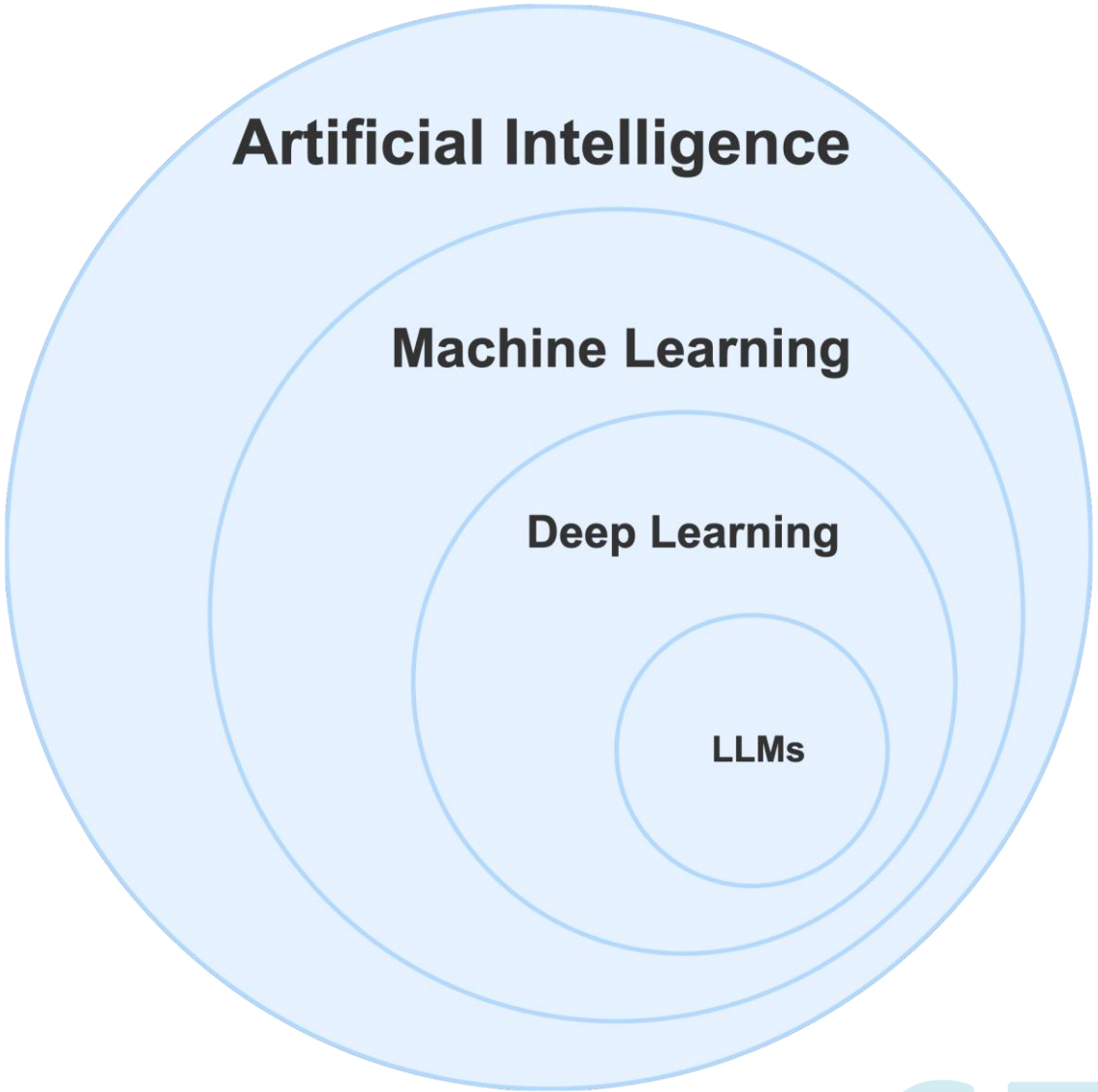
What is AI?

Key Concept:

"AI is programming a computer to react to data the way an intelligent being would."

[Artificial intelligence is] the science and engineering of making intelligent machines, especially intelligent computer programs. It is related to the similar task of using computers to understand human intelligence, but AI does not have to confine itself to methods that are biologically observable.

John McCarthy, 2007



What Are Large Language Models That Power Generative AI? (2024). Confidentialmind.com. https://www.confidentialmind.com/post/large-language-models-llms

Language AI (Natural Language Processing)

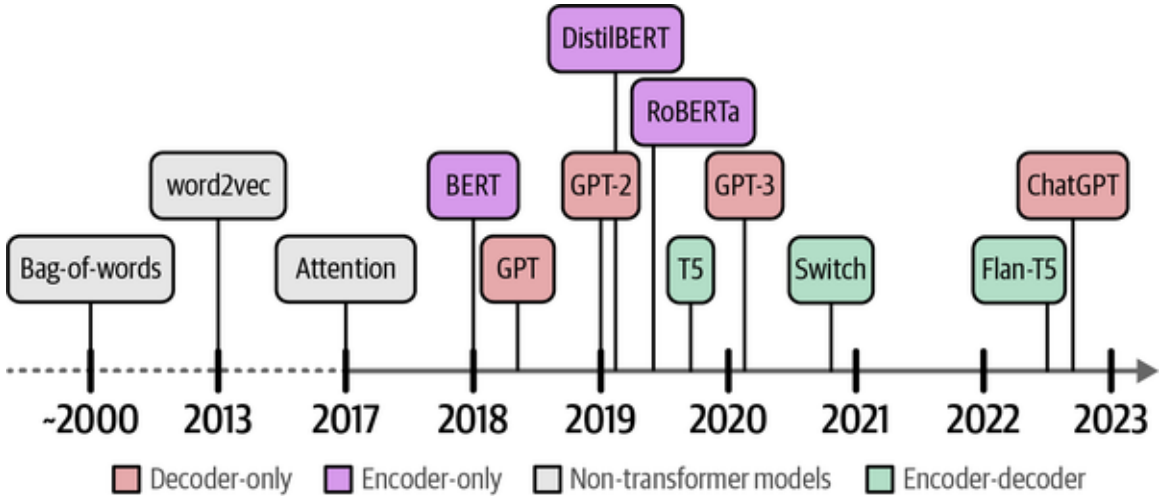
> “Language AI refers to a subfield of AI that focuses on developing technologies capable of understanding, processing, and generating human language.”
>
> Hands on Large Language Model P.2

Language AI

What is Language AI?

A broad term that includes technologies enabling machines to understand, process, and **generate** human language.

Encompasses **Large Language Models (LLMs)** and other tools that enhance language processing capabilities.



Introduction, A. (2022). *Hands-On Large Language Models*. O'Reilly Online Learning. https://www.oreilly.com/library/view/hands-on-large-language/9781098150952/ch01.html

Deep Learning Based Language AI

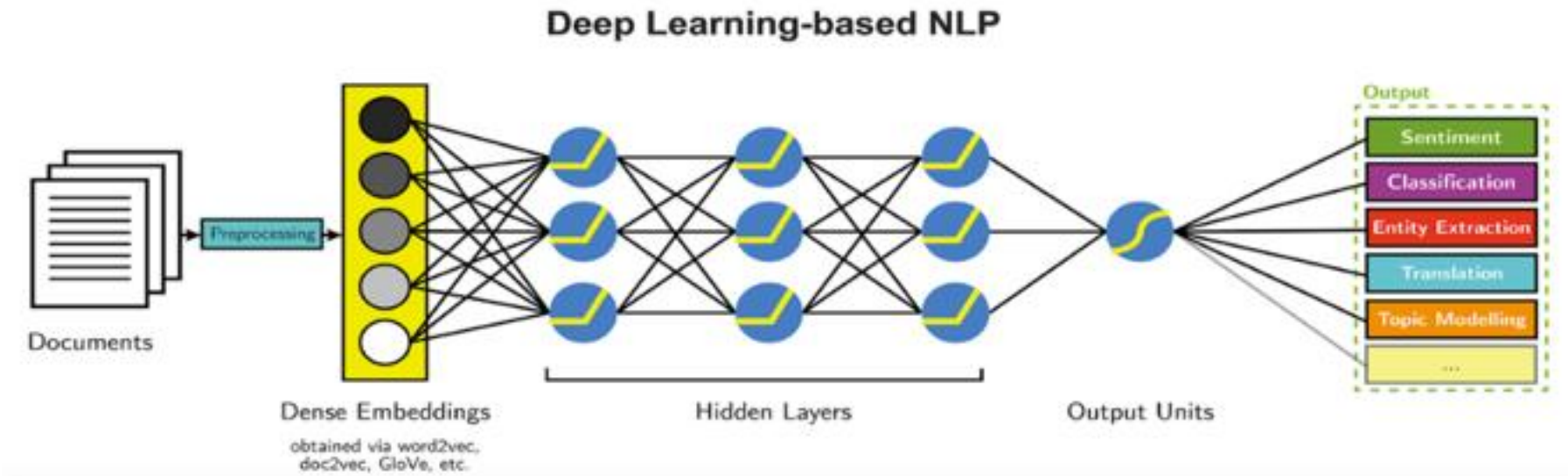


Image Credit: Parsa Ghaffari on the Aylien Blog

Language AI Evolution

Language AI Evolution

Bag-of-Words

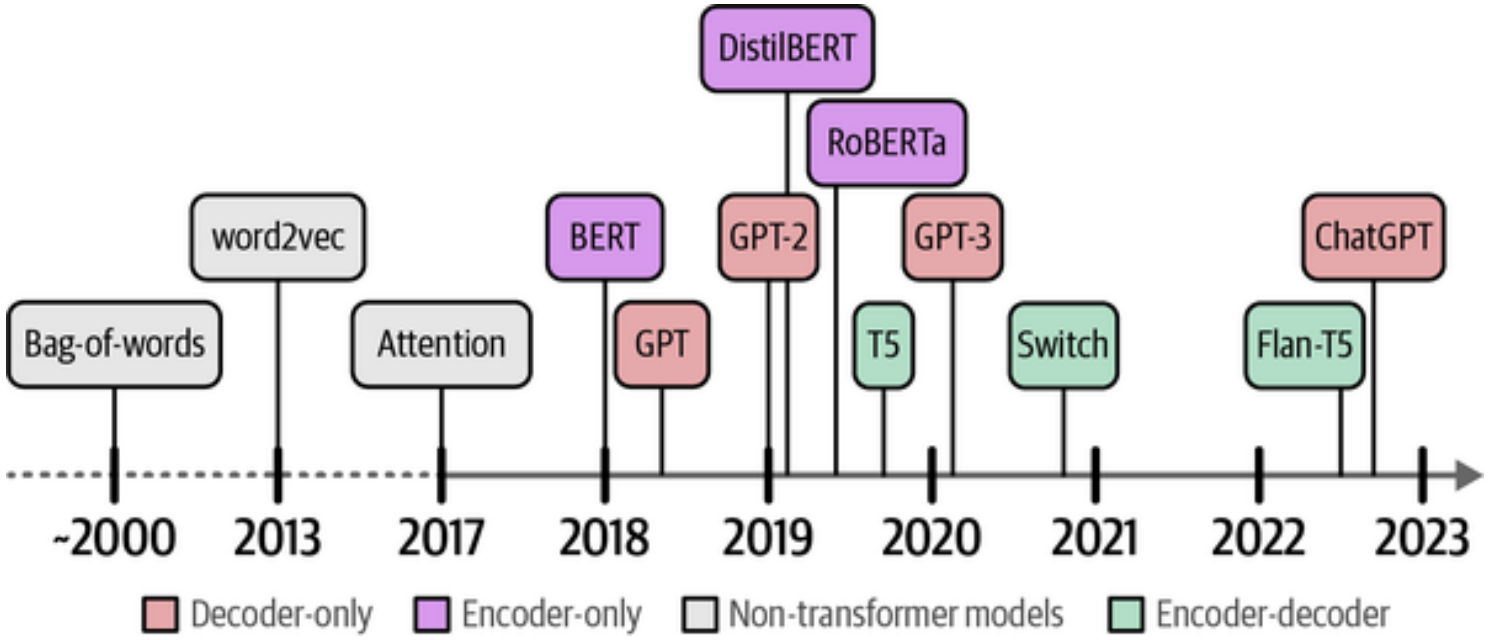
word2vec

Attention

Encoder only

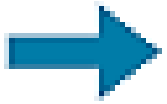
Decoder only

Encoder-decoder



One-Hot Encoding

Color
Red
Red
Yellow
Green
Yellow



Red	Yellow	Green
1	0	0
1	0	0
0	1	0
0	0	1
0	1	0

Vivek Praharsha. (2022, January 28). *One hot encoding in TensorFlow (tf.one_hot)*. OpenGenus IQ: Learn Algorithms, DL, System Design. https://iq.opengenus.org/one-hot-encoding-in-tensorflow/

One-Hot Encoding

- Traditional representation: one-hot vectors
  - ✓ *High-dimensional, sparse*
  - ✓ *Cannot capture similarity (e.g., "king" and "queen" look unrelated)*

Bag-of-Words (BoW)

Purpose: Convert text into numerical representations for machine learning.

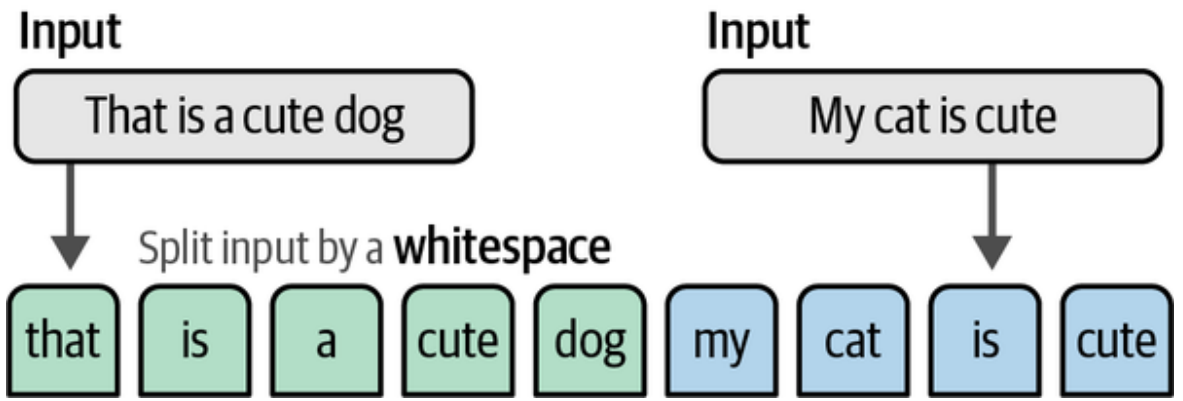
Process:

1. Tokenization
2. Vocabulary Creation
3. Vectorization

Bag-of-Words (BoW)

Process:

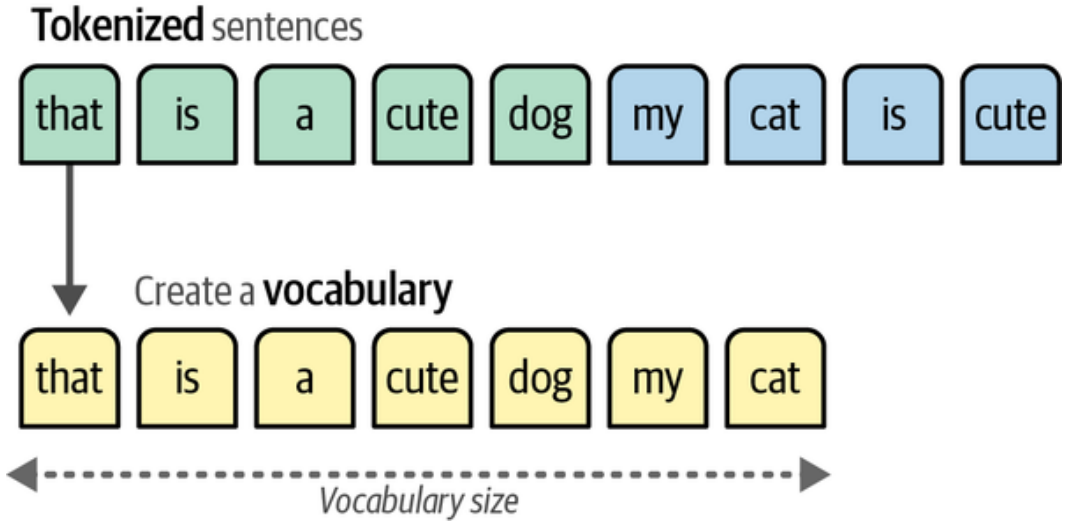
1. Tokenization
2. Vocabulary Creation
3. Vectorization



Bag-of-Words (BoW)

Process:

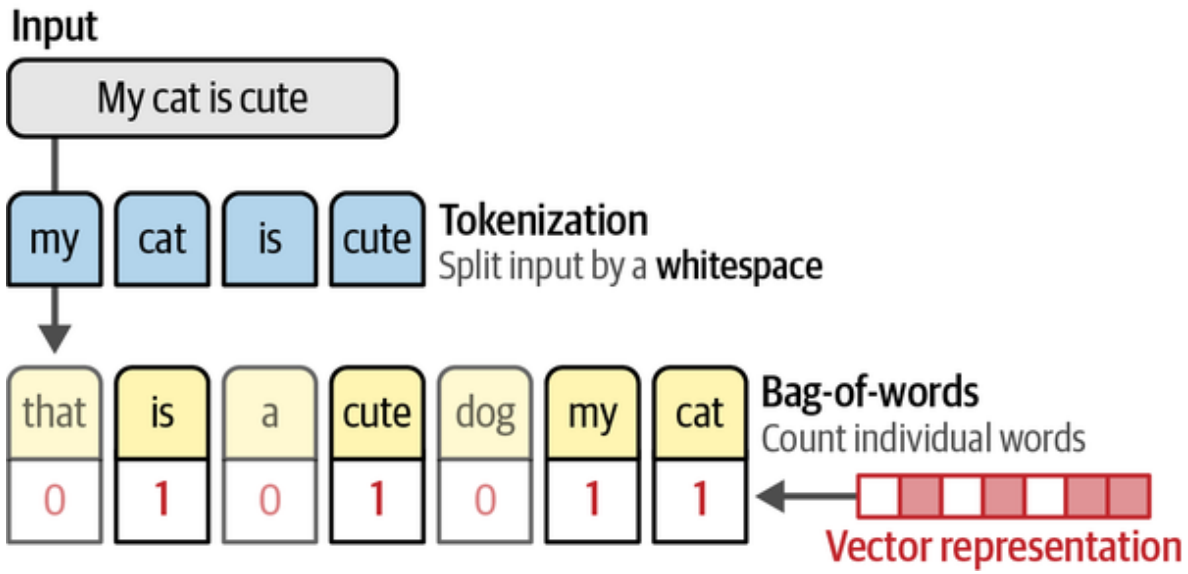
1. Tokenization
2. **Vocabulary Creation**
3. Vectorization



Bag-of-Words (BoW)

Process:

1. Tokenization
2. Vocabulary Creation
3. **Vectorization**



Limitations of Bag-of-Words

Treats text as a collection of individual words, ignoring context and semantics.

