Chapter 1 - Introduction to Large Language Models

Principles and Mechanisms

Murat Yıldızoğlu
Bordeaux School of Economics
University of Bordeaux
UMR CNRS-INRAE 6060
https://yildizoglu.fr

Master 1, IREF-ERDS

What are Large Language Models?

- **Large Language Models** (LLMs, Brown *et al.* (2020)) are a transformative technology in artificial intelligence (AI) that enable machines to
  - process,
  - understand,
  - translate, and
  - generate human language.
- They are
  - based on *deep learning techniques* and
  - are trained on massive datasets.
- They revolutionized the field of natural language processing (**NLP**) and artificial "intelligence" (**AI**).
- Examples: ChatGPT, Claude, Gemini, Llama, Mistral (Cocorico!) and many others...
- In economics, LLMs have applications such as analyzing financial reports, forecasting economic trends, and simulating policy impacts.

Why Language?

(. . .) humans can generally perform **a new language task** *from only a few examples or from simple instructions – something which current NLP systems still largely struggle to do.*
(Brown et al., 2020, Abstract)

Why Large?

(. . .) [Here] we train GPT-3, an autoregressive language model with **175 billion parameters,** *10x more than any previous non-sparse language model, and test its performance in the few-shot setting.*
(Brown et al., 2020, Abstract)

For which tasks and performance ... and dangers?

(...) GPT-3 achieves **strong performance on many NLP datasets**, *including translation, question-answering, and cloze tasks, as well as several tasks that require on-the-fly reasoning or domain adaptation, such as unscrambling words, using a novel word in a sentence, or performing 3-digit arithmetic.*
(...) Finally, we find that GPT-3 can generate samples of news articles which **human** *evaluators* **have difficulty distinguishing** *from articles written by humans.*
(Brown et al., 2020, Abstract)

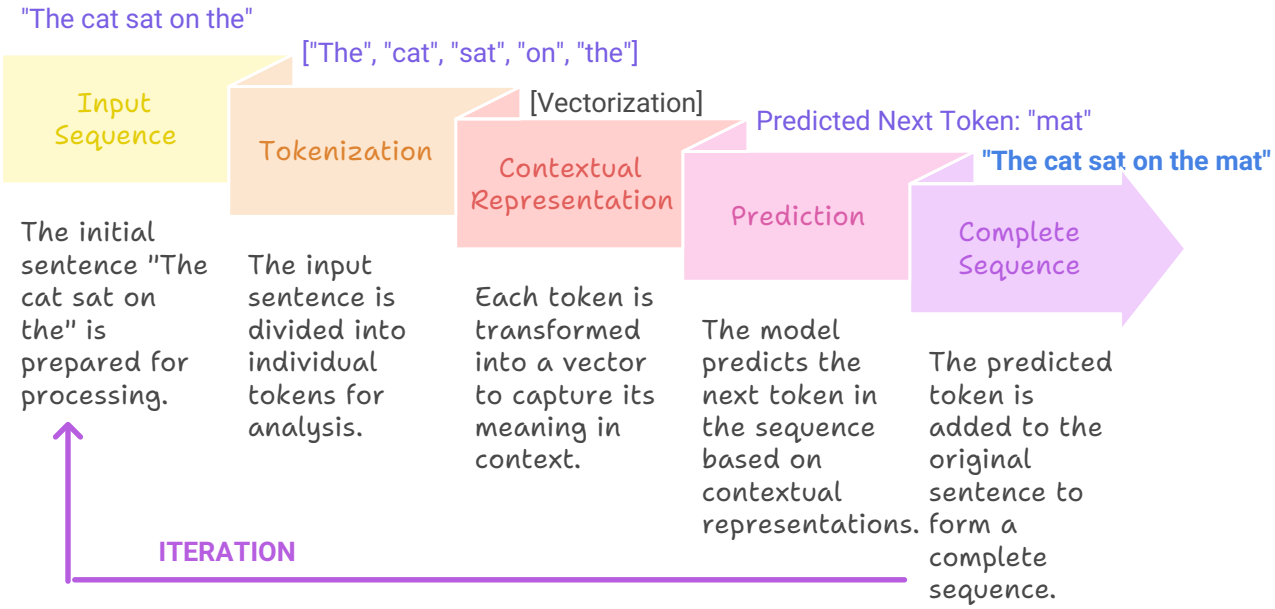
Through which principles?

- The effectiveness of LLMs comes from their ability to generalize knowledge from training data and apply it to new tasks.
- LLMs rely on **sequence modeling**.
- Given a sequence of **tokens** $X = (x_1, x_2, \ldots, x_T)$, the goal is to model the **probability** of the sequence:

$$P(X) = \prod_{t=1}^{T} P(x_t \mid x_1, x_2, \ldots, x_{t-1}),$$

where x_t represents the current token, and $P(x_t \mid x_1, \ldots, x_{t-1})$ is the conditional probability of the token given its preceding context.

Sequence modeling in LLMs



A possible economic application

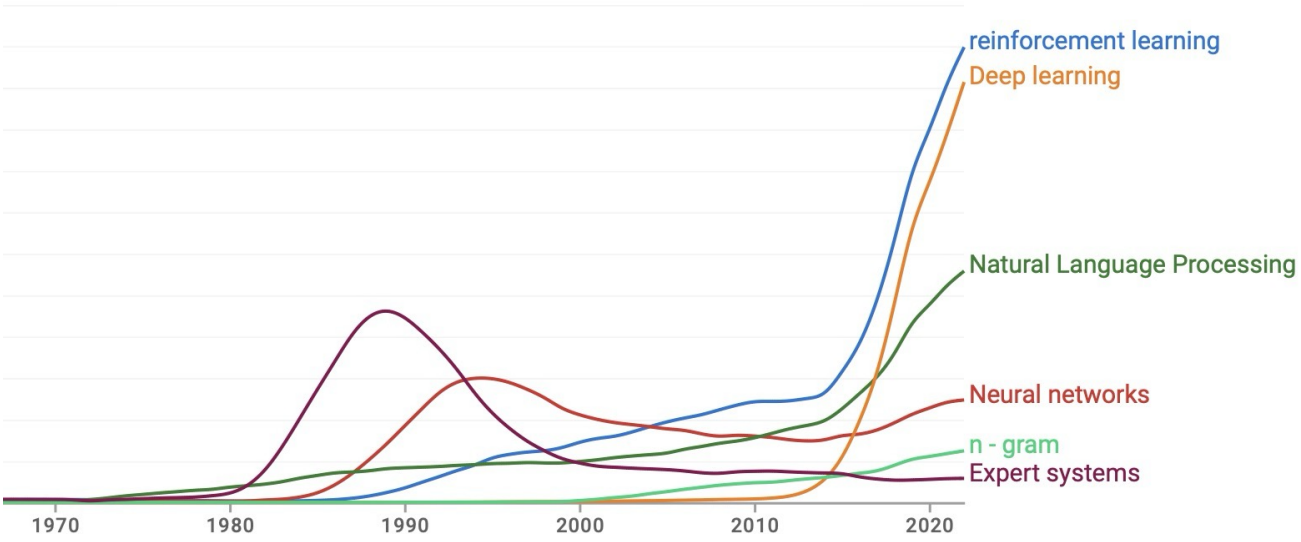
- Imagine you are tasked with analyzing the fiscal policies of multiple countries.
- Using an LLM, you can input detailed policy reports, and the model can summarize key insights, such as GDP projections or budget deficits.
- This saves time for economists who would otherwise manually sift through hundreds of pages of text.

Historical development

The development of language models has evolved through several stages:

- **1950s-1980s: Rule-based** systems used predefined, handcrafted rules to process language. These systems were limited in flexibility and failed to handle ambiguous language.
- **1980s-2000s: Statistical methods**, such as n-gram models, used probabilities to predict text. They improved performance but struggled with long-term dependencies.
- **2000s-2010s: Neural network**s introduced vector representations of words (e.g., Word2Vec, Mikolov *et al.* 2013b), capturing semantic relationships between words. Deep learning (**DL**) by multilayer convolutional neural networks (**CNV**) (LeCun *et al.*, 2015) has the ability to discover intricate structure in large data sets.
- **Note:** Word2Vec captures semantic and syntactic relationships between words by analyzing their co-occurrence patterns in a large corpus of text.
- **2010s-present: Transformers** (Vaswani *et al.*, 2017) revolutionized NLP by enabling models to process entire sequences of text efficiently and understand context better.