Fails to capture the meaning or relationships between words.

Language AI Evolution

Bag-of-Words

word2vec

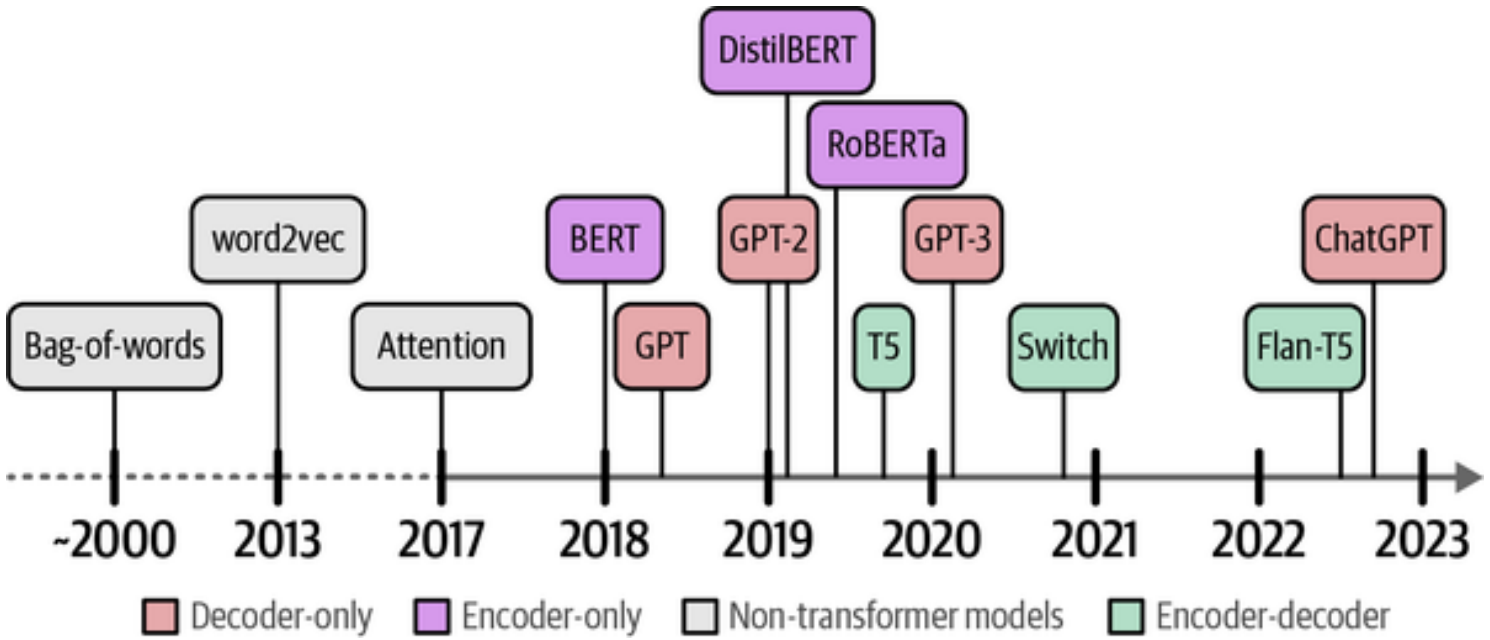
seq2seq

Attention

Encoder only

Decoder only

Encoder-decoder



Introduction to Word2Vec

- Developed in 2013
- Addresses BoW limitations
- Generates dense vector representations
- capture semantic relationships

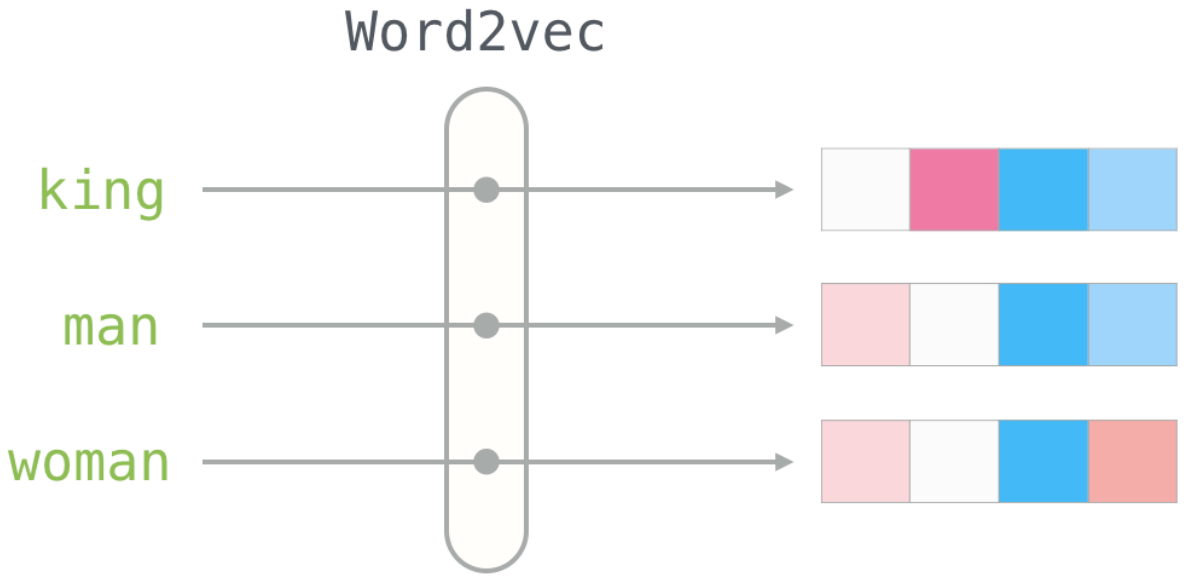
Word Embeddings

What is Embedding?

- Word2Vec is a model to turn **words into vectors** (embeddings)
- It captures **semantic meaning** (e.g., king – man + woman ≈ queen)
- Learns **from context**, unlike Bag-of-Words which ignores word order

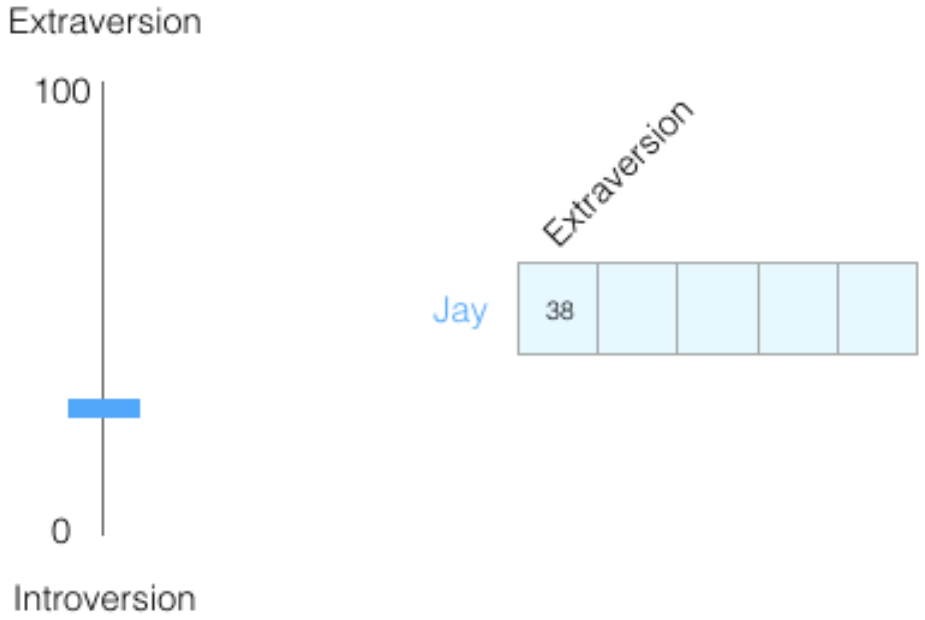
Word Embeddings

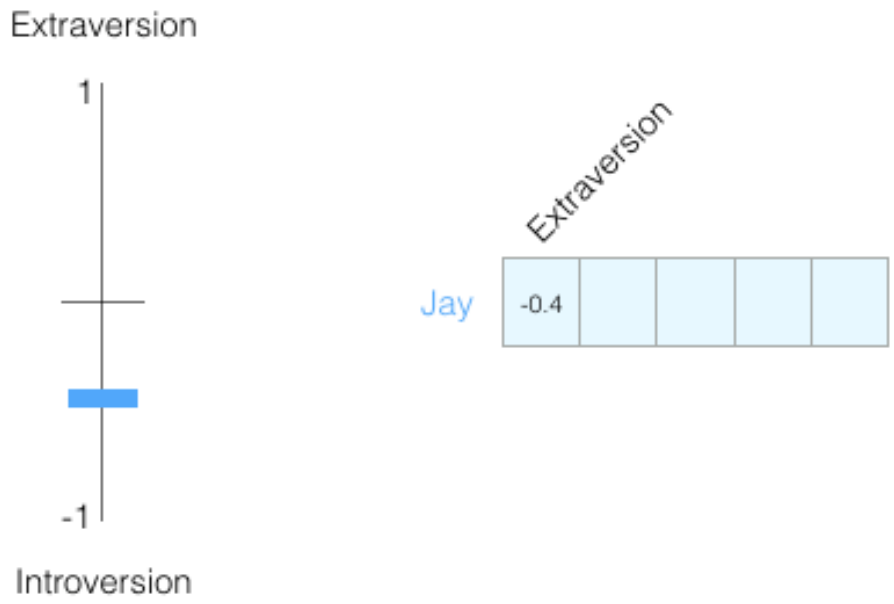
- A **word embedding** is a vector of real numbers (e.g 300 dimensions)
- Similar words → similar vectors
- Can capture **relationships** (gender, plural, country–capital, etc.)



Alammar, J. (2015). *The Illustrated Word2vec*. Github.io. https://jalammar.github.io/illustrated-word2vec/

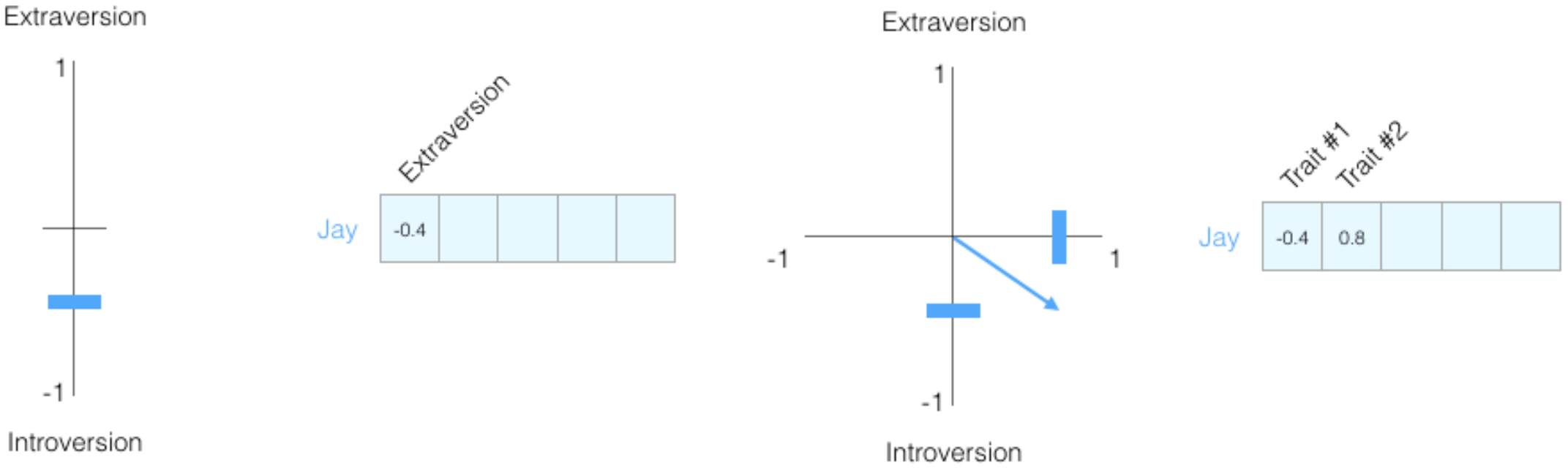
Personality Embedding





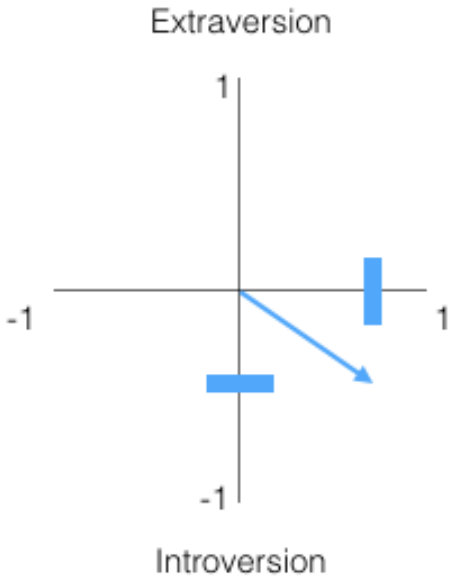
Alammar, J. (2015). *The Illustrated Word2vec*. Github.io. https://jalammar.github.io/illustrated-word2vec/

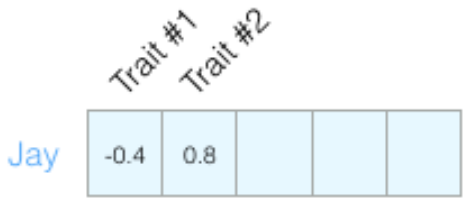
Personality Embedding

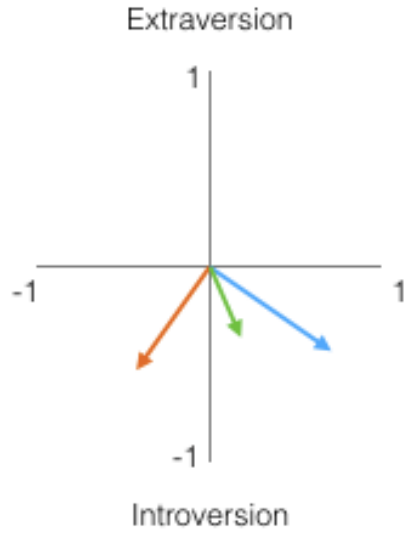


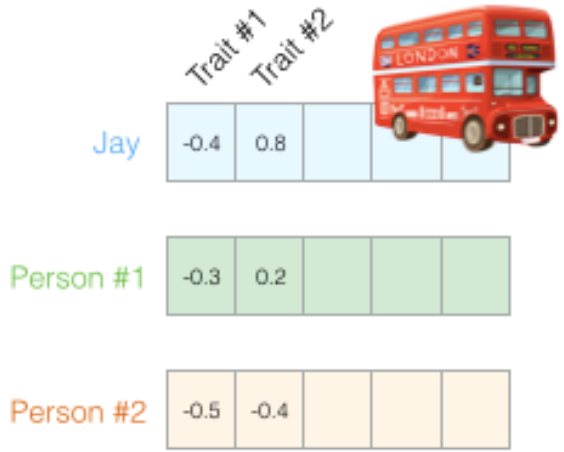
Alammar, J. (2015). *The Illustrated Word2vec*. Github.io. https://jalammar.github.io/illustrated-word2vec/

Personality Embedding



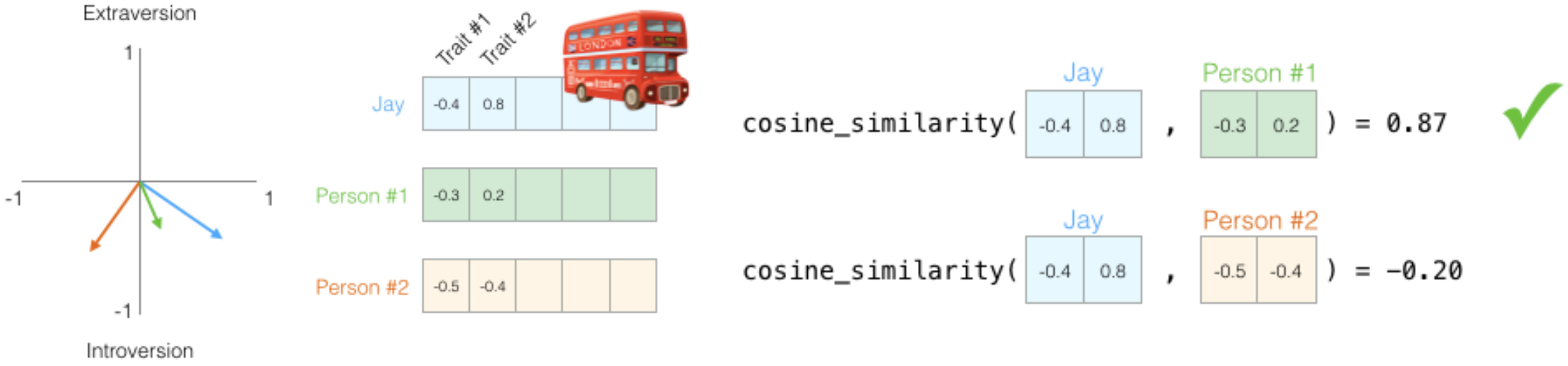






Alammar, J. (2015). *The Illustrated Word2vec*. Github.io. https://jalammar.github.io/illustrated-word2vec/

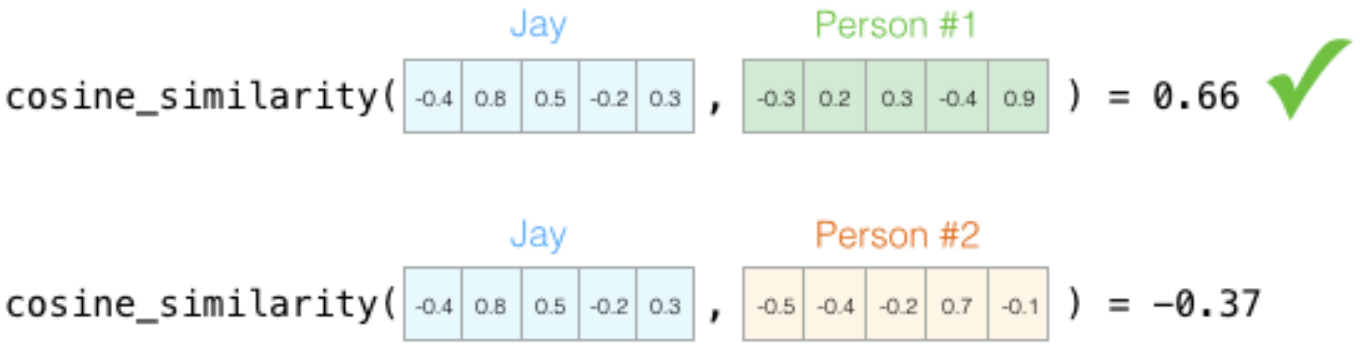
Personality Embedding



Alammar, J. (2015). *The Illustrated Word2vec*. Github.io. https://jalammar.github.io/illustrated-word2vec/

Personality Embedding

	Trait #1	Trait #2	Trait #3	Trait #4	Trait #5
Jay	-0.4	0.8	0.5	-0.2	0.3
Person #1	-0.3	0.2	0.3	-0.4	0.9
Person #2	-0.5	-0.4	-0.2	0.7	-0.1



Alammar, J. (2015). *The Illustrated Word2vec*. Github.io. https://jalammar.github.io/illustrated-word2vec/

Personality Embedding

1- We can represent things (and people) as vectors of numbers
(Which is great for machines!)