Google n-gram: References to technics



Source: **GoogleBooks Ngram Viewer**

Learning, learning, learning...

- As we see, the fundamental dimensions of artificial intelligence systems, including the LLMs are
  - Learning from data (machine learning and training),
  - Discovering patterns in the data (pattern recognition),
  - And ability to recognize patterns in new observations (generalization).
- Different methods have been developed for the learning process.
- Neural networks have become the main model for implementing these three dimensions.
- **Disclosure.** During the preparation of this course various versions of many LLMs have been mistreated: ChatGPT, Claude, Perplexity, Napkin, etc.

Outline

Machine Learning Basics

- Machine learning (**ML**) is the foundation of LLMs, enabling models to learn patterns from data.
- There are three core paradigms:
  - **Supervised Learning:** Models learn from labeled data, where inputs are paired with outputs.
  - **Unsupervised Learning:** Models find patterns in unlabeled data, such as clustering or dimensionality reduction.
  - **Reinforcement Learning:** Models learn through trial and error, receiving feedback to improve performance (Sutton & Barto, 2018).
  - **Semi-supervised Learning:** It combines elements of supervised learning and unsupervised learning. It is particularly useful when labeled data is scarce or expensive to obtain. It leverages the small amount of labeled data to guide the learning process while utilizing the unlabeled data to improve the model's performance.

Example of supervised learning

- In economics, supervised learning can train models to predict GDP growth based on historical data.
- For instance, features such as government spending, inflation rates, and interest rates serve as **inputs** (x),
- while GDP growth rates are the labeled **outputs** y.
- The model learns the relationships (parameters θ of the model) during training and can predict future GDP growth for new economic conditions.
- In supervised learning, the loss function measures the error between predictions and true labels:

$$\mathcal{L}(\theta) = \frac{1}{N} \sum_{i=1}^{N} \ell(f(x_i; \theta), y_i),$$

  where ℓ is the loss function (e.g., mean squared error), $f(x_i; \theta)$ is the model prediction, and y_i is the observed value.
- Neural networks are the perfect candidate for learning a flexible relationship between these inputs and outputs.

Reinforcement learning - 1

- Reinforcement Learning (RL) is a machine learning paradigm where an **agent** learns to make decisions by interacting with an **environment**.
- The agent's goal is to maximize the cumulative reward it receives over time.
- RL is modeled as a sequential decision-making process and is often represented using a **Markov Decision Process (MDP)**.

RL - 2: Key components

- **Agent**: The decision-maker that learns and takes actions.
- **Environment**: The system or world the agent interacts with.
- **State (S)**: The current situation or context of the environment.
- **Action (A)**: The decision or move the agent makes in a given state.
- **Reward (R)**: Feedback received after taking an action, which guides the agent's learning.
- **Policy (π)**: A strategy that maps states to actions.
- **Value Functions**: Functions that estimate the expected cumulative reward:
  - **State-Value Function ($V(s)$)**: The expected reward starting from state s.
  - **Action-Value Function ($Q(s, a)$)**: The expected reward starting from state s and taking action a.

RL - 3: How it works?

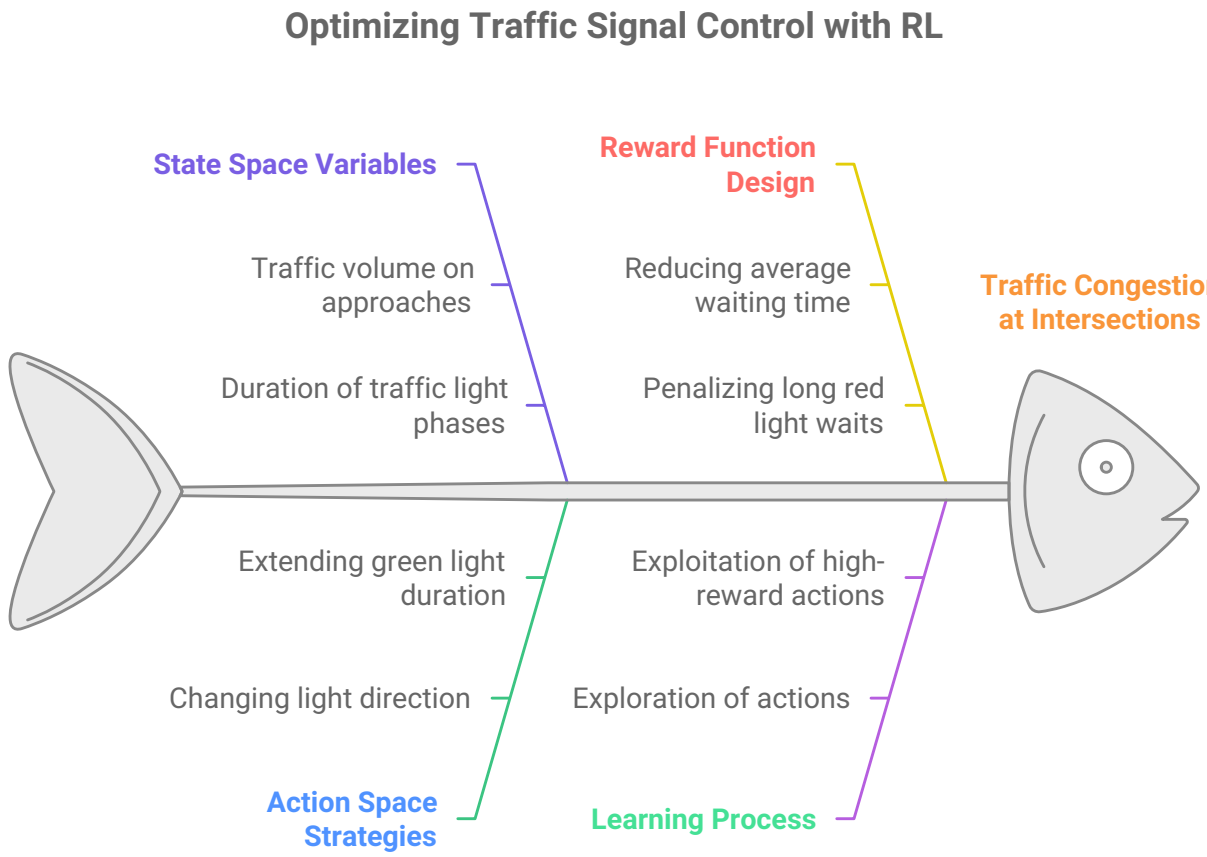
The agent interacts with the environment in a loop:

1. The agent observes the current **state** (s).
2. It selects an **action** (a) based on its policy (π).
3. The environment transitions to a new **state** (s') and provides a **reward** (r).
4. The agent updates its policy to improve future decisions, aiming to maximize the cumulative reward.

RL - 4: Exploration vs Exploitation

The agent must balance:

- **Exploration**: Trying new actions to discover better strategies.
- **Exploitation**: Using known actions to maximize rewards.



RL - 5: The Softmax Algorithm

- The **softmax function** is a mathematical function that converts a vector of real numbers into a probability distribution.
- It is commonly used in reinforcement learning and neural networks **to model the selection probabilities of different actions or outputs**.
- The softmax function is defined as:

$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{n} e^{z_j}}, \quad \text{for } i = 1, 2, \ldots, n$$

Where:

- z_i is the i-th element of the input vector.
- n is the number of elements in the vector.
- e^{z_i} is the exponential of z_i.

Softmax Example

- Suppose the logits (raw scores) for three actions are $[3, 1.2, 0.55]$.
- The softmax probabilities are calculated as:

$$\text{softmax}(z) = \left[\frac{e^3}{e^3 + e^{1.2} + e^{0.55}}, \frac{e^{1.2}}{e^3 + e^{1.2} + e^{0.55}}, \frac{e^{0.55}}{e^3 + e^{1.2} + e^{0.55}} \right]$$

- After computing, the probabilities might look like:

$$\text{softmax}(z) = [0.84, 0.11, 0.05]$$

- This means the first action is most likely to be chosen.