Jay	-0.4	0.8	0.5	-0.2	0.3

2- We can easily calculate how similar vectors are to each other

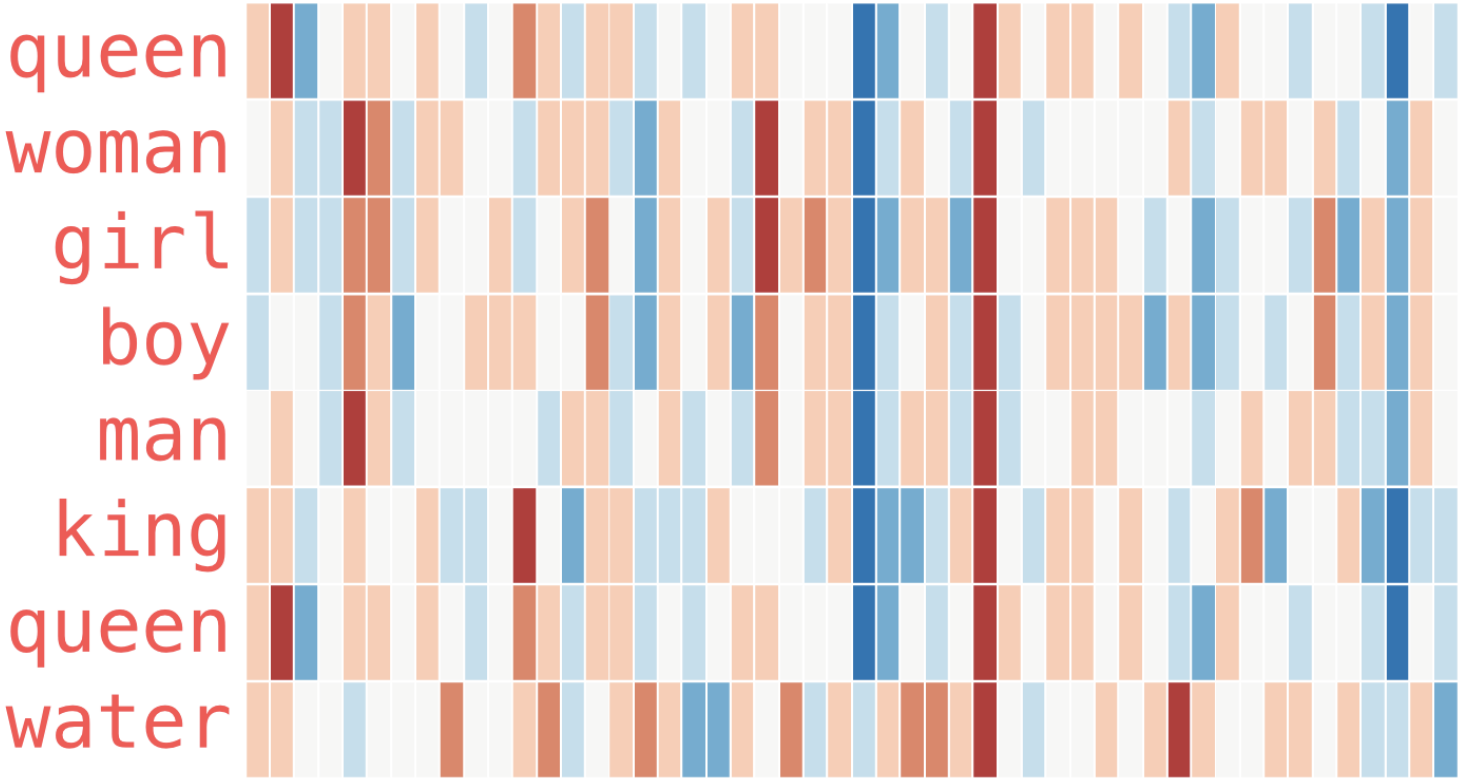
The people most similar to Jay are:

	cosine_similarity ▼
Person #1	0.86
Person #2	0.5
Person #3	-0.20

Alammar, J. (2015). *The Illustrated Word2vec*. Github.io. https://jalammar.github.io/illustrated-word2vec/

Word Embeddings

Let's Colab!

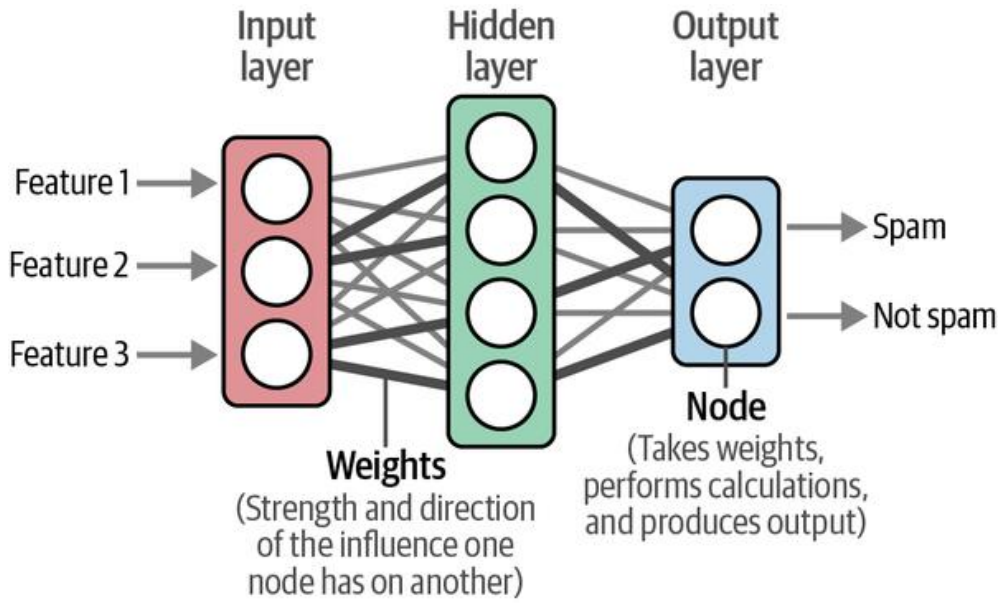


Alammar, J. (2015). *The Illustrated Word2vec*. Github.io. https://jalammar.github.io/illustrated-word2vec/

How Word2Vec Works

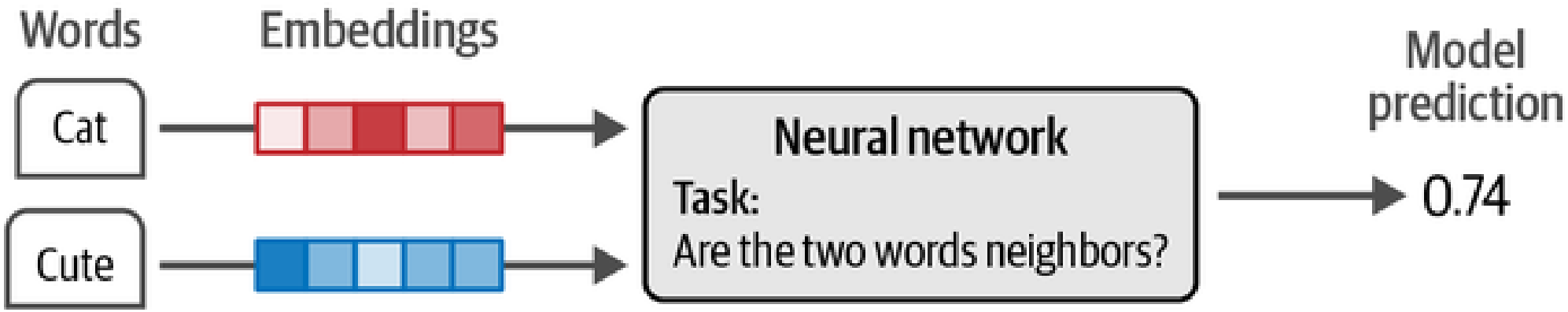
Uses **neural networks** to learn patterns

Embeddings = vectors capturing word meaning



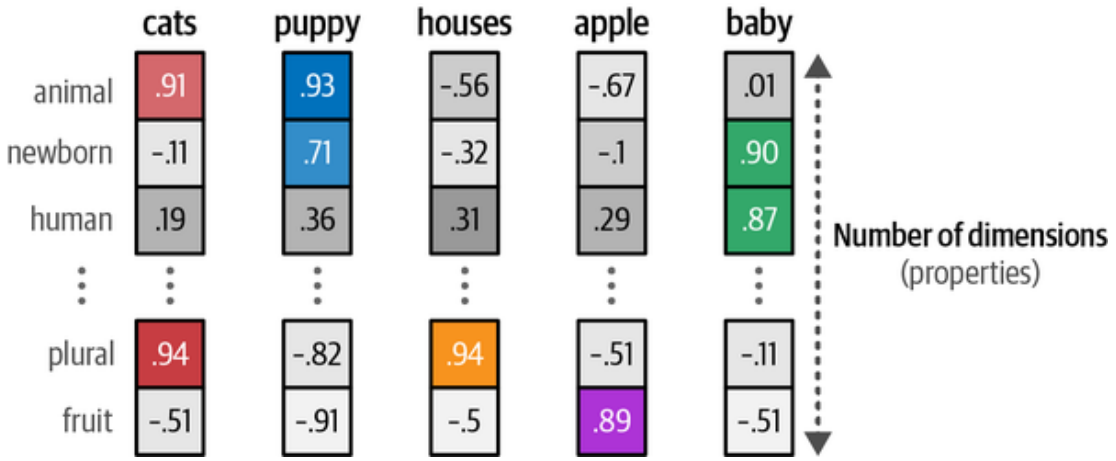
Alammar, J. (2015). *The Illustrated Word2vec*. Github.io. https://jalammar.github.io/illustrated-word2vec/

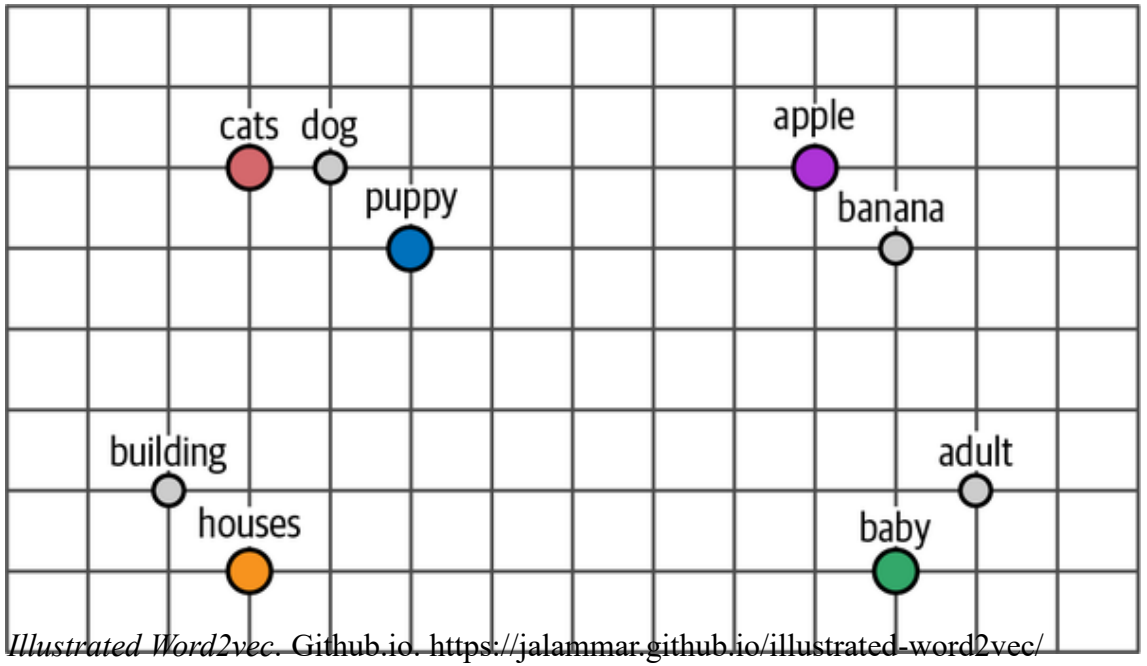
Word2Vec



Alammar, J. (2015). *The Illustrated Word2vec*. Github.io. https://jalammar.github.io/illustrated-word2vec/

Word2Vec

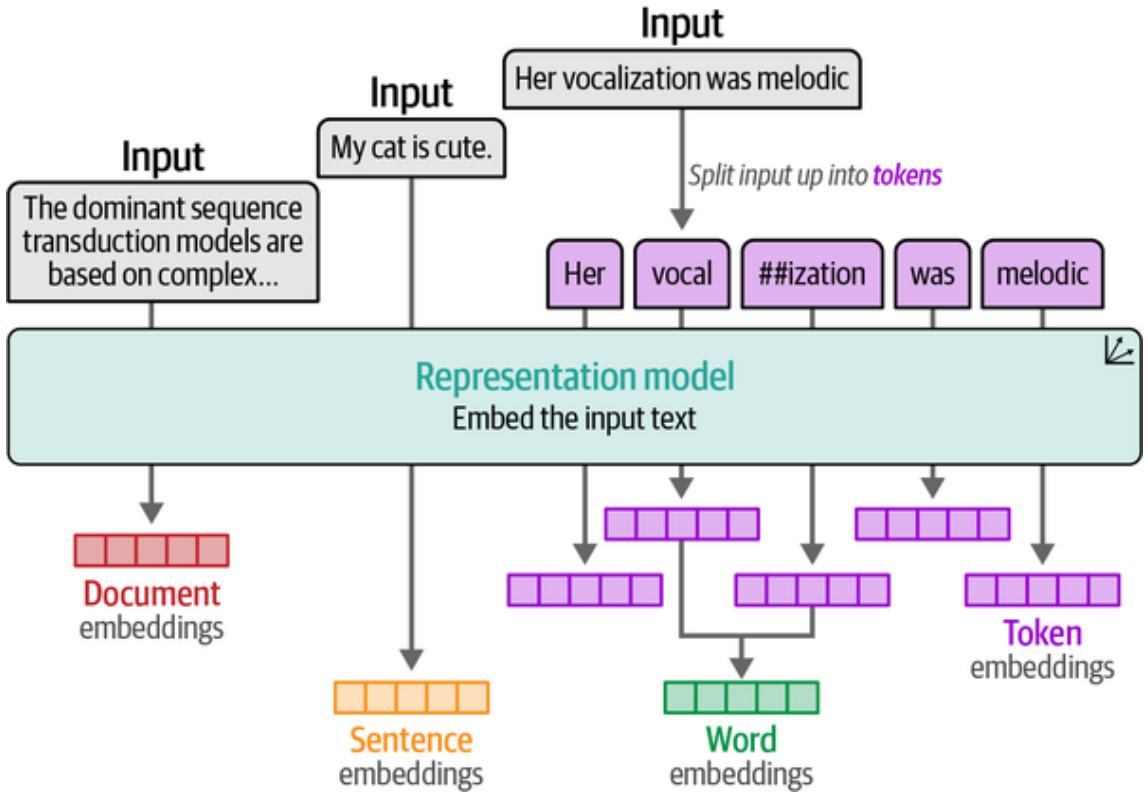




Alammar, J. (2015). *The Illustrated Word2vec*. Github.io. https://jalammar.github.io/illustrated-word2vec/

Types of Embeddings

- **Word embeddings** (e.g., Word2Vec)
- **Sentence/document embeddings** (e.g., Bag-of-Words)



Alammar, J. (2015). *The Illustrated Word2vec*. Github.io. https://jalammar.github.io/illustrated-word2vec/

Embedding Applications

- Classification
- Clustering
- Semantic search & retrieval

Limitation of word2vec

- **Word2Vec**: One fixed vector per word (e.g., “bank” = same for finance and river)
- ❌ Fails to capture **contextual meaning**