RL - 6: Softmax with Temperature

- The **temperature parameter** (T) is introduced in the softmax function to control the "sharpness" or "smoothness" of the output probability distribution.
- The modified softmax function with temperature is defined as:

$$\text{softmax}_T(z_i) = \frac{e^{z_i/T}}{\sum_{j=1}^{n} e^{z_j/T}}, \quad \text{for } i = 1, 2, \ldots, n$$

Where:

- $T > 0$ is the temperature parameter.
- When $T = 1$, the function behaves like the standard softmax.
- When $T > 1$, the output probabilities become more **uniform** (smoother distribution $\rightarrow$ **More exploration**).
- When $T < 1$, the output probabilities become more **peaked** (sharper distribution $\rightarrow$ **More Exploitation**).

RL - 7: Softmax in RL

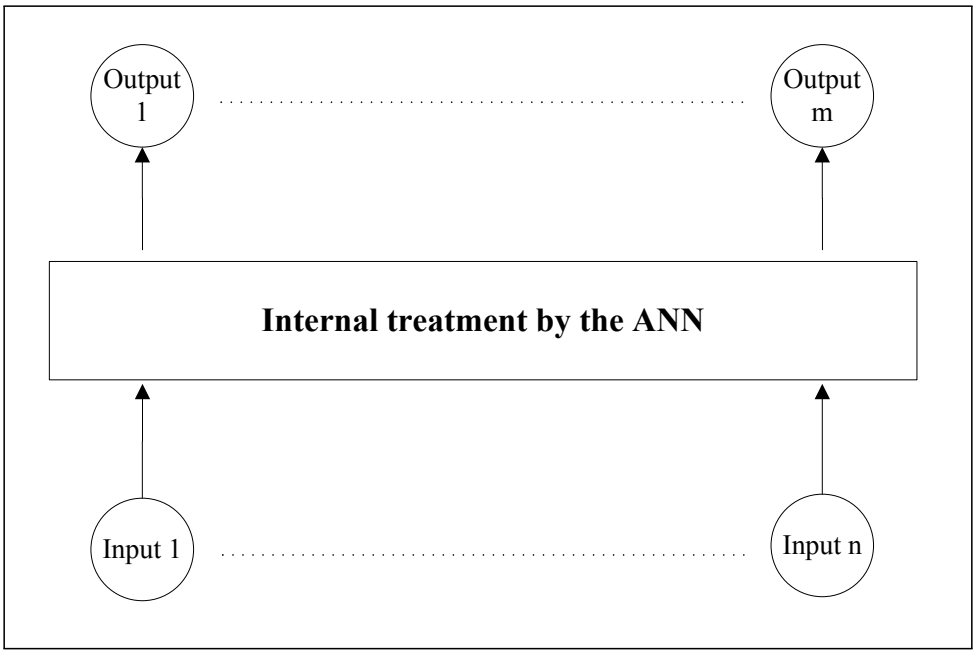
- **In RL**: The softmax function is often used to model the agent's policy.
- It converts the action-value estimates ($Q(s, a)$) into probabilities, allowing the agent to select actions probabilistically (*exploration*).
- **In LLMs**: The softmax function is used in the output layer to convert the raw logits (unscaled scores) into probabilities over the vocabulary.
- This ensures that the model generates coherent and probabilistic responses.

Outline

Artificial Neural Networks (ANN)

- Neural networks are computational architectures inspired by the human brain.
- They consist of interconnected layers of neurons, each performing a weighted sum of inputs followed by a nonlinear activation.
- Neural networks are the foundation of LLMs, allowing them to learn complex patterns in data.
- Feedforward networks are the simplest form of artificial neural networks.
- They consist of layers of neurons where information flows in one direction, from input to output. Each neuron in a layer is connected to all neurons in the subsequent layer.
- Read also a very soft introduction to ANNs.

Basic structure of an ANN



General structure of an ANN

ANN with hidden layers

A very common class of ANNs.
The network $\rightarrow$ neurons that receive the inputs I_i and transform them to outputs.

$$\left.\begin{array}{c} \theta_1 I_1 \\ \vdots \\ \theta_{n-1} I_{n-1} \\ 1 \cdot I_n \end{array}\right\} \longrightarrow \Sigma = \sum_{i=1}^{n-1} \theta_i I_i + I_n \longrightarrow f(\Sigma) \longrightarrow Output$$

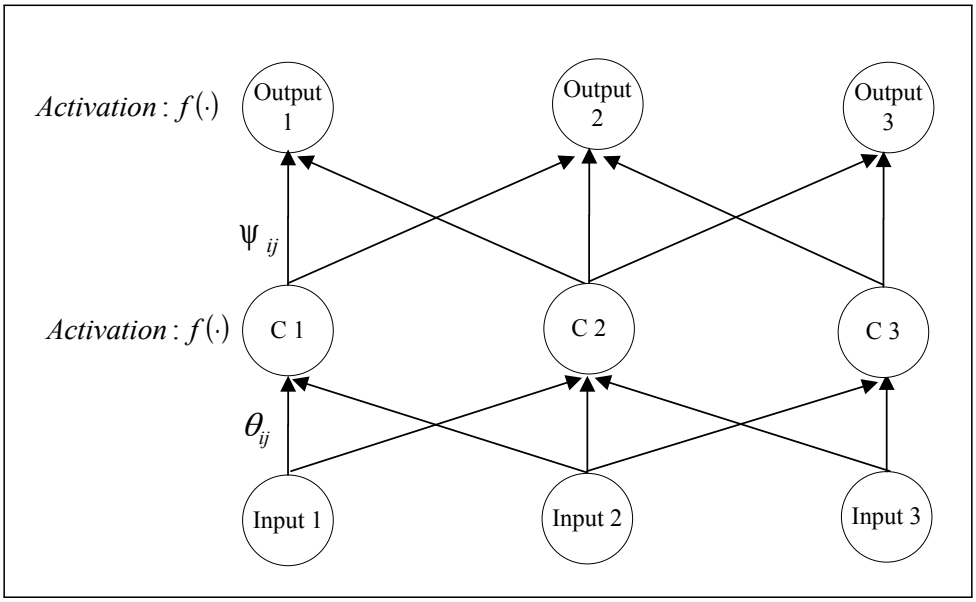
where f is the *activation function*.
The workings of the neuron is principally determined by the coefficients θ_i but the learning speed can strongly depend on the activation function.

Simplest ANN: A neuron uniquely composed of inputs nodes (corresponding to I_i) and one output node.
$\rightarrow$ f function can be disregarded (through a redefinition);
ANN $\rightarrow$ to estimate

$$O = I_n + \sum_{i=1}^{n-1} \theta_i I_i$$

$\rightarrow$ OLS if one use the minimization of the square of the errors.

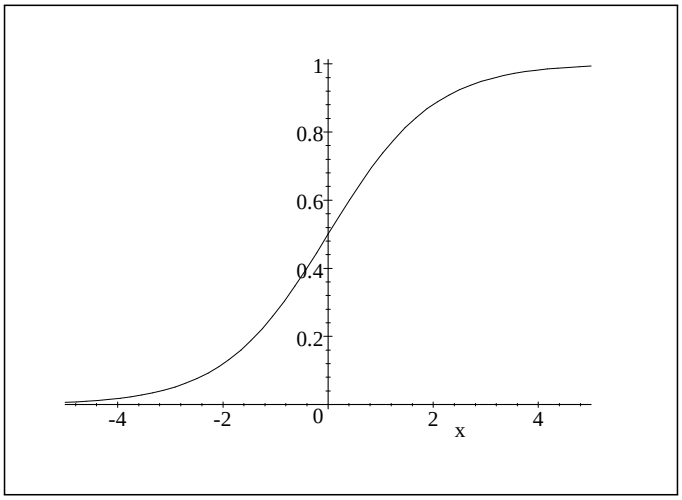
Hidden layer $\rightarrow$ the ANN can also learn non-linear relationships :



An ANN with one hidden layer

Same activation function at all nodes except for the output node (where a linear function ($f(x) = x$) is generally sufficient).
The *logistic* function is the most commonly used structure
$\rightarrow$ it possess a convenient derivative:

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



The logistic activation function

Learning in an ANN

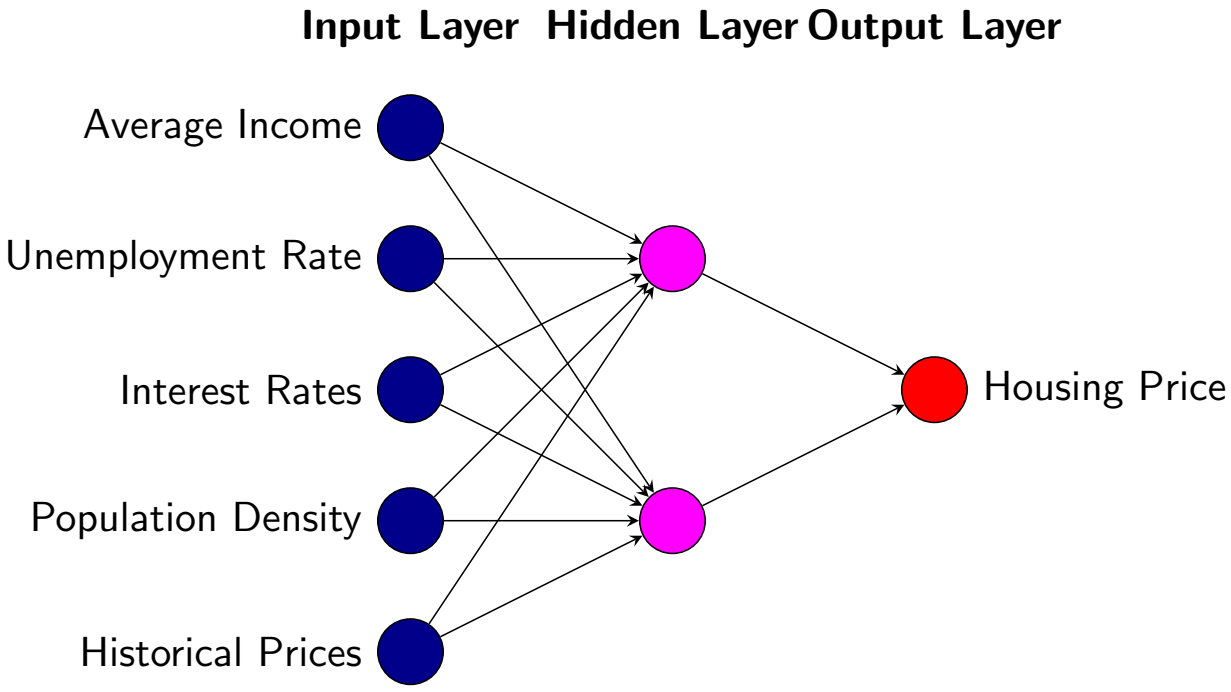
Learning→ adjustment of the coefficients θ_{ij} and ψ_{ij} in order to minimize the chosen measure for the errors.
General case → a strongly non-linear optimization problem with a high number of significantly correlated variables.
Training:

- → random initialization of all coefficients to strictly positive values
- → confrontation with an observations sample (*input*, *output*)
- → computation of the error
- → adjustment of the coefficients→ reducing the errors (such a cycle is called an *epoch*). (Rumelhart *et al.*, 1986)

The most common measure of the errors:
→ the mean square error ← its derivatives can be explicitly computed for each coefficient.
These derivatives are computed starting from the output nodes (*back propagation of errors)*, following an algorithm similar to *gradient descent*.

An Economic Example

- Predicting housing prices based on economic and demographic factors
- **Input Data:** The ANN takes input features such as
  - average income,
  - unemployment rate,
  - interest rates,
  - population density, and
  - historical housing prices in a region.
- **Hidden Layers:** The network processes these inputs through hidden layers, where it learns complex relationships between the features (e.g., how income and interest rates jointly affect housing prices).
- **Output:** The ANN predicts the housing price for a given region.
- $\rightarrow$ ANNs can capture non-linear relationships between economic variables, making them effective for forecasting in complex systems like housing markets.



Some general properties (Masters, 1993, p. 86-87)

(The capability of *learning* of a simple ANN: its ability to reduce arbitrarily the errors)

1. If the relationship to learn consists of a finite collection of points, a three-layer network (one hidden layer) **is capable** of learning it.
2. If the relationship is continuous and defined on a compact domain, a three-layer network **is capable** of learning it.
3. Many relationships that do not meet the above criteria can also be learned by a three-layer network. In particular, discontinuities can be theoretically tolerated under all conditions likely to be met in real life.
4. Under very general conditions, all other relationships that can be learned by a neural network can be learning by a four-layer (two hidden) network.

Outline

Advanced Neural Network Architectures

- **Convolutional Neural Networks (CNNs):** CNNs are particularly effective for processing grid-like data, such as images.
- They use convolutional layers to detect local patterns and pooling layers to reduce dimensionality (LeCun *et al.*, 1998).
- **Recurrent Neural Networks (RNNs):** RNNs were designed to handle sequential data by maintaining an internal state (memory).
- RNNs are specifically designed to handle data where the order of inputs matters, such as time-series data, text, or speech.
- They struggle with long-term dependencies due to the vanishing gradient problem (Hochreiter & Schmidhuber, 1997).
- Variants of RNNS: Long Short-Term Memory (LSTM) Networks (Hochreiter & Schmidhuber, 1997), and their simpler version, Gated Recurrent Units (GRUs)
- But the star of LLMs are **Transformer**-based architectures.

Transformers everywhere!

Company	**LLM**	**Neural Network Type**	**Key Features**
OpenAI	GPT (e.g., GPT-4)	Decoder-Only Transformer	Causal attention, autoregressive generation, scaled to billions of parameters.
Anthropic	Claude	Transformer-Based Architecture	Focus on alignment, RLHF, safety, and robustness.
Google	Gemini	Multimodal Transformer	Multimodal integration (text, images), advanced reasoning capabilities.
Mistral	Mistral 7B	Transformer-Based Architecture	Dense and Mixture-of-Experts models, optimized for efficiency and open access.
Meta	LLaMA	Decoder-Only Transformer	Efficient, open-weight models, optimized for smaller hardware setups.

Deep learning: Convolutional NN (LeCun *et al.*, 1998)

- CNNs are a specialized type of neural network designed to process **structured data** like images, videos, and other **grid-like data** (e.g., time-series data).
- CNNs are particularly effective for tasks involving spatial hierarchies, such as image classification, object detection, and natural language processing (e.g., text classification).
- The key principle of CNNs is the use of convolutional layers to automatically learn spatial features (e.g., edges, textures, shapes) from the input data.
- CNNs are inspired by the visual processing system of the human brain, where neurons respond to specific regions of an image (called receptive fields).
- The network learns hierarchical features:
  - Low-level features: Edges, corners, textures.
  - Mid-level features: Shapes, patterns.
  - High-level features: Objects, faces, or other complex structures.

How CNNs Work: Step-by-Step

1. **Input Layer:** The input is typically an image represented as a grid of pixel values (e.g., a 2D matrix for grayscale images or a 3D tensor for RGB images).
2. **Feature Extraction:** Convolutional layers apply filters to detect patterns in the input, producing feature maps.
3. **Pooling layers** downsample the feature maps to reduce dimensionality and retain important features.
4. **Classification:** Fully connected layers combine the extracted features to make predictions (e.g., classify the image into categories, using the *softmax* algorithm)).