Language AI Evolution

Bag-of-Words

Word2vec

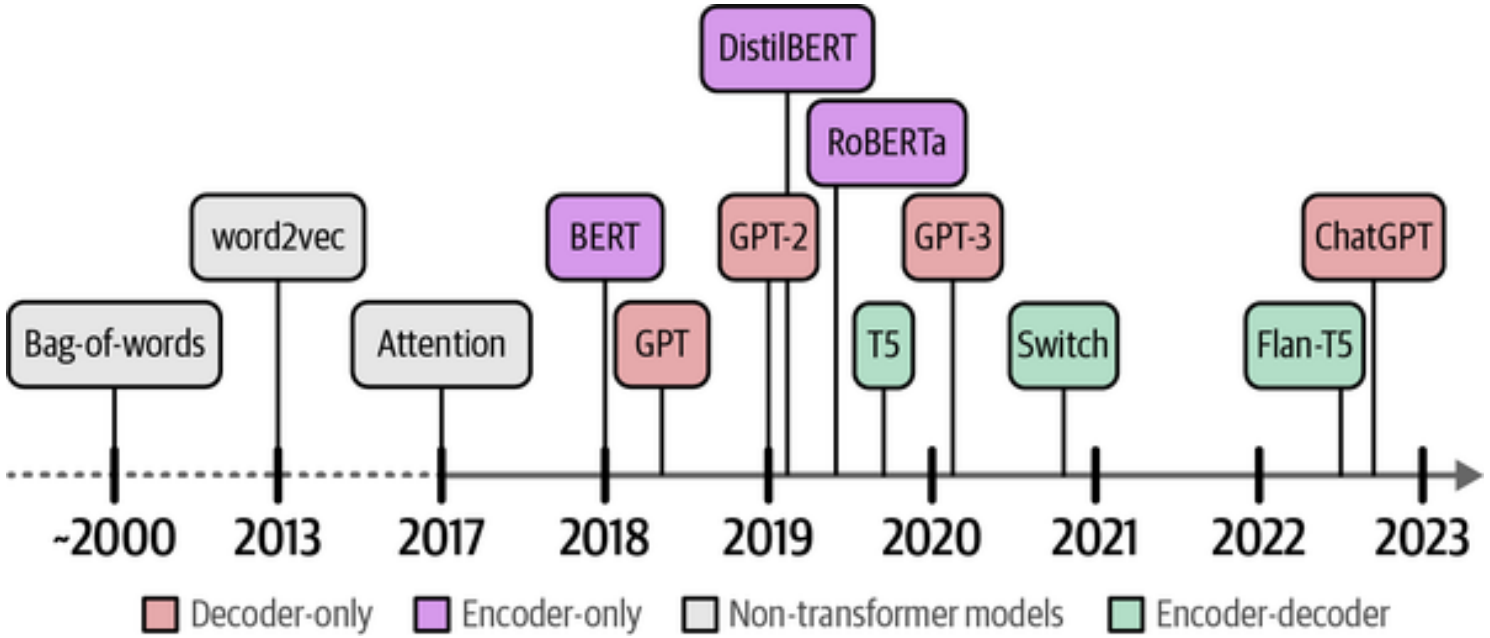
seq2seq

Attention

Encoder only

Decoder only

Encoder-decoder

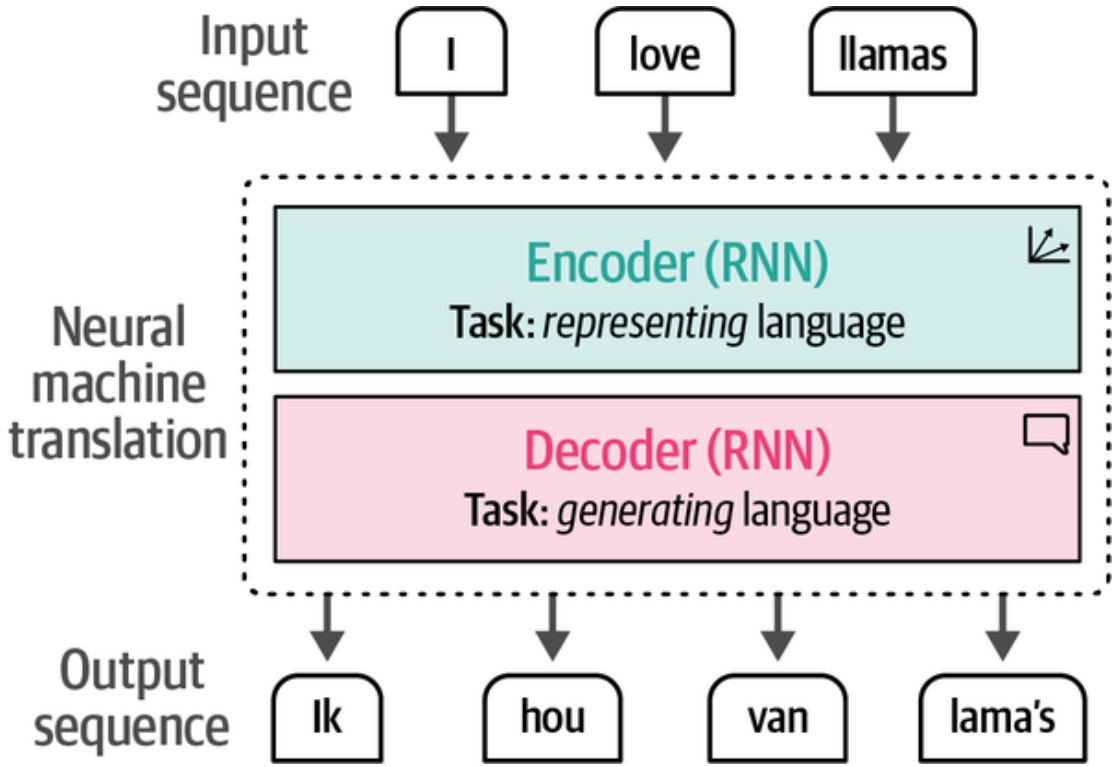


Alammar, J. (2015). *The Illustrated Word2vec*. Github.io. https://jalammar.github.io/illustrated-word2vec/

2014 – Introduction of Attention in RNNs

achieved through recurrent neural networks (RNNs)

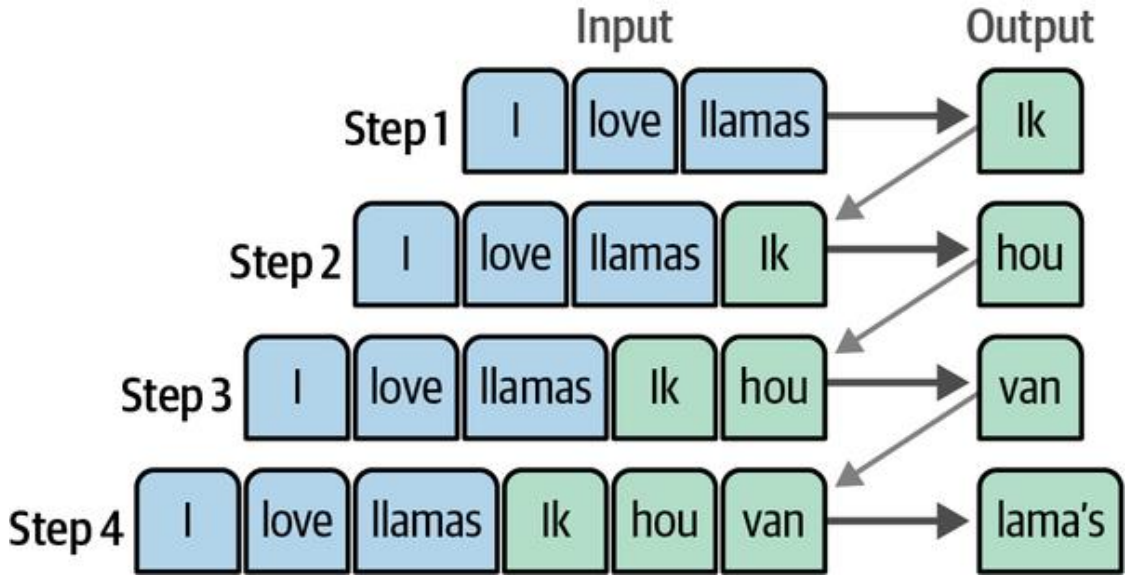
Bahdanau et al. introduced attention mechanisms in sequence-to-sequence (seq2seq) models.



Alammar, J. (2015). *The Illustrated Word2vec*. Github.io. https://jalammar.github.io/illustrated-word2vec/

2014 – Introduction of Attention in RNNs

Each step in this architecture is *autoregressive*



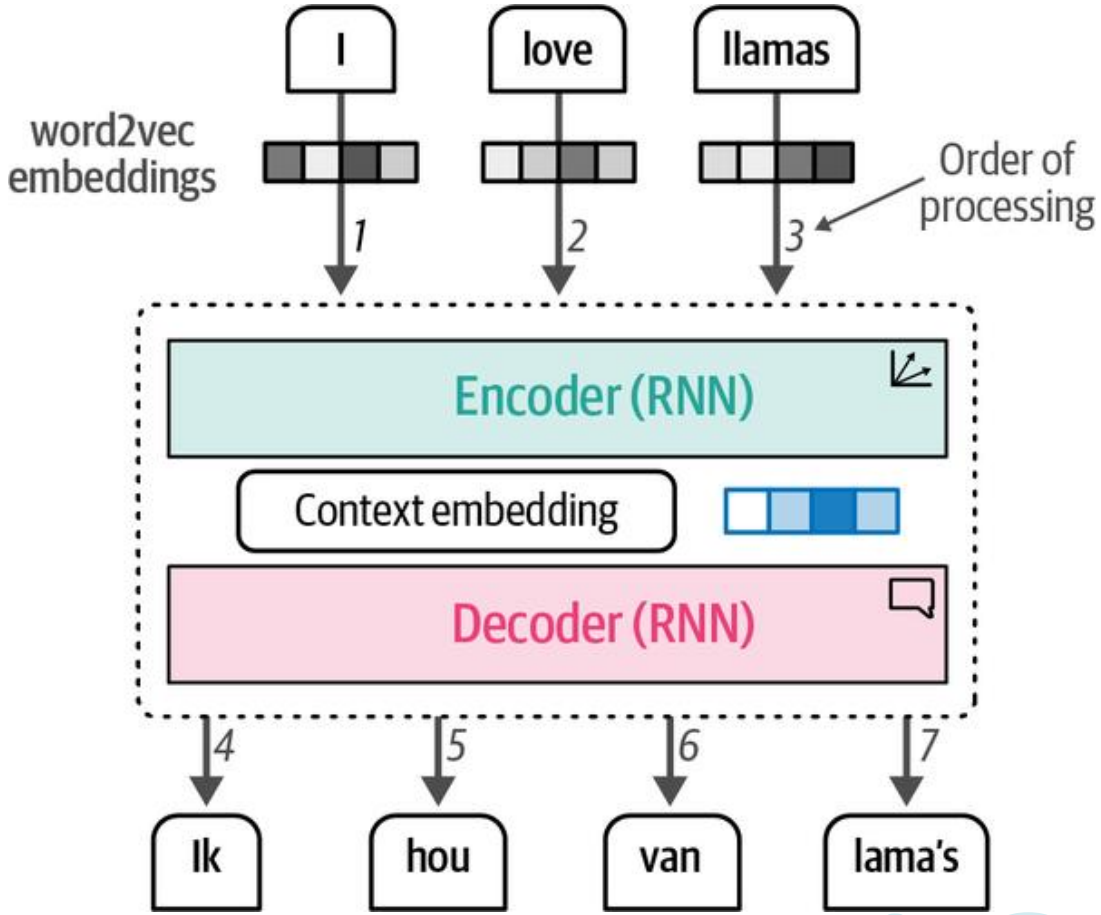
Alammar, J. (2015). *The Illustrated Word2vec*. Github.io. https://jalammar.github.io/illustrated-word2vec/

2014 – Introduction of Attention in RNNs

Objective: Transform an input sequence into a meaningful context representation for the decoder.

Process:

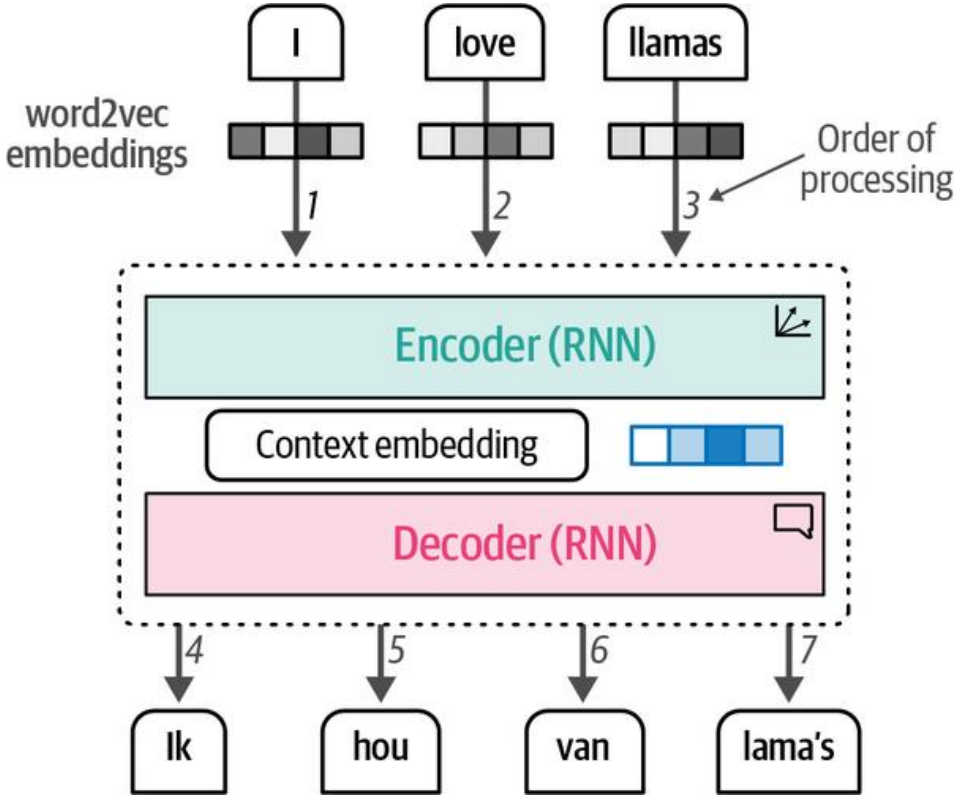
- Word Embeddings (word2vec)
- Sequential Processing (seq2seq)
- Context Generation



Alammar, J. (2015). *The Illustrated Word2vec*. Github.io. https://jalammar.github.io/illustrated-word2vec/

The Challenge: Fixed Context Vector

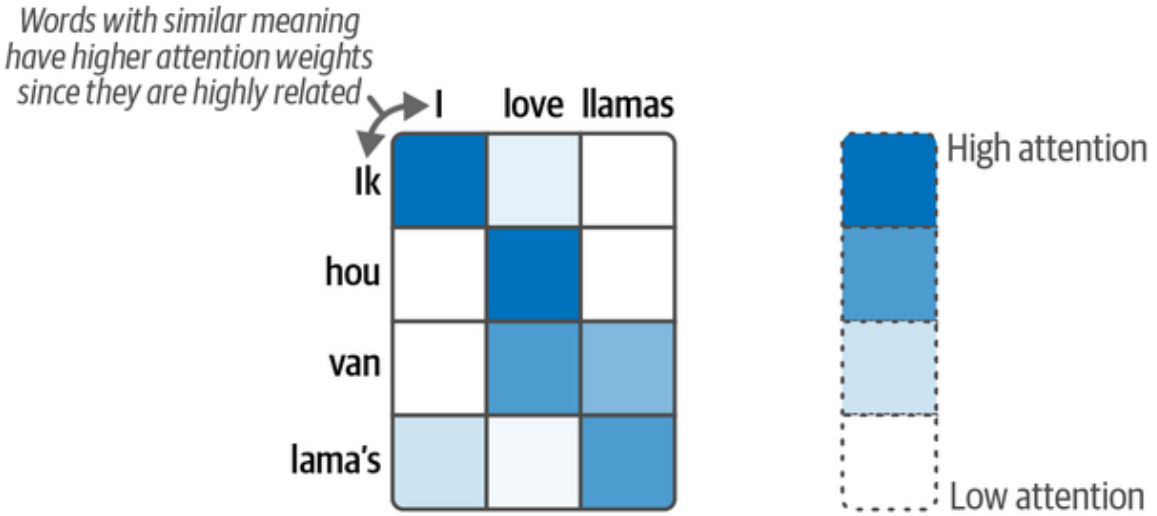
- encodes the entire input sequence into a single fixed-length context vector.
- struggles with longer sentences



Alammar, J. (2015). *The Illustrated Word2vec*. Github.io. https://jalammar.github.io/illustrated-word2vec/

The Solution: Attention Mechanism (Introduced in 2014)

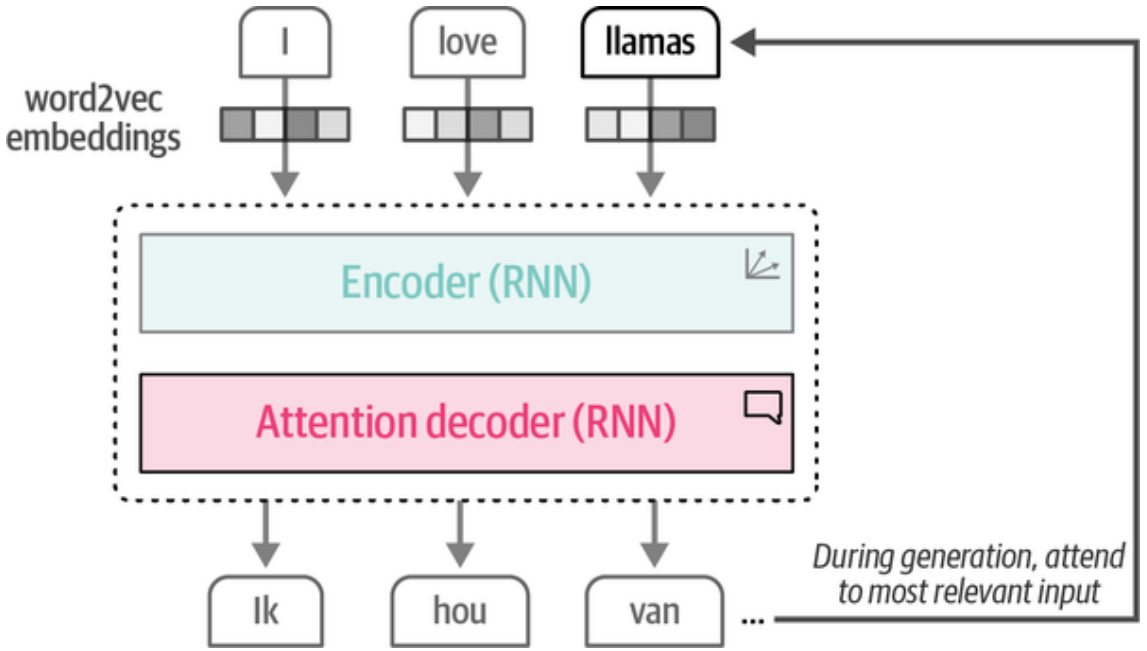
Dynamic Focus: Allows the model to focus on different parts of the input sequence during each step of decoding.



Alammar, J. (2015). *The Illustrated Word2vec*. Github.io. https://jalammar.github.io/illustrated-word2vec/

Attention Mechanism

Generates a context vector as a weighted sum of encoder states, emphasizing relevant parts of the input.



Alammar, J. (2015). *The Illustrated Word2vec*. Github.io. https://jalammar.github.io/illustrated-word2vec/

Language AI Evolution

Bag-of-Words

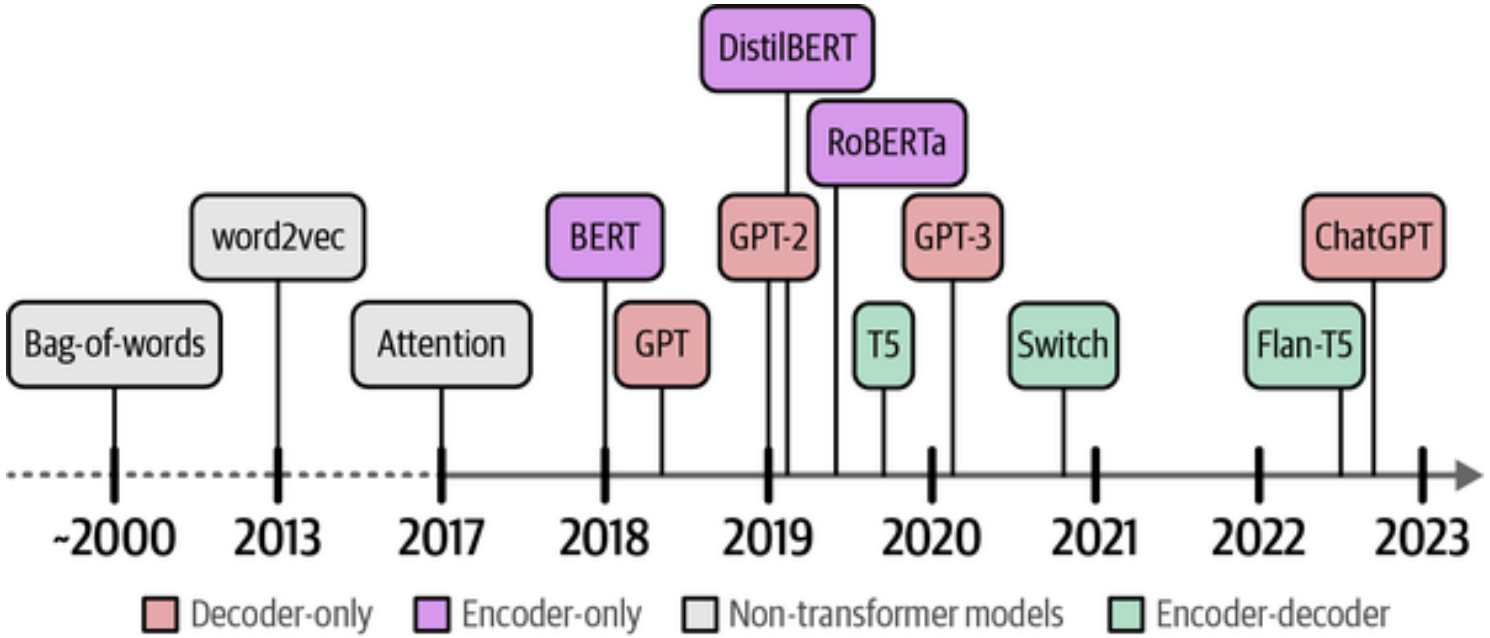
word2vec

Attention

Encoder only

Decoder only

Encoder-decoder



Alammar, J. (2015). *The Illustrated Word2vec*. Github.io. https://jalammar.github.io/illustrated-word2vec/

Limitation of RNNs

Processes tokens **one by one** (sequentially)

❌ **No parallel training**

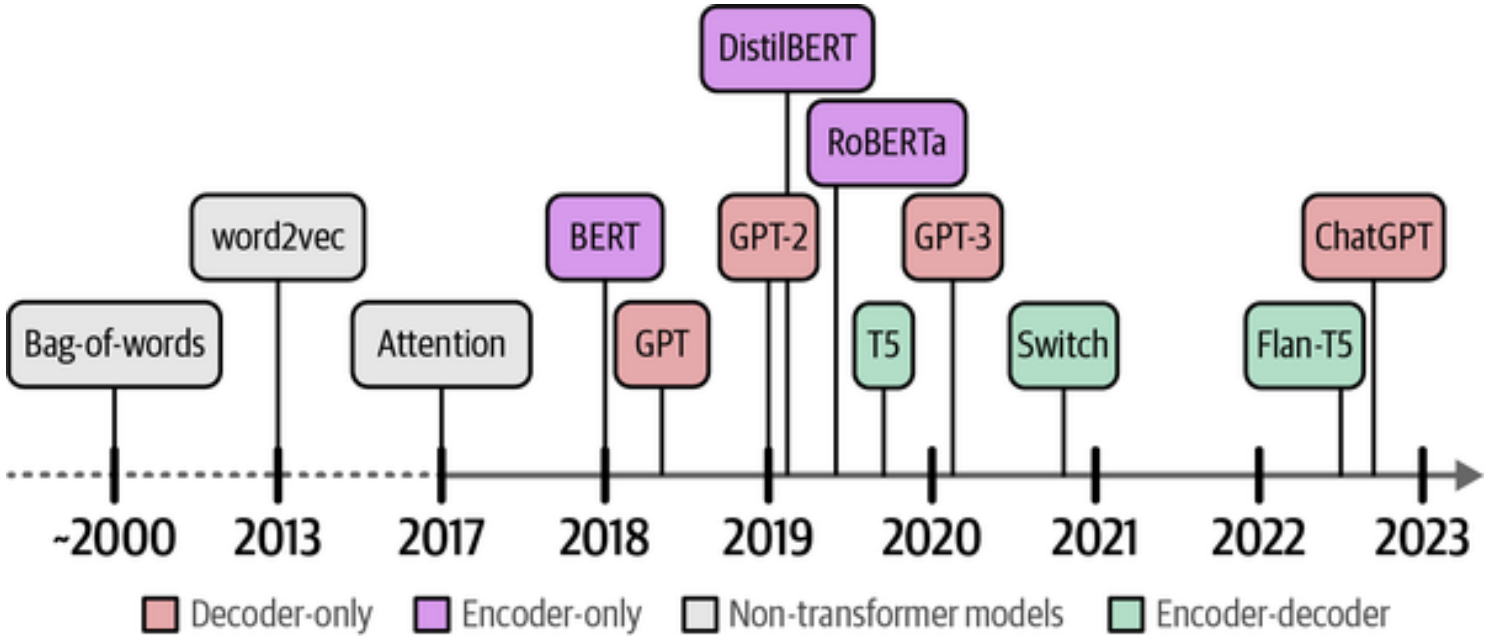
Language AI Evolution

Bag-of-Words (2000)

Word2vec (2013)

Seq2seq with attention (2014)

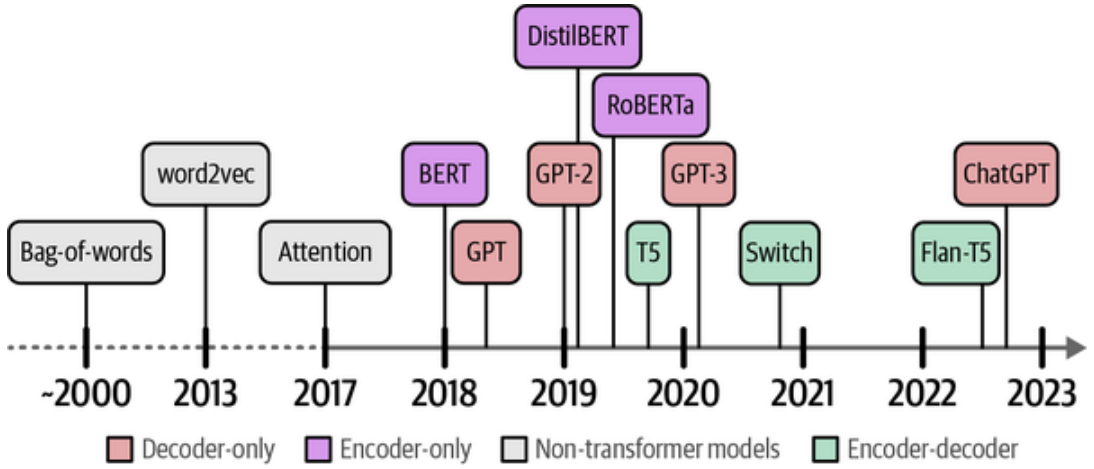
Attention (2017)



Alammar, J. (2015). *The Illustrated Word2vec*. Github.io. https://jalammar.github.io/illustrated-word2vec/

2017 – Emergence of the Transformer Architecture

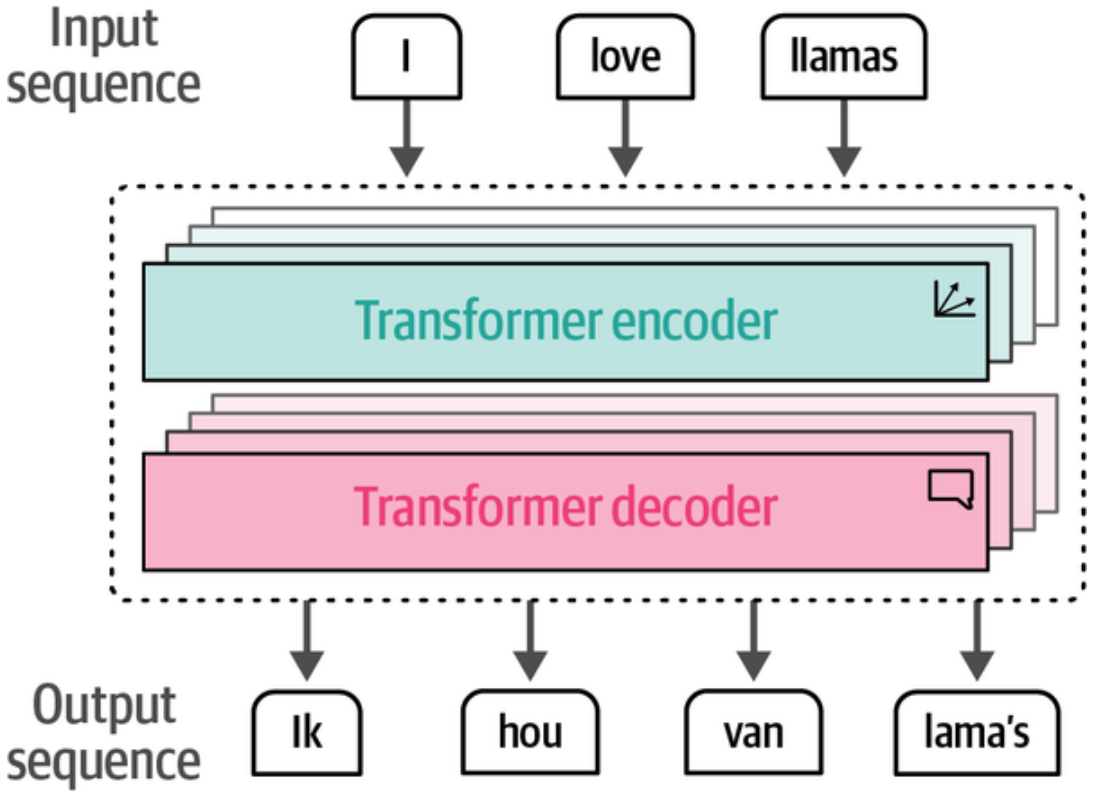
- ✓ **Key Breakthrough**
- ✓ Introduced the **Transformer** architecture
- ✓ **Removed RNNs** entirely — no recurrence
- ✓ Used **only attention mechanisms**
- ✓ Enables **parallel training**
- ✓ Much **faster** than RNNs



Alammar, J. (2015). *The Illustrated Word2vec*. Github.io. https://jalammar.github.io/illustrated-word2vec/

2017 – Emergence of the Transformer Architecture

- ✓ Built with **stacked encoders + decoders**
- ✓ Still **autoregressive** (one word at a time)
- ✓ Input flows through each layer to refine output
- ✓ Powers GPT, BERT, and modern LLMs



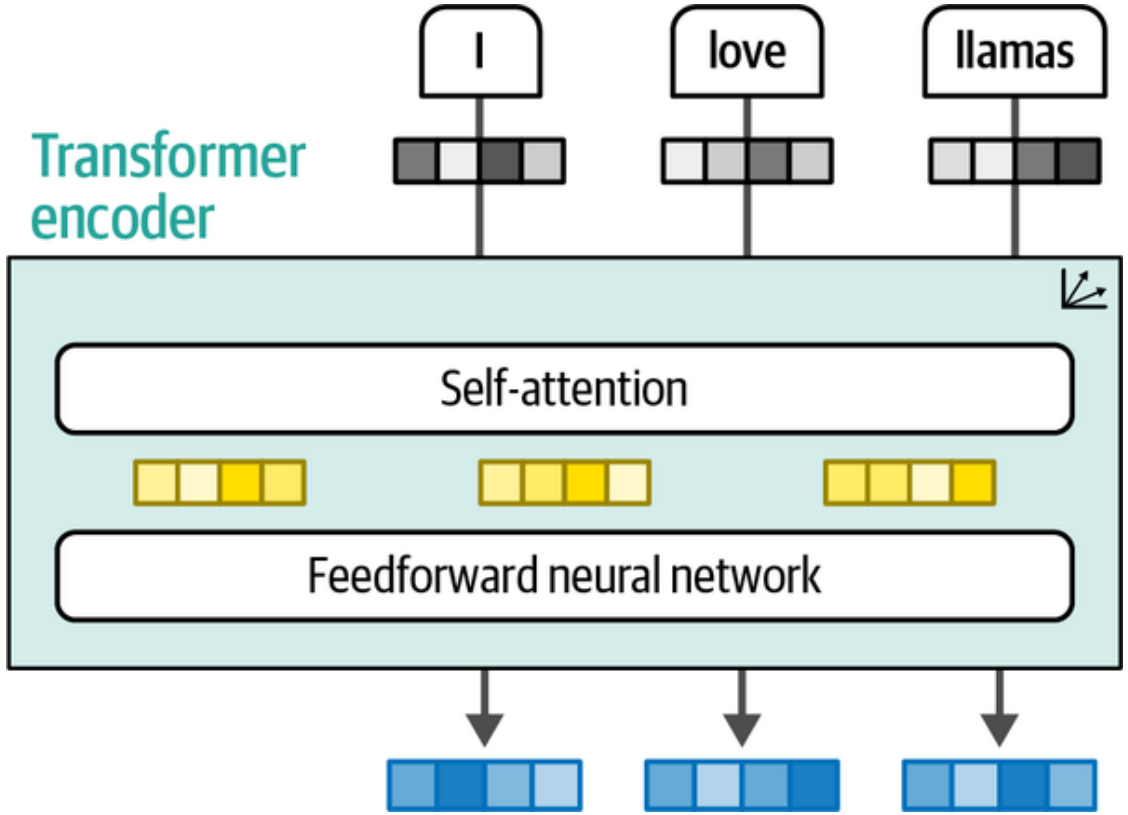
Alammar, J. (2015). *The Illustrated Word2vec*. Github.io. https://jalammar.github.io/illustrated-word2vec/

Encoder

No RNNs — Only Attention

Encoder & decoder use **attention**, not RNNs

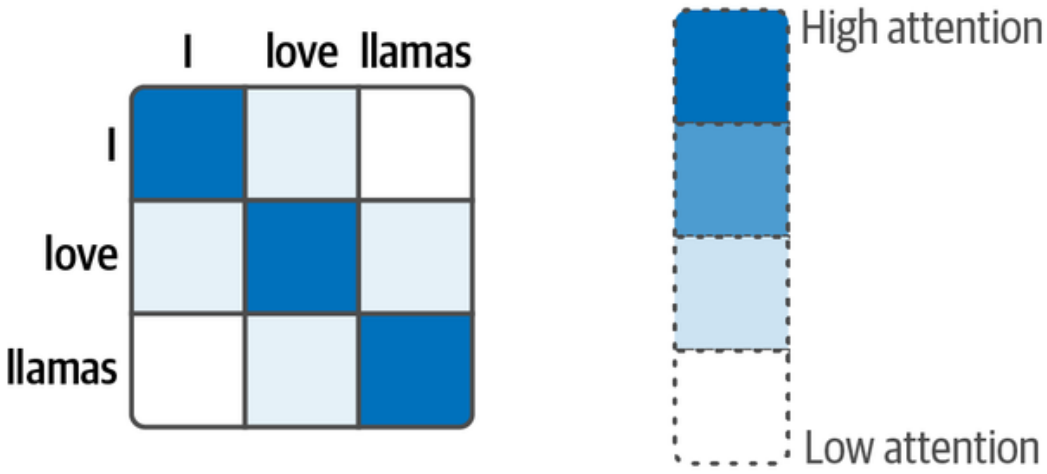
- Fully replaces recurrence with **self-attention**
  → *Each word attends to every other word*
- **Feedforward Neural Network**
  → *Adds non-linearity and depth*



Alammar, J. (2015). *The Illustrated Word2vec*. Github.io. https://jalammar.github.io/illustrated-word2vec/

Self-Attention

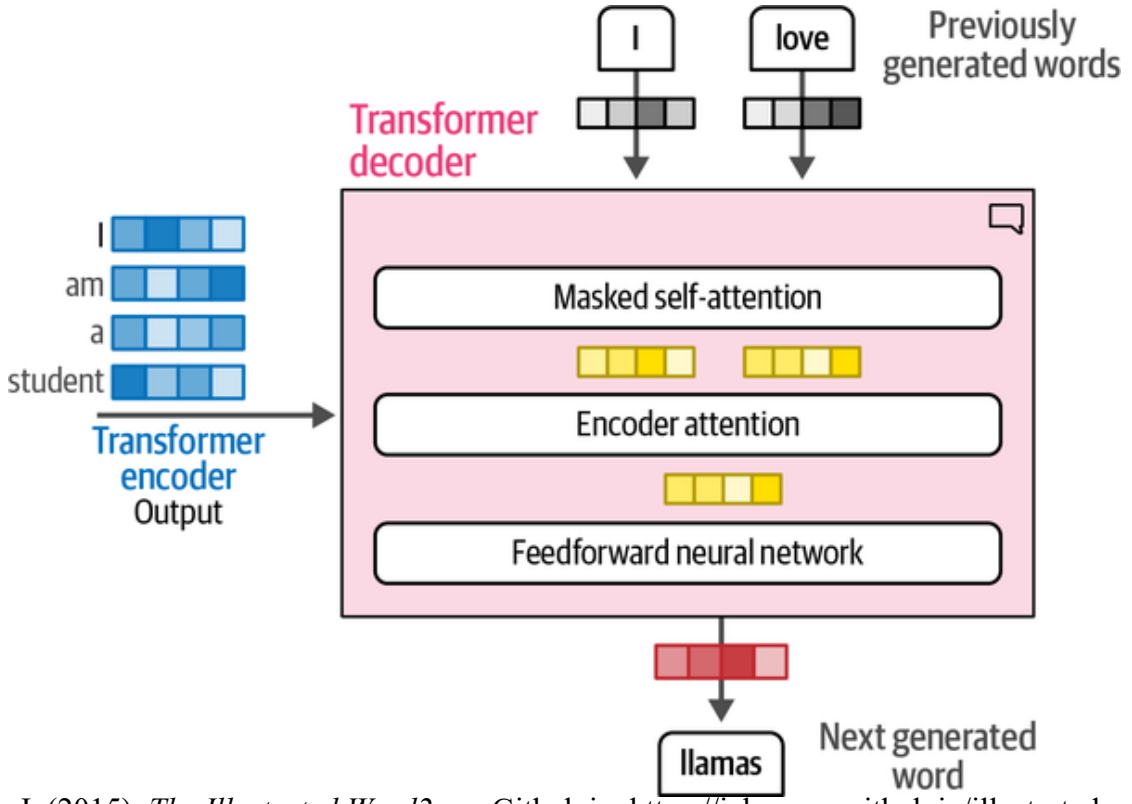
Self-attention attends to all parts of the input sequence so that it can "look" both forward and back in a single sequence.