ConvolutionNN Example

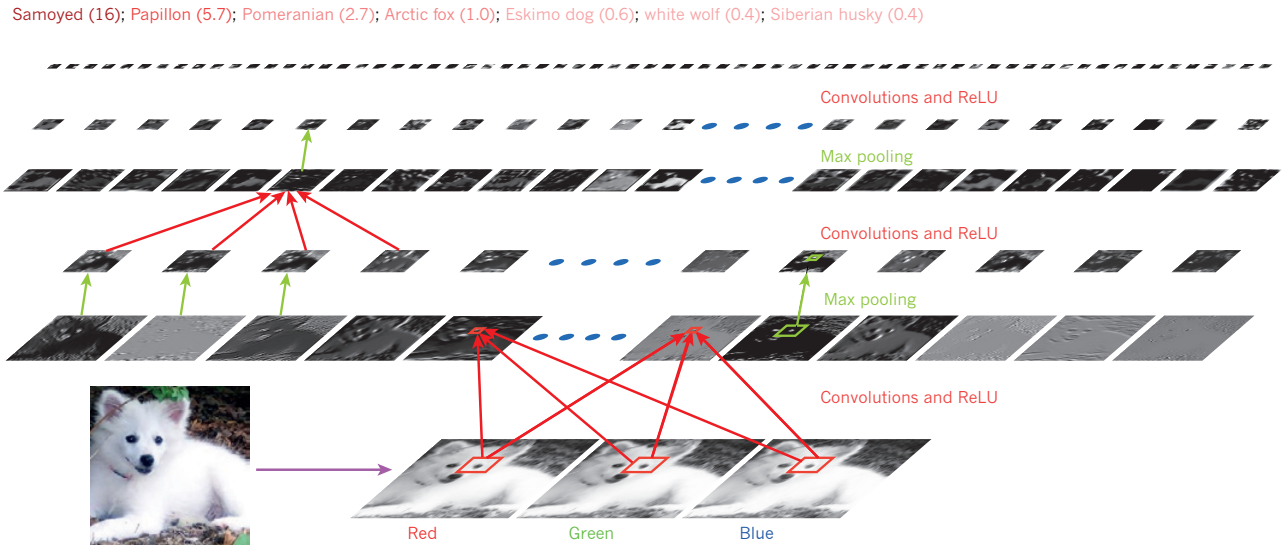


Figure 2 | Inside a convolutional network. The outputs (not the filters) of each layer (horizontally) of a typical convolutional network architecture applied to the image of a Samoyed dog (bottom left; and RGB (red, green, blue) inputs, bottom right). Each rectangular image is a feature map corresponding to the output for one of the learned features, detected at each of the image positions. Information flows bottom up, with lower-level features acting as oriented edge detectors, and a score is computed for each image class in output. ReLU, rectified linear unit.

Evolution of NLP

- Natural Language Processing (NLP) is the foundation of Large Language Models (LLMs), enabling machines to process, understand, and generate human language.
- NLP is a subfield of artificial intelligence that focuses on enabling machines to understand and generate human language.
- Over the years, NLP has evolved
  - from rule-based systems (Chomsky, 1957)
  - to statistical methods (Brown *et al.*, 1992) and, more recently,
  - to deep learning-based approaches (Mikolov *et al.*, 2013a; Vaswani *et al.*, 2017).
- LLMs, such as GPT (Radford *et al.*, 2018), BERT (Devlin *et al.*, 2018), and T5 (Raffel *et al.*, 2020), represent the culmination of these advances.

Text processing

- Text preprocessing is a critical step in NLP, preparing raw text data for analysis or model training.
- Key techniques include:
  - **Tokenization**: Splitting text into smaller units, such as words or subwords. Subword tokenization methods like Byte Pair Encoding (BPE) (Sennrich *et al.*, 2016) are widely used in LLMs.
  - **Stemming and Lemmatization**: Reducing words to their root forms (Porter, 1980).
  - **Stop Word Removal**: Removing common words that may not add significant meaning.
  - **The attention mechanism** allows the model to focus on different parts of the text with varying intensities, similar to how a financial analyst places more importance on certain indicators depending on the context.

Word Representations

- A cornerstone of NLP, enabling models to encode words as numerical vectors.
- Early methods: one-hot encoding, lacked semantic relationships.
- The introduction of **distributed representations**, such as Word2Vec (see above) and GloVe (Pennington *et al.*, 2014): a significant advancement.
- LLMs, however, generate contextual embeddings, where the meaning of a word depends on its **context in a sentence**.
- For example, BERT (Devlin *et al.*, 2018) captures the contextual meaning of words using bidirectional attention.

Tokenization

Tokenization goes far beyond simple word splitting. Let's examine the different strategies:

1. **Word Tokenization:**
   "Underlying inflation reaches 2.5%"
   → ["Underlying", "inflation", "reaches", "2.5", "%"]
2. **Subword Tokenization:**
   "Underlying inflation
   → ["Under", "##lying", "in", "##flation"]
3. **Tokenization Sensitive to Financial Context**
   "CAC 40" → ["CAC40"] (unique token)
   "S&P 500" → ["S&P500"] (unique token)

Language Modeling

- Language modeling is the task of predicting the probability of a sequence of words.
- Traditional approaches, such as n-gram models (Brown *et al.*, 1992): limited by their inability to capture long-range dependencies.
- Neural language models, such as RNNs (Elman, 1990) and LSTMs (Hochreiter & Schmidhuber, 1997), addressed this limitation but were computationally expensive.
- The Transformer architecture (Vaswani *et al.*, 2017) revolutionized language modeling by enabling parallel processing and capturing long-range dependencies through self-attention.

Transformer Architecture: Overview

- The Transformer architecture is the backbone of modern LLMs.
- Key components include:
- **Self-Attention Mechanism**: Self-attention allows the model to weigh the importance of different words in a sequence when encoding each word.
- Multi-head attention performs this process multiple times in parallel, allowing the model to capture different types of relationships (Vaswani *et al.*, 2017).
- The attention mechanism is defined as:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V,$$

where Q, K, and V are the query, key, and value matrices, respectively (Vaswani *et al.*, 2017). T is the temperature of the *Softmax* algorithm.

- **Positional Encodi**ng:
- Since the Transformer doesn't inherently understand the order of words,
- positional encodings are added to the input embeddings to provide information about the position of each word in the sequence,
- enabling the model to process sequential data.

Attention Mechanism

Concrete Example of Attention

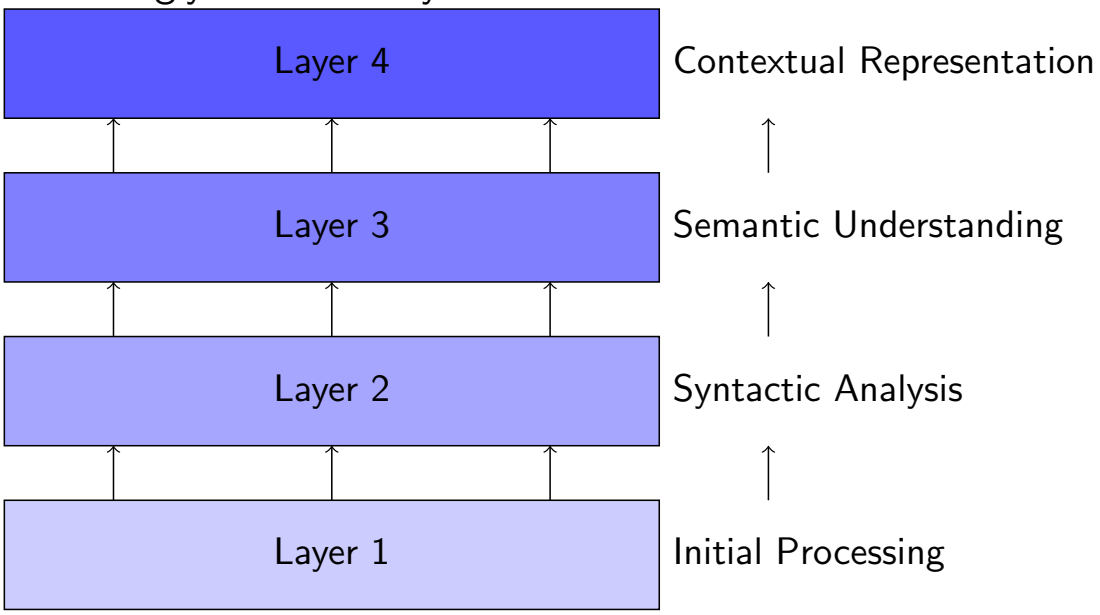
Let's take the sentence: "The ECB raises its key interest rates by 25 basis points"
The model will particularly "pay attention" to:

- "ECB": the main actor
- "raises": the action
- "25 basis points": the magnitude

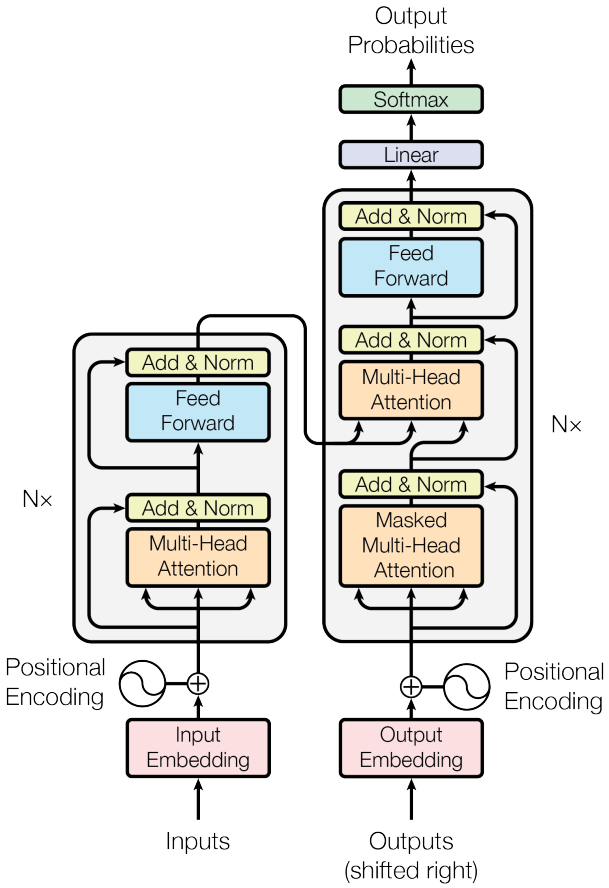
This differentiated attention allows understanding that it is a rate hike, not a cut, and that it is the ECB acting, not the Fed.

Encoding Layers

The architecture uses several layers that process information in an increasingly abstract way:

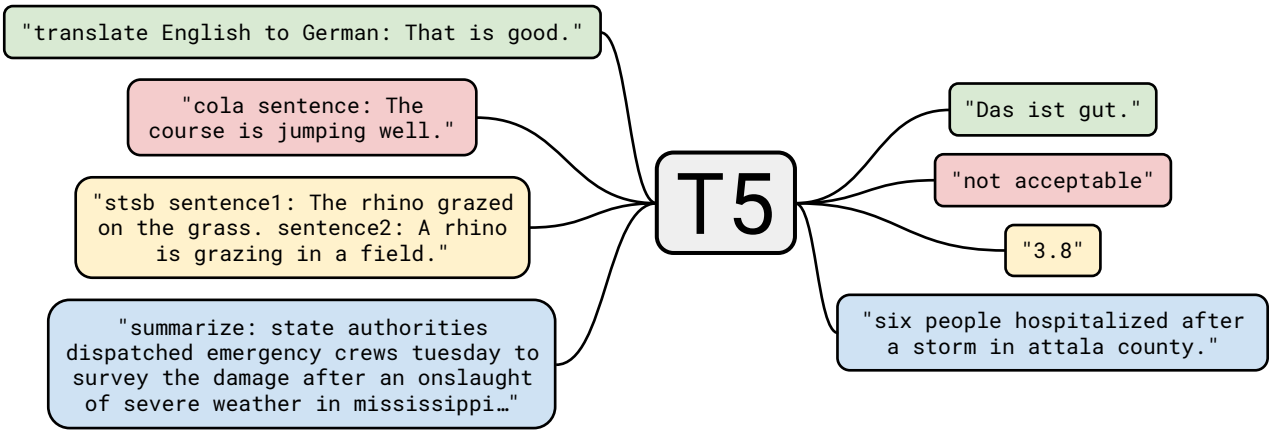


The Transformer Architecture (Vaswani *et al.*, 2017)



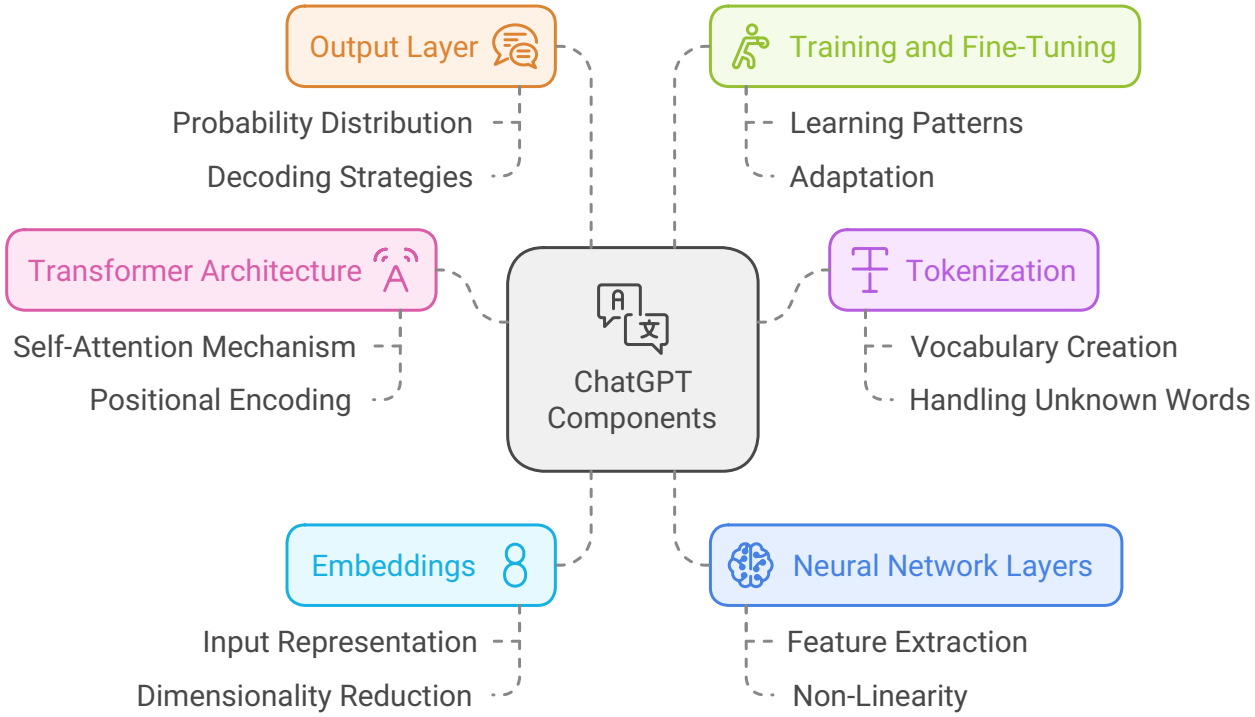
Transformer Variants

- **BERT** (Bidirectional Encoder Representations from Transformers) uses only the encoder part of the Transformer and is pre-trained on masked language modeling and next sentence prediction tasks (Devlin *et al.*, 2018).
- **GPT** (Generative Pre-trained Transformer) uses only the decoder part of the Transformer and is pre-trained on causal language modeling (Raffel *et al.*, 2020).
- **T5** (Text-to-Text Transfer Transformer) frames all NLP tasks as text-to-text problems and uses the full encoder-decoder architecture (Raffel *et al.*, 2020).