Alammar, J. (2015). *The Illustrated Word2vec*. Github.io. https://jalammar.github.io/illustrated-word2vec/

Decoder

the decoder has an additional layer that pays attention to the output of the encoder (to find the relevant parts of the input)



Alammar, J. (2015). *The Illustrated Word2vec*. Github.io. https://jalammar.github.io/illustrated-word2vec/

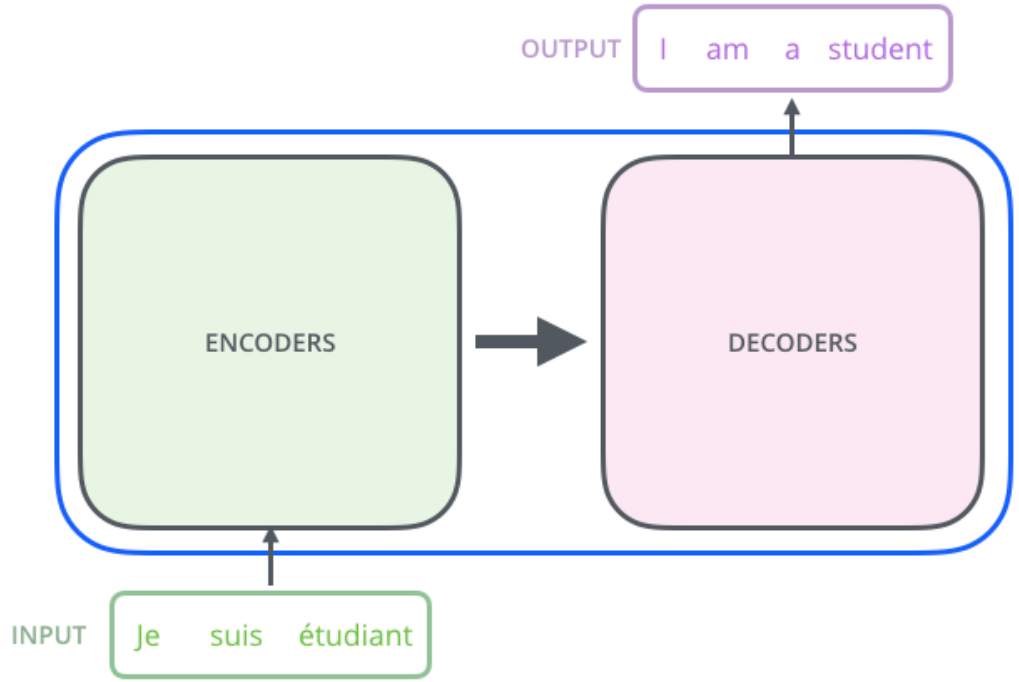
Transformer

- **Black box** model view
- Input: one language; Output: translated sentence

Alammar, J. (2025). *The Illustrated Transformer*. Github.io. https://jalammar.github.io/illustrated-transformer/

Transformer

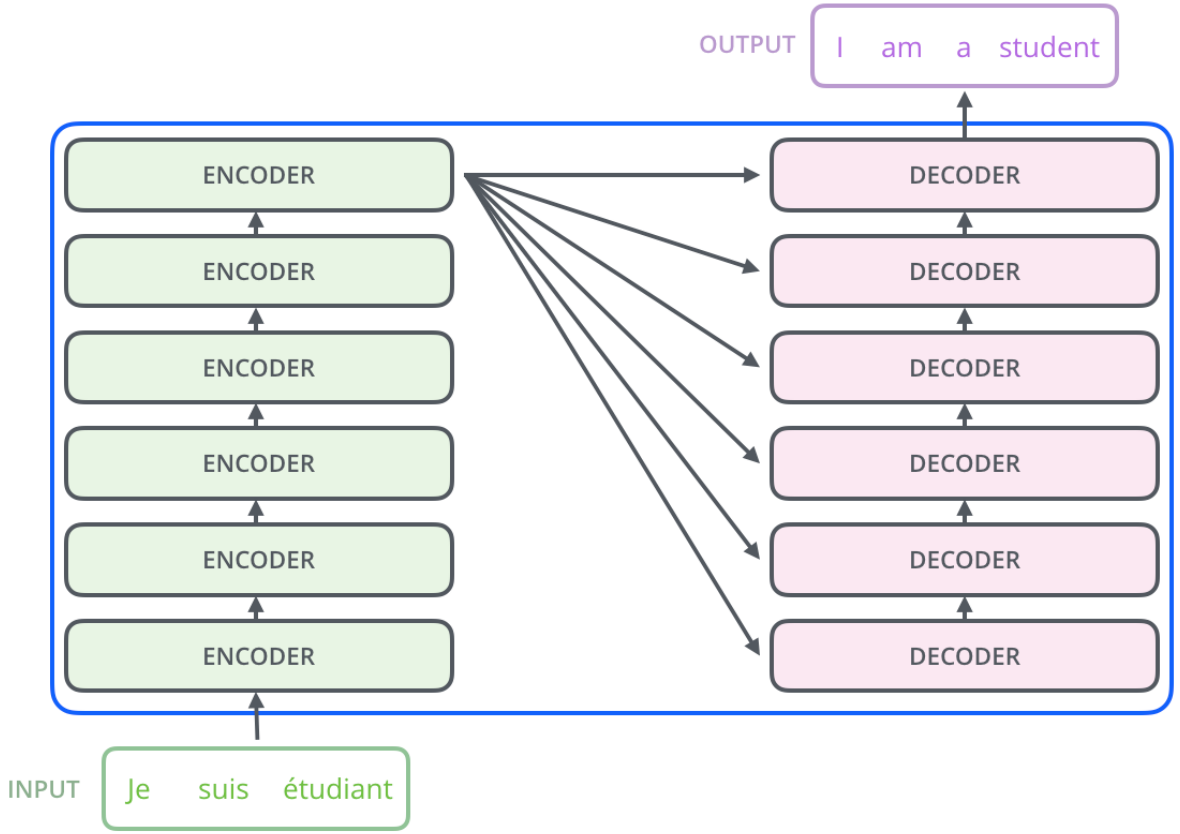
- Inside: **encoder**, **decoder**, and **connections**



Alammar, J. (2025). *The Illustrated Transformer*. Github.io. https://jalammar.github.io/illustrated-transformer/

Transformer

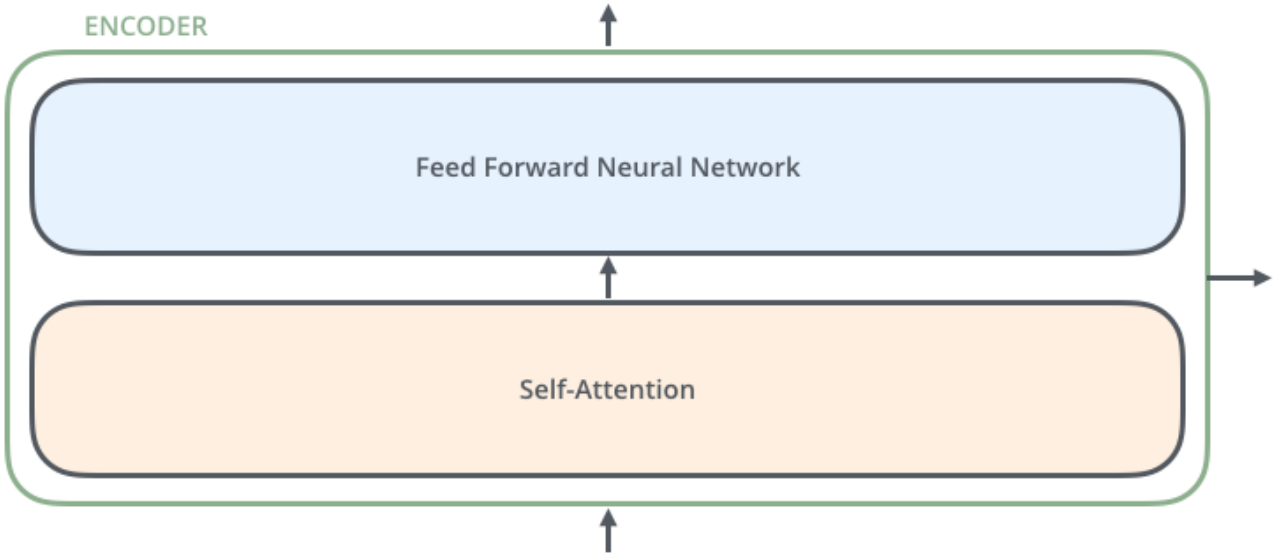
- **Encoder stack**: typically 6 layers (adjustable)
- **Decoder stack**: same number of layers
- Layer count is **flexible**, not fixed



Alammar, J. (2025). *The Illustrated Transformer*. Github.io. https://jalammar.github.io/illustrated-transformer/

Transformer

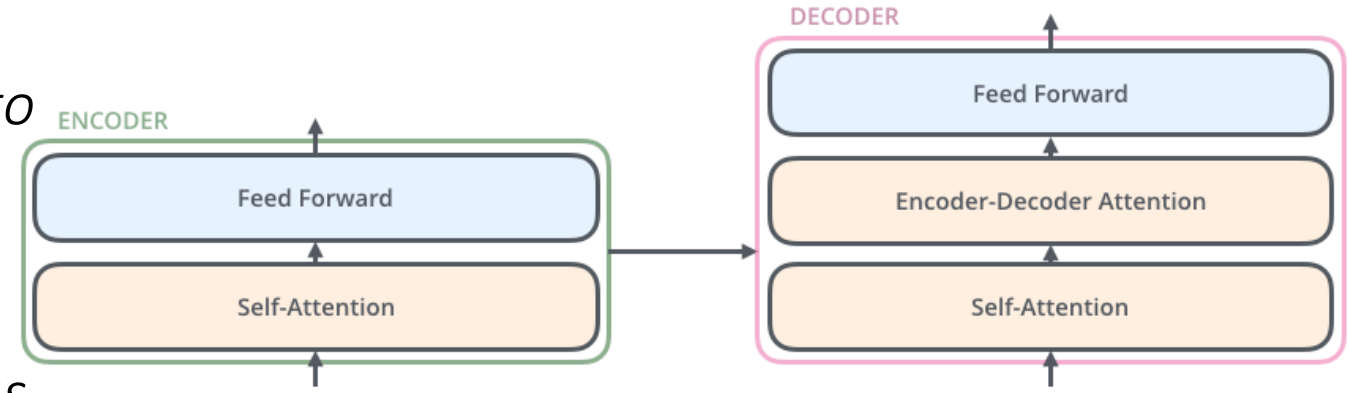
- **Encoders**: identical structure
- **No weight sharing**
- Each has **2 sub-layers**



Alammar, J. (2025). *The Illustrated Transformer*. Github.io. https://jalammar.github.io/illustrated-transformer/

Transformer

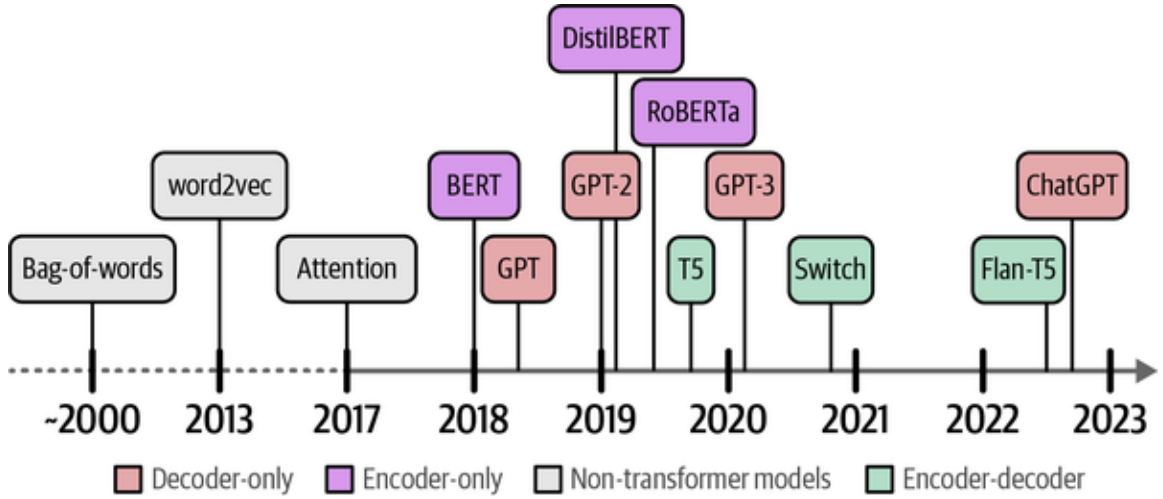
- **Encoder layers**:
  - ✓ ***Self-attention*** *(looks at other input words)*
  - ✓ ***Feed-forward network*** *(applied to each position)*
- **Decoder layers**:
  - ✓ ***Self-attention***
  - ✓ ***Encoder-decoder attention*** *(focus on input sentence)*
  - ✓ ***Feed-forward network***



Alammar, J. (2025). *The Illustrated Transformer*. Github.io. https://jalammar.github.io/illustrated-transformer/

Encoder-Only Transformers

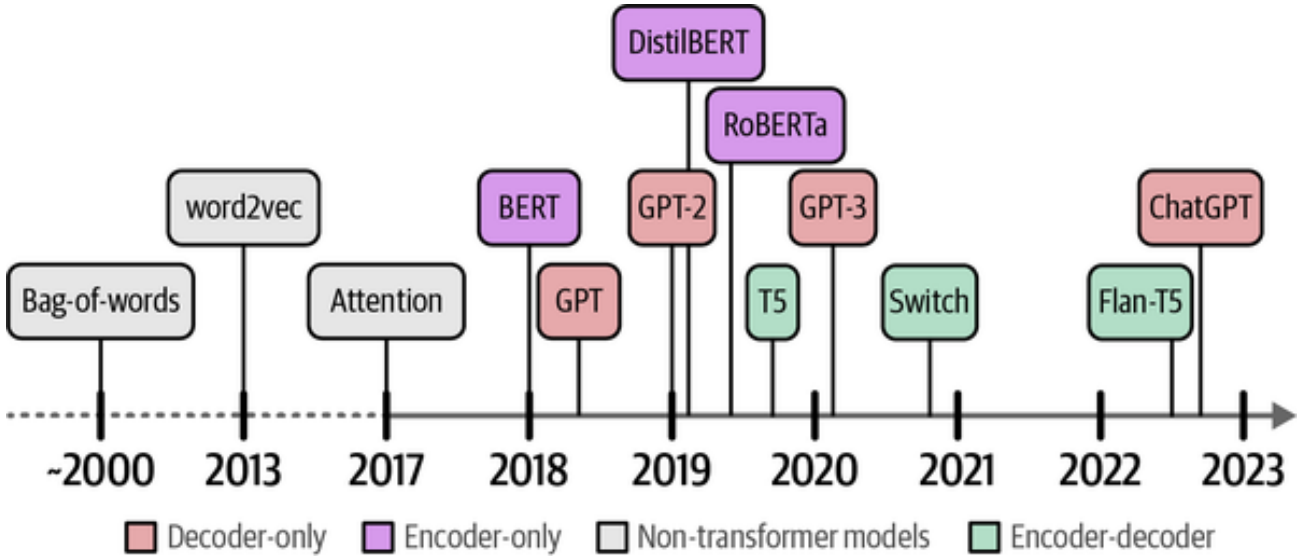
- **Purpose**: Understand and represent input text
- **Architecture**: Transformer encoders without decoders
- **Key Features**:
  - ✓ *Bidirectional self-attention*
  - ✓ *Masked language modeling*
- **Examples**: BERT, RoBERTa
- **Ideal for**: Classification, named entity recognition, sentence embedding



Alammar, J. (2015). *The Illustrated Word2vec*. Github.io. https://jalammar.github.io/illustrated-word2vec/

2018- Decoder-Only Transformers

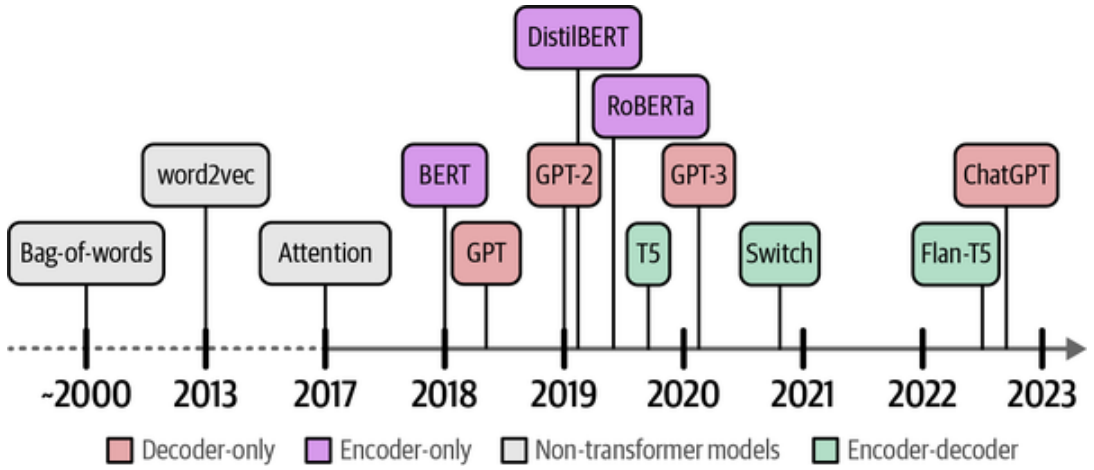
- **Purpose**: Generate text sequentially
- **Architecture**: Transformer decoders without encoders
- **Key Features**:
  - ✓ *Autoregressive generation*
- **Examples**: **Generative Pre-trained Transformer (GPT)** series, LLaMA
- **Ideal for**: Text generation, summarization, translation



Alammar, J. (2015). *The Illustrated Word2vec*. Github.io. https://jalammar.github.io/illustrated-word2vec/

Encoder-Decoder Transformers

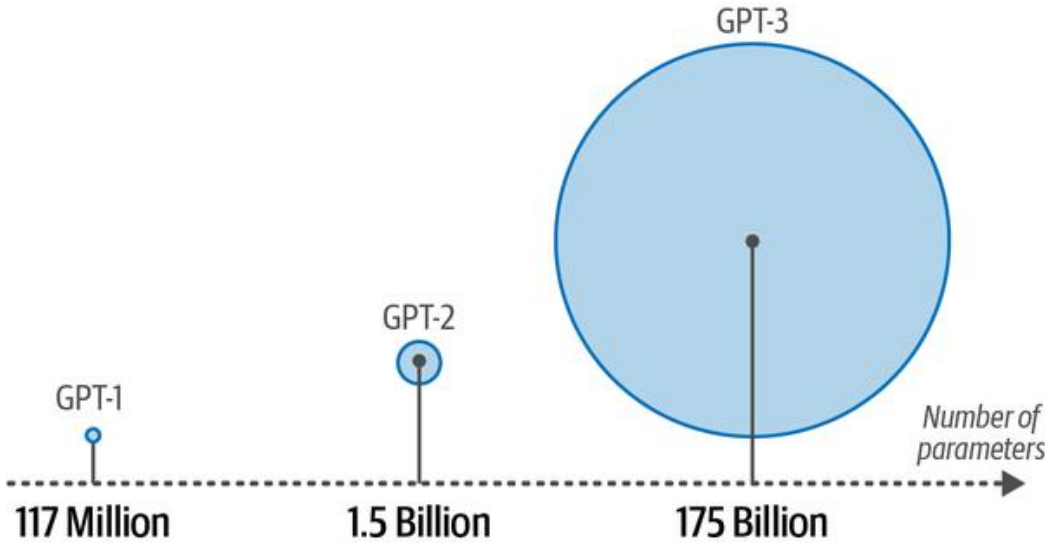
- **Purpose**: Transform input sequences into output sequences (e.g., translation, summarization)
- **Architecture**: Stacked encoder and decoder blocks with attention mechanisms
- **Key Features**:
- Encoder processes the entire input sequence simultaneously
- Decoder generates output tokens one at a time, attending to encoder outputs
- Utilizes self-attention and cross-attention mechanisms
- **Examples**: T5, BART, MarianMT, PEGASUS
- **Ideal for**: Machine translation, text summarization, question answering



Alammar, J. (2015). *The Illustrated Word2vec*. Github.io. https://jalammar.github.io/illustrated-word2vec/

2018 - GPT-1

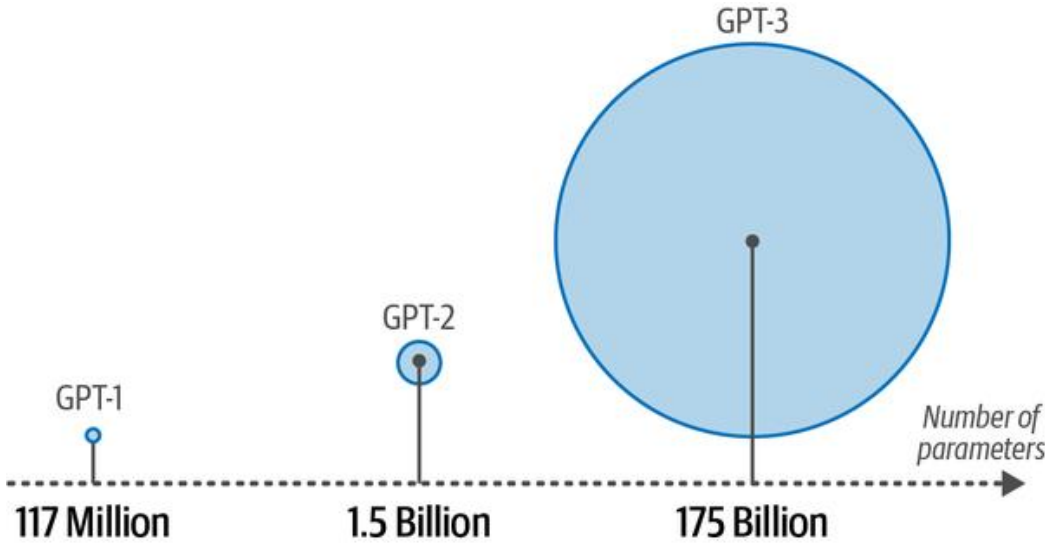
- **Parameters**: 117 million
- **Training Data**: BooksCorpus (7,000 books)
- **Architecture**: 12-layer decoder-only Transformer
- **Significance**: Introduced transformer-based language modeling



Alammar, J. (2015). *The Illustrated Word2vec*. Github.io. https://jalammar.github.io/illustrated-word2vec/

2019 - GPT-2

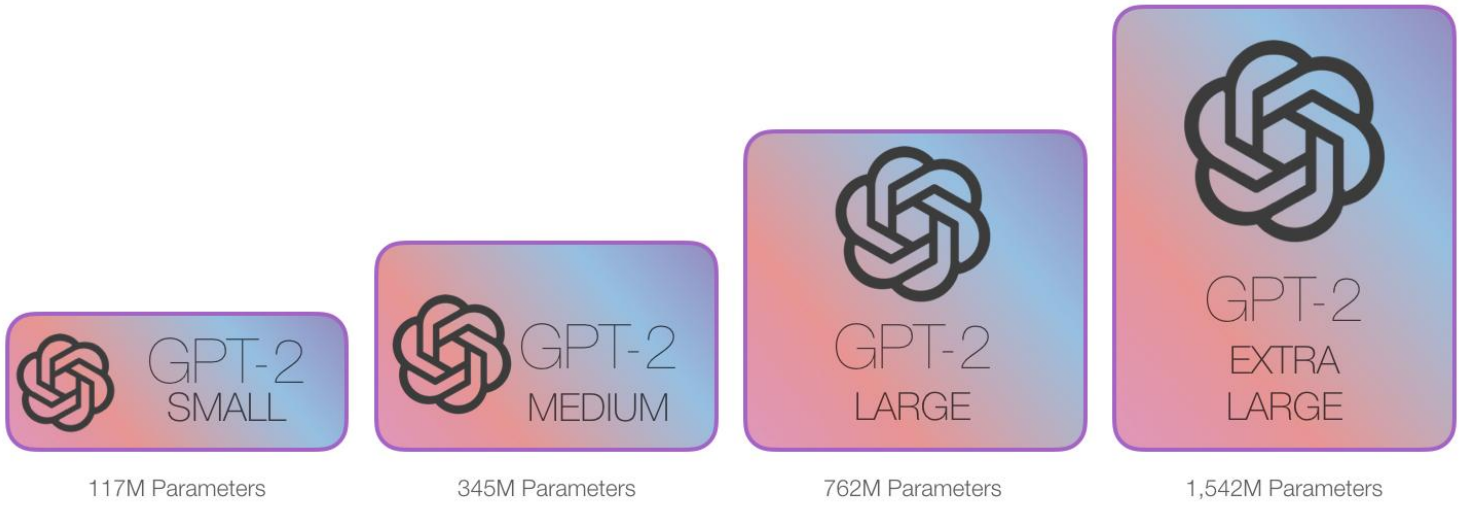
- **Parameters**: 1.5 billion
- **Training Data**: WebText (8 million web pages – 40GB)
- **Enhancements**: Improved coherence and context understanding



Alammar, J. (2015). *The Illustrated Word2vec*. Github.io. https://jalammar.github.io/illustrated-word2vec/

2019 - GPT-2

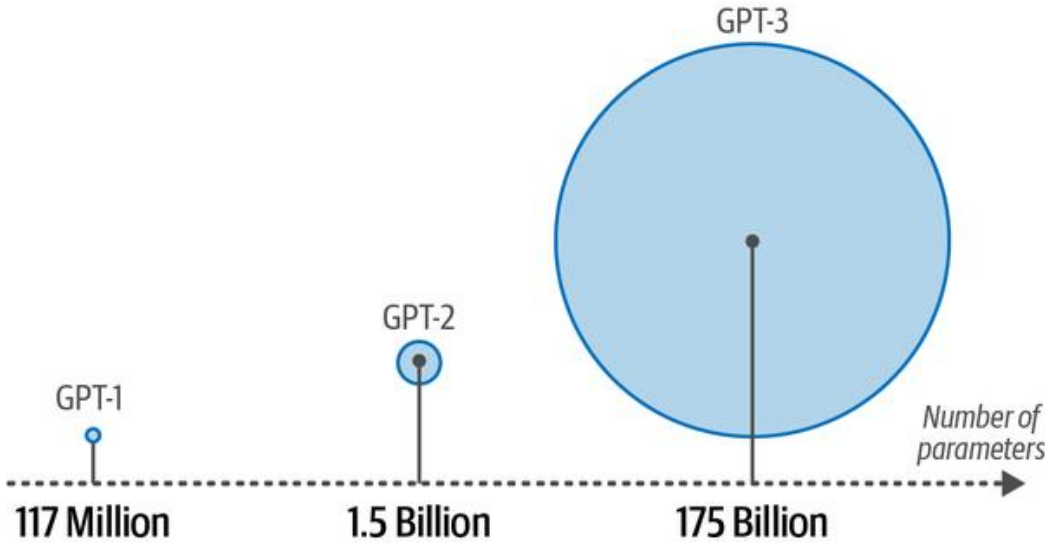
- Compared to **SwiftKey (78MB)**
- **Smallest GPT-2**: ~500MB
- **Largest GPT-2**: ~6.5GB (13× larger)



Alammar, J. (2019). *The Illustrated GPT-2 (Visualizing Transformer Language Models)*. Github.io. https://jalammar.github.io/illustrated-gpt2/

2019 - GPT-3

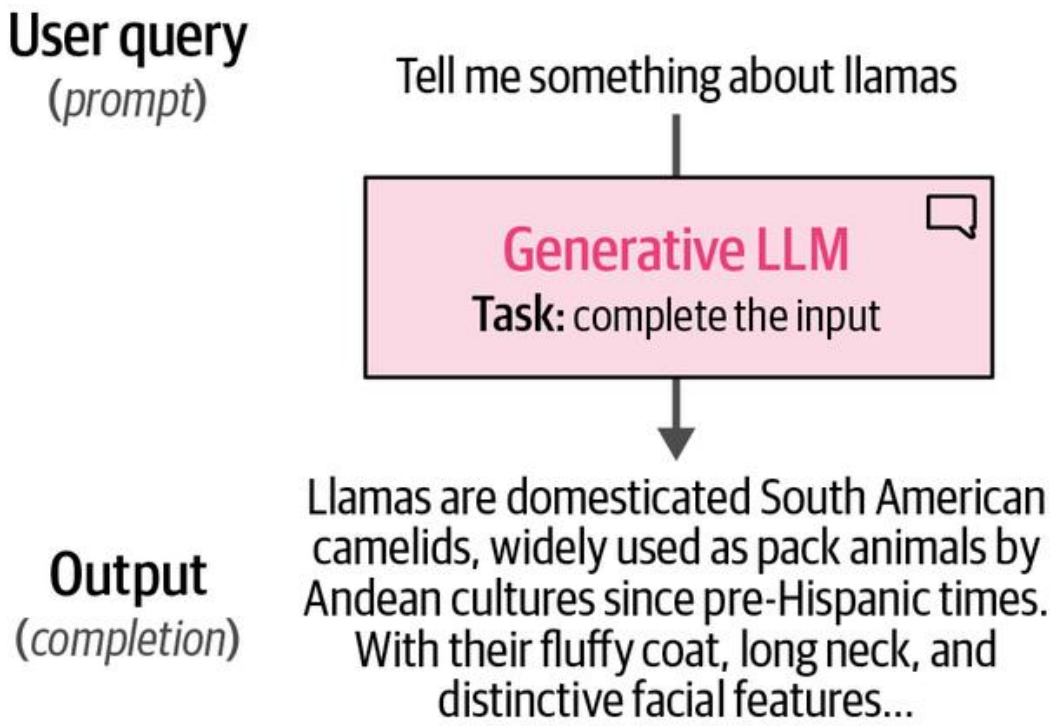
- **Parameters**: 175 billion
- **Training Data**: Common Crawl, WebText, Books, Wikipedia
- **Advancements**: Exhibited few-shot and zero-shot learning capabilities



Alammar, J. (2015). *The Illustrated Word2vec*. Github.io. https://jalammar.github.io/illustrated-word2vec/

Understanding Large Language Models (LLMs)

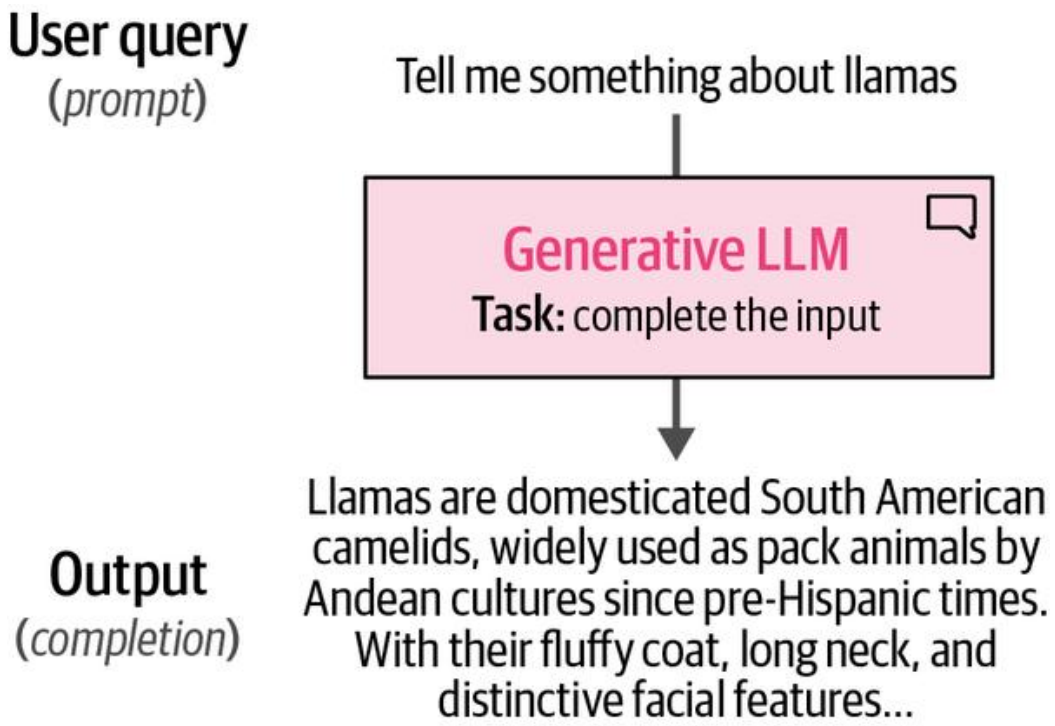
- **LLMs**: Encompass both generative (decoder-only) and representation (encoder-only) models
- **Generative LLMs**: Predict subsequent text based on input sequences
- **Representation LLMs**: Focus on understanding and encoding input text



Alammar, J. (2015). *The Illustrated Word2vec*. Github.io. https://jalammar.github.io/illustrated-word2vec/

Transition to Instruction and Chat Models

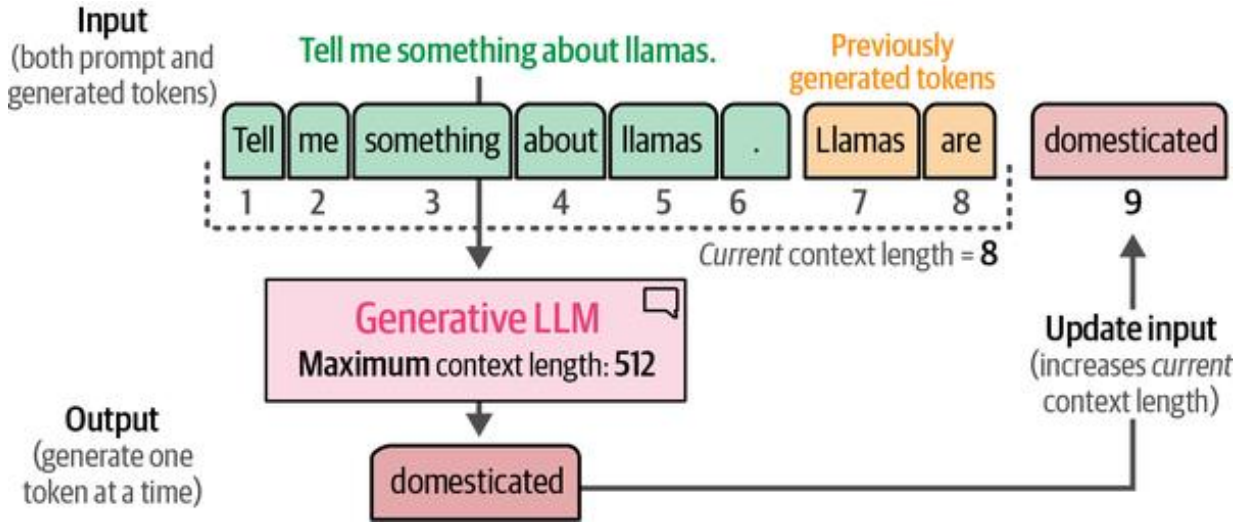
- **Initial Use**: Autocomplete or continue given text sequences
- **Advancement**: Fine-tuned to follow specific instructions and engage in dialogues
- **Outcome**: Development of models capable of answering questions and following user prompts



Alammar, J. (2015). *The Illustrated Word2vec*. Github.io. https://jalammar.github.io/illustrated-word2vec/

Understanding Context Length in LLMs

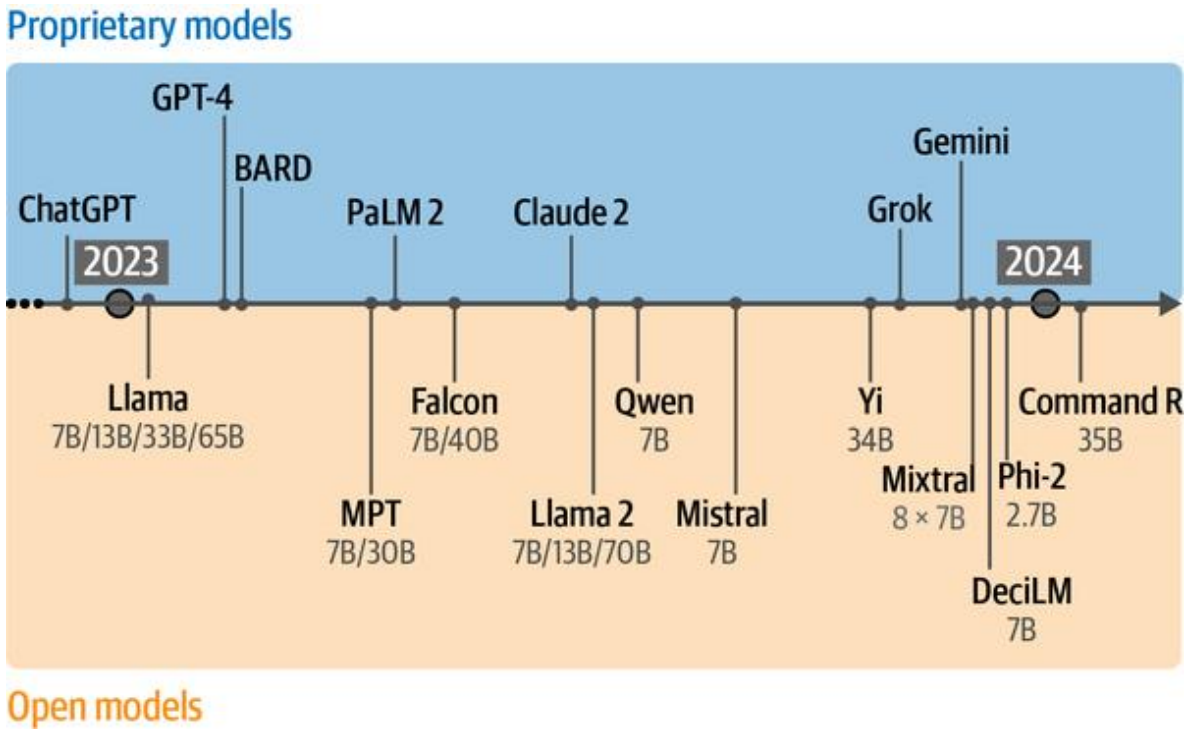
- **Context Length**: The maximum number of tokens an LLM can process at once.
- **Importance**: A larger context window > larger passed documents.
- **Autoregressive Nature**: increase as new tokens are generated.
- **Limitations**: The context length is the maximum context an LLM can handle



Alammar, J. (2015). *The Illustrated Word2vec*. Github.io. https://jalammar.github.io/illustrated-word2vec/

2023-The Year of Generative AI

- Many LLMs released in 2023
- Includes open source & proprietary models
- Foundation models widely used



Alammar, J. (2015). *The Illustrated Word2vec*. Github.io. https://jalammar.github.io/illustrated-word2vec/

Small vs Medium vs Large Models

Model Size	Parameter Range	Examples	Characteristics
Small Models	~33M to 500M	- answerai-colbert-small-v1 (~33M) - DistilBERT (~66M)	- Lightweight and efficient - Suitable for resource-constrained environments - Faster inference times
Medium Models	~1B to 10B	- GPT-2 (1.5B) - GPT-Neo (1.3B & 2.7B)	- Balanced performance and resource needs - Handles more complex tasks - Widely used in applications
Large Models	10B and above	- GPT-3 (175B) - GPT-4 (up to 1.76T est.)	- High performance on diverse tasks - Requires significant computational power - Used in advanced AI systems

What Are Large Language Models That Power Generative AI? (2024). Confidentialmind.com. https://www.confidentialmind.com/post/large-language-models-llms#toc-comparing-proprietary-and-open-source-llm-models

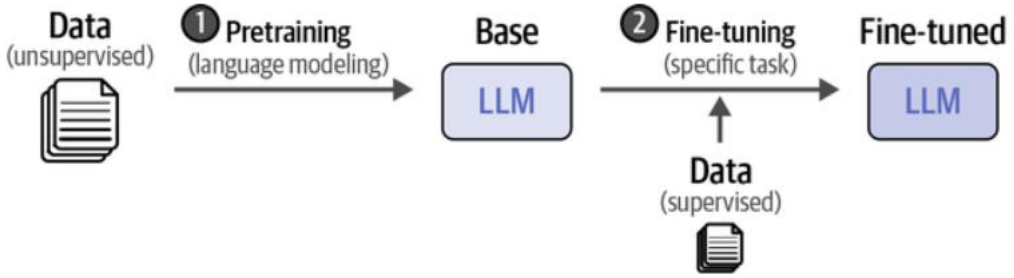
Training Paradigm of Large Language Models

🧠 Traditional Machine Learning:

- *One-Step Process*

🔍 **LLMs: Two-Step Process**

1. *Pretraining (Language Modeling)*
2. *Fine-Tuning (Instruction Tuning or Task Specialization)*



Alammar, J., & Grootendorst, M. (2024). *Hands-on large language models: language understanding and generation*. " O'Reilly Media, Inc.".