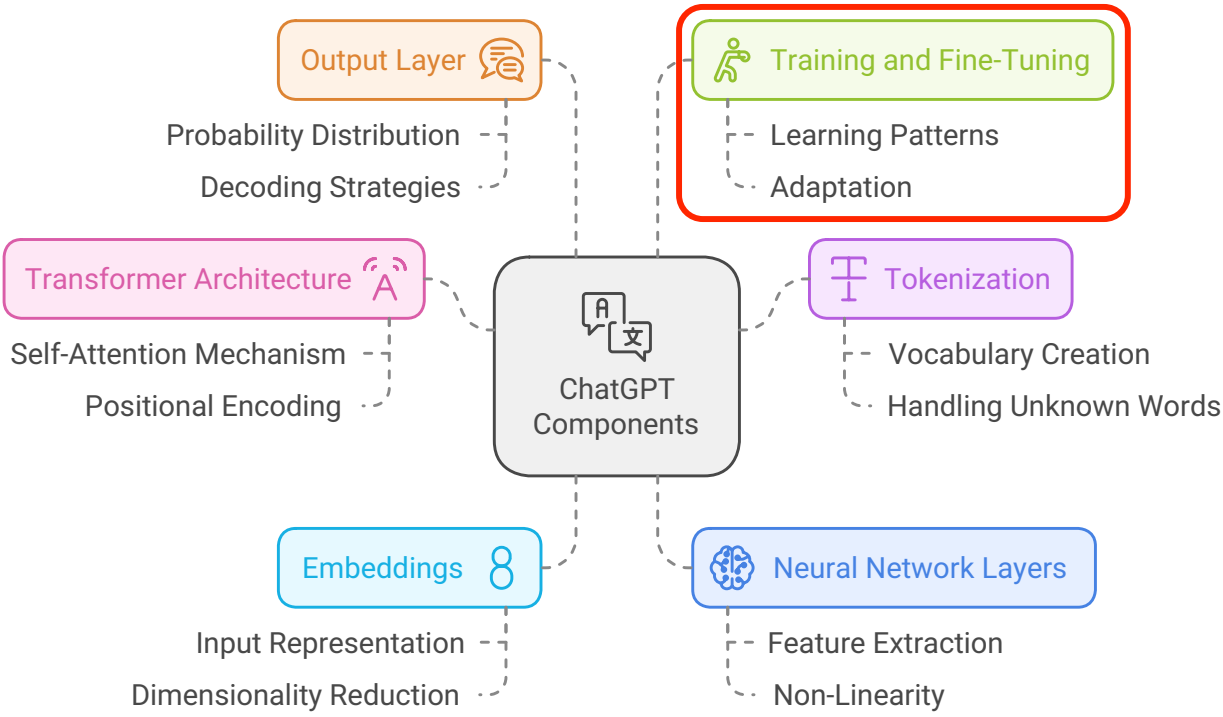


T5 framework. Source: (Raffel *et al.*, 2020)

Components of ChatGPT



Components of ChatGPT



Training Objectives

- Training objectives define the tasks that a language model is optimized to perform during training.
- These objectives guide the model in learning patterns from data.
- Common objectives include:
  - **Masked Language Modeling (MLM):** In MLM, some tokens in the input sequence are masked, and the model is trained to predict these masked tokens.
  - This allows the model to learn bidirectional context (Devlin *et al.*, 2018, for BERT).
  - In MLM, the sentence "The ___ is blue" is completed as "The sky is blue."

Training Objectives - 2

- **Next Sentence Prediction (NSP):** NSP trains the model to understand the relationship between sentences by predicting whether two given sentences are consecutive in the original text (Devlin *et al.*, 2018, idem).
  - In NSP, given two sentences, "The stock market crashed yesterday." and "Investors are worried about the future," the model predicts whether the second sentence logically follows the first. This is useful in economics for analyzing the coherence of financial reports.
- **Causal Language Modeling (CLM):** This objective trains the model to predict the next token given the previous tokens, which is particularly useful for text generation tasks (Radford *et al.*, 2018).
- **Span Prediction**: Identifying a span of text that answers a question.
- **Contrastive Learning:** Learning representations by distinguishing between similar and dissimilar examples.

Example: Objective for MLM

For MLM, the objective is to minimize the negative log-likelihood of the masked tokens:

$$\mathcal{L}_{\text{MLM}} = -\sum_{i \in \text{masked}} \log P(x_i \mid X_{\setminus i}),$$

where $X_{\setminus i}$ represents the input sequence with the i-th token masked.

Optimization Techniques during training

- Competing optimization techniques aim to increase the precision and the robustness of prediction and the efficacy in terms of computing resources.
- **Stochastic Gradient Descent** (**SGD**) updates parameters using a small batch of data (Bottou, 2010).
- **Adam** optimizer: Adam (Adaptive Moment Estimation) is an extension of SGD that adapts the learning rate for each parameter, often leading to faster convergence (Kingma & Ba, 2014).

- **Learning rate schedules** adjusts the learning rate during training.
  - They are critical in training large models like GPT.
  - For instance, a "warm-up" phase with a gradually increasing learning rate followed by a decay phase ensures stable convergence.
  - This is particularly important when training on large datasets like economic reports.
- **Gradient clipping** prevents exploding gradients by capping their magnitude.
- **Layer-wise Adaptive Rate Scaling (LARS)** scales learning rates for large-batch training.

SGD

- Stochastic Gradient Descent (SGD) is an optimization algorithm widely used in training machine learning models, including Large Language Models (LLMs).
- It is a variant of the gradient descent algorithm that updates model parameters using **a small, randomly selected subset** (mini-batch) of the training data at each iteration, rather than the entire dataset.
- The update rule for SGD is:

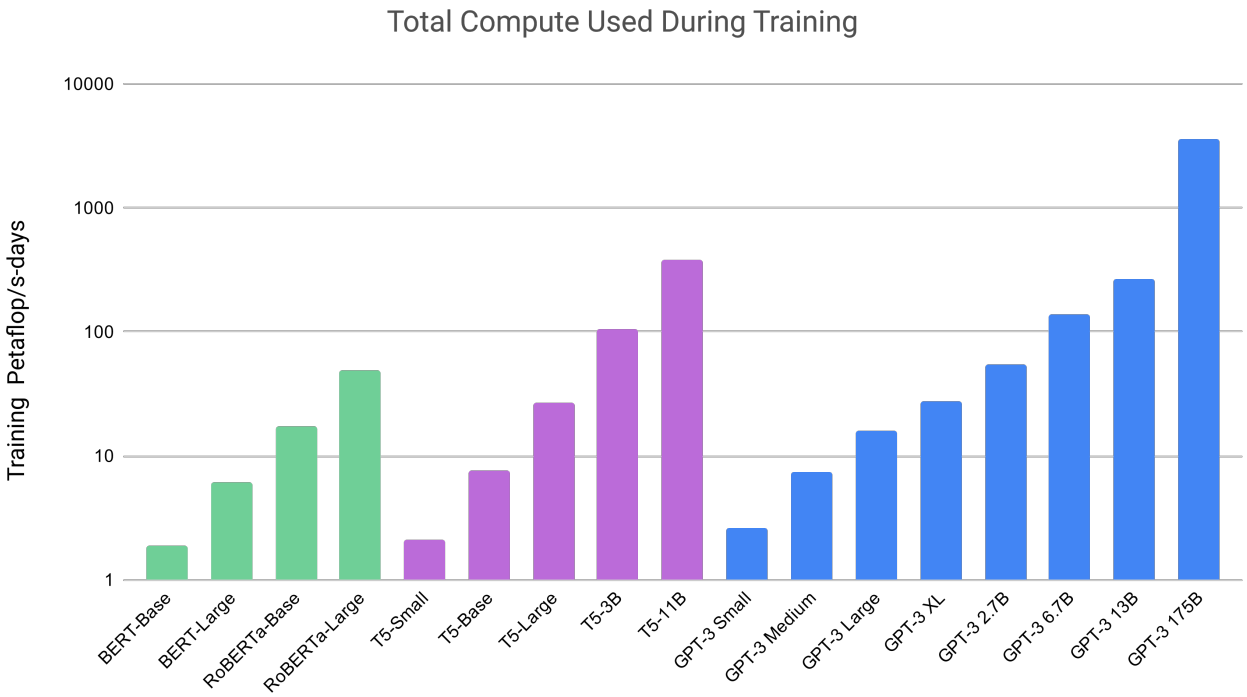
$$\theta_{t+1} = \theta_t - \eta \nabla_\theta \mathcal{L}(\theta; x_i, y_i)$$

Where:

- θ_t: Model parameters at iteration t.
- η: Learning rate, which controls the step size of the update.
- $\nabla_\theta \mathcal{L}(\theta; x_i, y_i)$: Gradient of the loss function $\mathcal{L}$ with respect to the parameters θ, computed for a single training example (x_i, y_i) or a mini-batch.