Pretraining

Introduction to Pretraining LLMs

- Pretraining is the **foundation** of all large language models (LLMs).
- The model learns by **predicting the next token** in a sequence.
- Trained on **massive text datasets** — books, web data, code, etc.
- This process enables the model to acquire **linguistic and semantic understanding**.

🧠 This training approach is called **language modeling**.

Language Modeling is Self-Supervised

What is Self-Supervised Learning?

The model generates its own labels from input data.

No manual annotation is needed.

Commonly used in LLMs: **next-token prediction**.

🧪 Example:
Input: “The cat sat on the ___”
Model tries to predict: “mat”

🏗️ The Outcome – Base or Foundation Model

What Does Pretraining Produce?

A **Base Model** (also called **Pretrained** or **Foundation Model**)

It has general-purpose knowledge about:

- Language
- Grammar
- Facts
- Reasoning patterns

📦 Think of it as a **raw but powerful engine** — not yet fine-tuned for specific tasks.

Why Base Models Aren’t Enough

- Base models are **powerful but generic**.
- Not optimized for direct interaction or specific use cases (e.g., summarization, QA).
- **End users may find them hard to use** effectively without further adaptation.

💡 **Next Step**: Fine-tuning or instruction-tuning for real-world applications.

Fine-Tuning 1

Making LLMs More Useful with Fine-Tuning

Why Fine-Tuning?

Base models often don't **follow user instructions** naturally.

For example, when asked to write an article, a base model might:

- List steps instead of generating content.
- Misinterpret the prompt.

💡 Users expect **direct and relevant** responses.

What Is Supervised Fine-Tuning (SFT)?

Definition:
Supervised Fine-Tuning (SFT) is the process of adapting a base model to better follow instructions by training it on **instruction-response pairs**.

Still uses **next-token prediction**.

But it's **conditioned on user inputs or prompts**.

Updates base model parameters to align with desired behavior.

🧠 The model becomes more helpful, direct, and focused.

🧪 How SFT Works

Training Process:

- Input: User prompt
- Target: Desired model response
- Objective: Predict the next token based on both input and expected output.

📊 Example:

- Prompt: “Write a poem about the ocean.”
- Target: “The waves crash softly on the shore…”

🔁 This refines the model to follow instructions **more faithfully**.

🧰 Applications of Supervised Fine-Tuning

- Instruction-following (chatbots, agents)
- Text classification
- Summarization
- Translation

📈 SFT is a key step to transform a **general-purpose model** into a **task-specific assistant**.

From Base Model to Instruction Model

- Base model: Trained on broad data, lacks specific instruction-following behavior.
- SFT model: Adapted to **respond to prompts**, improving usability.

From “language model” to **instructional agent**.

Fine-Tuning 2

Preference Tuning

🎯 What Is Preference Tuning?

Definition:
Preference tuning is a fine-tuning step that aligns model outputs with **human preferences** or **AI safety objectives**.

- Refines model behavior further than Supervised Fine-Tuning (SFT).
- Uses human-labeled or preference-based data.
- Helps ensure the model responds in a **desirable, safe, and helpful way**.

🛠️ How Preference Tuning Works

- Involves **ranking** multiple possible outputs for a prompt.
- The model is trained to prefer outputs that:
  - ❑ *Are clearer*
  - ❑ *Are safer*
  - ❑ *Align better with human intent*

🧪 Often powered by **human feedback** on model completions.

🔁 Preference Tuning vs. Supervised Fine-Tuning

Aspect	Supervised Fine-Tuning (SFT)	Preference Tuning
Goal	Learn to follow instructions	Align outputs with preferences
Data Type	Prompt → Desired Output	Output Comparisons or Rankings
Output Style	Direct and relevant responses	Helpful, polite, and safe responses

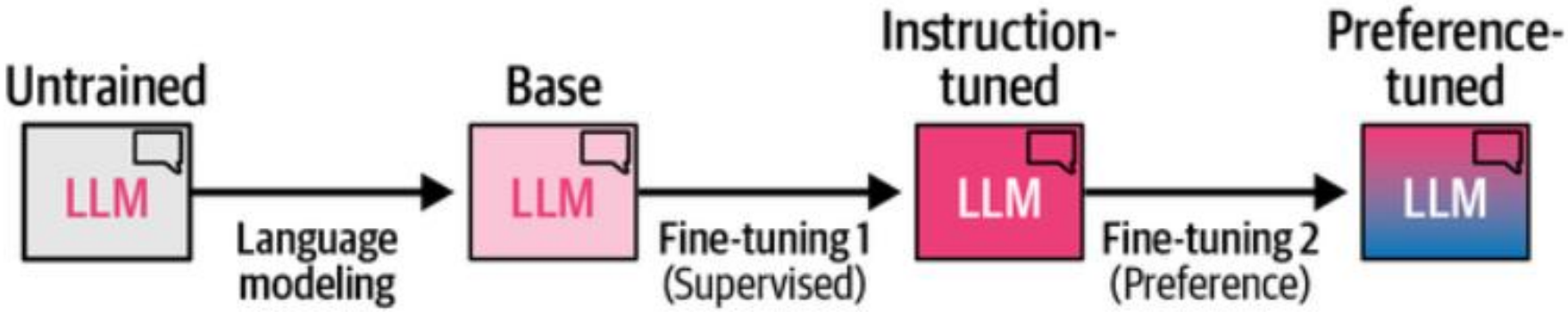
🚀 The Full Training Pipeline

Pretraining: Massive text datasets → Language understanding

SFT: Instruction-response pairs → Task following

Preference Tuning: Ranked outputs → Aligned behavior

📊 These three stages transform a base LLM into a **safe, capable assistant**.



Alammar, J., & Grootendorst, M. (2024). *Hands-on large language models: language understanding and generation*. " O'Reilly Media, Inc.".

Getting Started with Generative Models on Hugging Face

🌐 Hugging Face Model Hub

- Over **800,000 models** available
- Covers:
  - *Large Language Models (LLMs)*
  - *Computer Vision*
  - *Audio Processing*
  - *Tabular Data*
- Most models are **open source** and easy to experiment with

🚀 Starting with a Generative Model

- We begin with **Phi-3-mini**
  - *A 3.8B parameter LLM*
  - *Optimized for generative tasks*
- Suitable for devices with:
  - *< 8 GB VRAM*
  - *< 6 GB VRAM (with quantization)*

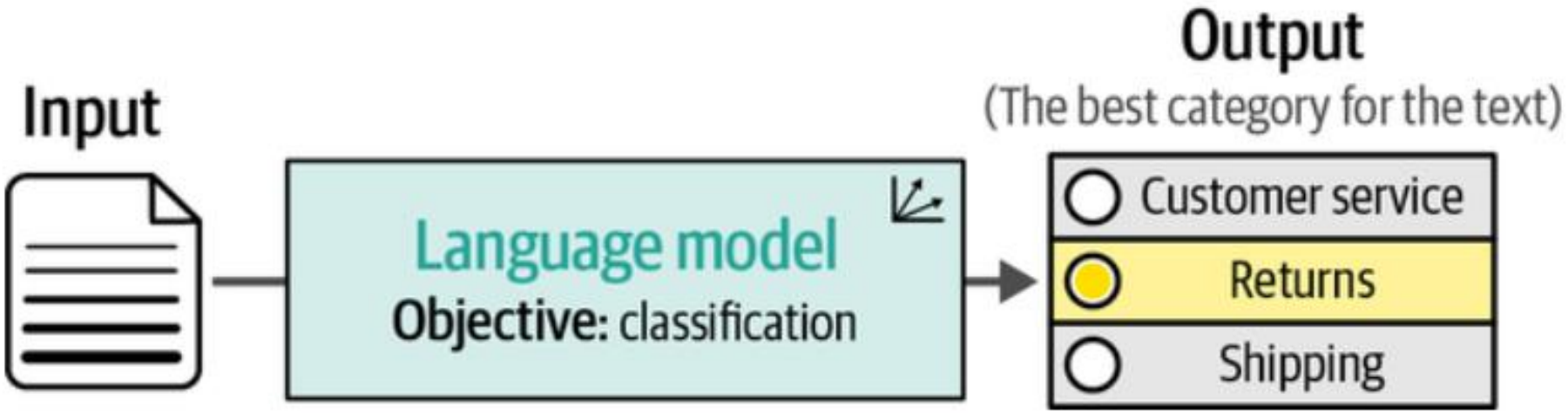
🔓 **Licensing and Use**

Licensed under **MIT License**

- ✔ Free for commercial use
- ✔ No restrictive constraints

Text Classification

Text Classification



Alammar, J., & Grootendorst, M. (2024). *Hands-on large language models: language understanding and generation*. " O'Reilly Media, Inc.".

Text Classification

TEXT CLASSIFICATION WITH REPRESENTATION MODELS

TEXT CLASSIFICATION WITH GENERATIVE MODELS

TEXT CLASSIFICATION WITH REPRESENTATION MODELS

What Are Representation Models?

- Also called *encoder-based models*.
- Output dense **vector representations** (embeddings) of text.
- Do **not** generate text — focus on understanding.
- Examples: **BERT**, **RoBERTa**, **DistilBERT**.

Task-Specific Classification

- Pretrained model + classification head (e.g., softmax layer).
- Fine-tuned end-to-end on a specific dataset.
- Suited for: Sentiment analysis, intent detection, etc.

```
from transformers import
BertForSequenceClassification
```

✅ Pros: High accuracy, fast inference
⚠️ Cons: Needs labeled data per task

Embedding-Based Classification

- Use embeddings from models like **Sentence-BERT**, **E5**, or **BGE**.
- Apply **cosine similarity**, **k-NN**, or **clustering** for classification.
- Useful for **few-shot**, **retrieval**, or **multi-label** tasks.

✅ Pros: Flexible, task-agnostic
⚠️ Cons: May need external classifier

🧪 **Example Flow**:

1. Encode labeled examples
2. Encode test sample
3. Match nearest label via similarity

TEXT CLASSIFICATION WITH GENERATIVE MODELS

What Are Generative Models?

- Decoder-only or encoder-decoder models.
- Trained to **generate tokens**, not just embed text.
- Examples: **GPT**, **LLaMA**, **Phi-3**, **Claude**, **Gemini**

🧠 But: They can **perform classification** with clever prompting.

Prompt-Based Classification

- Use instructions to guide output to be one of the known classes.

✅ Pros: No training needed (zero-shot)
⚠️ Cons: Prone to hallucination, slower

🧪 **Example Prompt**:

- *"Classify the following review as Positive or Negative: 'The movie was amazing!'"*
- *Output: Positive*

Open Source Generative Models

- Examples: **LLaMA 3**, **Phi-3**, **Mistral**, **GPT-J**
- Run locally or in private cloud
- Customizable and transparent

🛠️ Tools: HuggingFace Transformers, LangChain, vLLM

Prompting

Prompting AI for Everyday Problem Solving

Harnessing the power of language models to save time, spark creativity, and make better decisions.

What Is Prompting?

- Prompting is how we ask AI to do something.
- Like giving instructions to a very smart assistant.
- The clearer and more specific you are, the better the results.

What Can AI Help You With?

Area	Example Prompts
Writing	"Summarize this article in 3 bullet points."
Planning	"Help me plan a healthy 3-day meal plan."
Brainstorming	"Give me creative birthday party ideas."
Learning	"Explain quantum computing like I’m 12."
Productivity	"Draft a polite follow-up email."

Prompt Example – Writing a Letter

Prompt: “Write a formal complaint email to my internet provider about frequent outages.”
AI Output: A ready-to-use draft with tone and structure matching a business email.
🔁 You can refine: “Make it more assertive.” / “Add specific dates.”

Iteration Makes It Better

- AI is not a one-shot tool—think of it as a conversation.
- Improve outputs by refining your prompt:
- “Make it shorter.”
- “Add humor.”
- “Use simpler language.”

Assigning Roles to the AI

Role Prompt	Effect
"Act as a nutritionist"	Gets advice grounded in dietary science
"Pretend you're a travel agent"	Suggests structured itineraries
"You're my personal assistant"	Helps prioritize tasks logically

Example – Learning Something New

Prompt: "Explain inflation like I’m a college freshman studying economics."
✨ Result: A simplified explanation with analogies and relevant examples.
🔁 Follow-up: “Can you add a visual metaphor?”

Prompting Best Practices

- Be specific: Who, what, when, how?
- Give context: “I’m a university lecturer writing a syllabus...”
- Assign a role: “You are a professional career coach...”
- Iterate: Don’t be afraid to ask again with tweaks.

Activity – Try It Yourself!

- Use ChatGPT or similar AI.
- Prompt: “You are my life coach. Give me 3 tips to reduce stress at work.”
- Refine it: “Make it data-backed” or “Add one book recommendation.”

Conclusion

AI is powerful when you know how to talk to it.

You don’t need to be technical to benefit.

Your clarity = better results.

Be curious, creative, and conversational.

Introduction to Prompt Engineering

Goal: Learn how to use GPT-4o (or any LLM) to automate, improve, and debug code via prompting.

LLMs can accelerate your development process.

Effective prompting is a key skill.

Writing a Basic Function

Prompt Example: "Can you write a Python function to add two numbers named a and b and return the result?"

- GPT outputs a basic function add_two_numbers(a, b).
- Includes an example usage.
- Try the same in JavaScript and C# with minor prompt tweaks.

Domain Knowledge Matters

Prompts require specificity, but domain expertise matters more.

Your knowledge of the language, library, or API enhances the prompt.

LLMs support coding, they don’t replace developers.

Refining Prompts with Libraries

Prompt Example: "Write a Python function using NumPy to add two arrays."

- GPT includes `import numpy as np` and uses `np.add`.
- Function has input and usage example.

Iterative Prompting for Code Quality

Prompt Example: Write a function to check if a number is prime.

GPT outputs a basic prime checker.

You follow up: "Add input validation for positive integers."

Model refines function with error handling.

Importance of Clear Prompts

Bad Prompt: "Make a function." **Good Prompt:** "Create a Python function that returns the area of a circle from a given radius, with error handling."

More context = better code.

Prompting for Web API with Flask

Prompt Example: "Create a Flask API with endpoint /multiply that multiplies two integers."
- GPT writes complete Flask app.
- Includes validation, error handling, and examples.

Prompting for Debugging

Prompt Example: "There is a bug in this code. Can you find it?"

GPT spots divide-by-zero.

Suggests fix and refactors code.

Prompting Best Practices

Be **specific**.

Assign a **role**.

Provide **context**.

Iterate and **give feedback**.

Assigning Roles to GPT

Examples:

- As a Python tutor
- As a security expert
- As a code reviewer

Each role changes:

- Tone
- Detail level
- Suggestions given

Combining Roles

Example: Ask GPT to review a script as both:

A **software architect**

A **security expert**

GPT gives structured critique for both performance and security.

Use Case: Software Tester Role

Goal: Find edge cases.

- Prompt GPT as a software tester.
- GPT suggests test cases, adds robustness.

Advanced Use: Project Integration

- Use LLMs in every stage:
  - Planning
  - Development
  - Testing
  - Documentation

Prompting Summary

Best Practices:

Be specific.

Use roles.

Request expertise.

Iterate with feedback.

Developer in the Loop

Always **test** AI-generated code.

LLMs are powerful, but you're the context expert.

Use LLMs as assistants, not replacements.

Your Turn (Assignment Idea)

Try This: "Write a Python function that validates emails. Ask GPT to add regex validation, handle exceptions, and comment code."

Then, refine your prompt to improve it.

Ethics & Responsible AI

Responsible LLM Development and Usage

Understanding the ethical and practical considerations in building and using language models.

Why Responsible LLM Use Matters

LLMs have widespread impact across industries.

Their capabilities come with serious societal implications.

Developers and users must consider ethical responsibilities.

Bias and Fairness

LLMs learn from vast datasets, many of which contain biases.

These biases can be reproduced and amplified by the model.

Transparency about training data is often limited.

Always test for unintended bias in outputs.

Risk of Harmful Content

LLMs can confidently generate incorrect or misleading information.

Risk of misuse: fake news, spam, misinformation.

Requires responsible prompting and strong content filters.

Intellectual Property Questions

Who owns the output of an LLM?

- The user?
- The model developer?
- The original author of the training data?

Lack of training data transparency complicates legal issues

Regulation in Progress

Governments are actively proposing and passing AI laws.

Example: European Union's AI Act.

Regulation will affect development, deployment, and commercial use.

Important to stay informed and compliant.

Ethical Development Practices

Use diverse and inclusive training data.

Monitor and evaluate model behavior.

Include ethical reviews in development cycles.

Communicate limitations and risks to users.

Abdulkarim M. Jamal Kanaan

Linked-In: a-kanaan

abdulkarim@utar.edu.my

a.kanaan@msn.com

017-2290877

Assistant Professor, GT

Research: R&D, Machine Learning, Deep Learning, Technopreneurship, Management Information System

Training: Visualization, Artificial Intelligence

System Architect: .NET Framework – Windows Application, Web Application

Programming Skills: C# & Python