Computational Considerations

- **Hardware requirements (GPUs, TPUs):** Training LLMs requires significant computational resources, often utilizing multiple GPUs or TPUs in parallel (Brown *et al.*, 2020, see next slide).
- **Distributed Training:** To handle the memory constraints of massive scale of LLMs, distributed training techniques are employed to split the model or data across multiple devices (Shoeybi *et al.*, 2019).
- **Mixed precision training:** Mixed precision training uses a combination of 32-bit and 16-bit floating-point representations to reduce memory usage and increase training speed (Micikevicius *et al.*, 2018).
- **Gradient accumulation:** Combines gradients over multiple steps to simulate larger batch sizes.
- **Model parallelism vs. data parallelism:** Splits the model or data across devices.



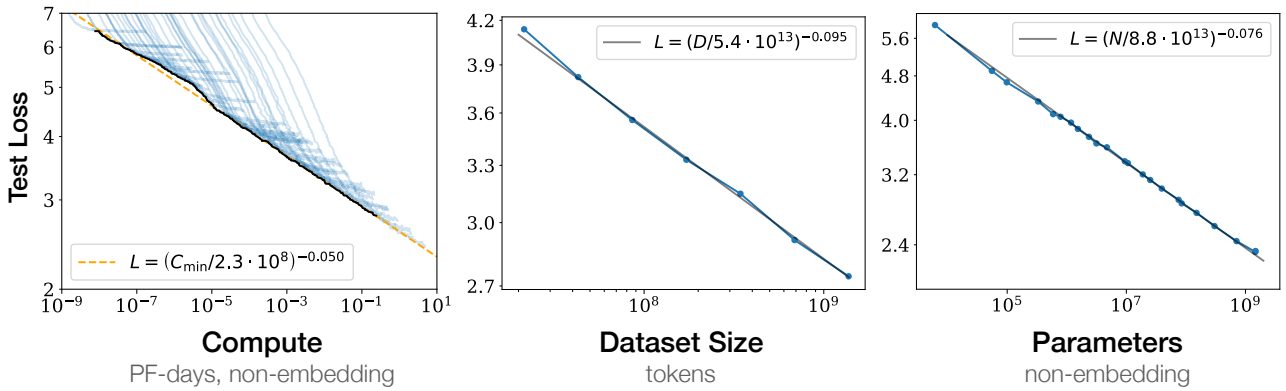
Source: Brown *et al.* (2020)

Scaling Laws in LLMs

- The performance of LLMs follows predictable scaling laws as the amount of compute used for training increases (Kaplan *et al.*, 2020):
- Performance depends strongly on scale, weakly on model shape: Model performance depends most strongly on **scale**, which consists of three factors:
  - the number of model parameters N (excluding embeddings),
  - the size of the dataset D, and
  - the amount of compute C used for training.
- Within reasonable limits, performance depends very weakly on other architectural hyperparameters.
- This relationship can be approximated by a power law:

$$\text{Loss} \propto (\text{Compute})^{-\alpha}$$

Where α is a scaling exponent that depends on the specific task and model architecture.



Source: (Kaplan *et al.*, 2020)

Implications for model development

- Larger models are significantly more sample-efficient.
- Optimally compute-efficient training involves training
  - **very large models**
  - on a relatively modest amount of data and
  - stopping significantly before convergence.

Emergent Abilities

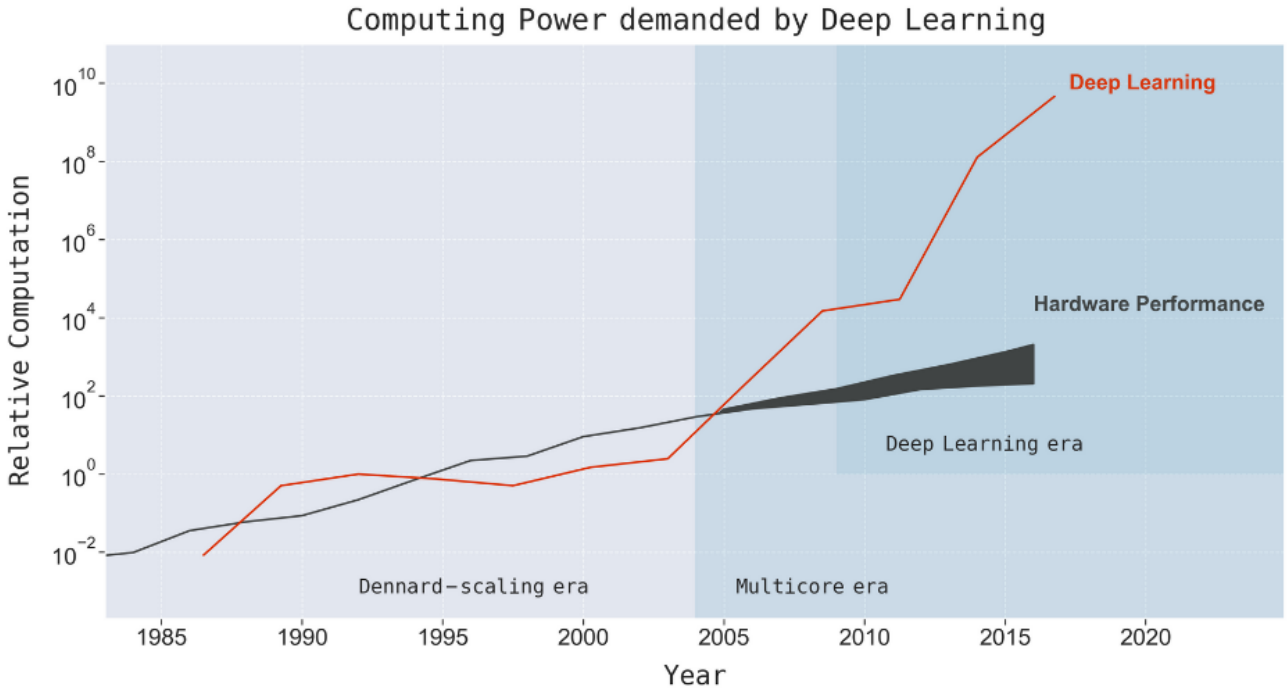
As LLMs scale, they demonstrate emergent abilities - capabilities that are not explicitly trained for but arise as the model becomes more powerful (Wei *et al.*, 2022). Examples include:

- **In-context learning:** The ability to perform new tasks based on a few examples provided in the input.
- **Few-shot learning:** The capacity to generalize from a small number of training examples.
- **Task decomposition:** The ability to break down complex tasks into simpler subtasks.
- **Chain-of-thought reasoning:** The ability to break down complex problems into intermediate reasoning steps, improving their performance on tasks requiring logical reasoning, arithmetic, or multi-step problem-solving.

Limitations of Scaling

While scaling has led to significant improvements, it also faces limitations:

- **Computational constraints:** The resources required for training grow exponentially with model size (Thompson *et al.*, 2020).
- **Diminishing returns:** Performance gains may slow down as models become extremely large (Kaplan *et al.*, 2020).
- **Environmental concerns:** The energy consumption of training large models has raised sustainability issues (Strubell *et al.*, 2019).
- **Natural data shortage:** Nearly all available original data sources are already exploited by the extant LLMs. Where to fine new data to increase performance?



Source: (Thompson *et al.*, 2020)

Murat Yıldızoğlu Chap1: Principles of LLMs

References I

Bottou, Léon. 2010. Large-scale machine learning with stochastic gradient descent. *Proceedings of COMPSTAT'2010*, 177–186.

Brown, Peter F, *et al.* 1992. Class-based n-gram models of natural language. *Computational linguistics*, **18**(4), 467–479.

Brown, Tom B, Mann, Benjamin, Ryder, Nick, Subbiah, Melanie, Kaplan, Jared, Dhariwal, Prafulla, Neelakantan, Arvind, Shyam, Pranav, Sastry, Girish, Askell, Amanda, *et al.* 2020. Language Models are Few-Shot Learners. *arXiv preprint arXiv:2005.14165*.

Chomsky, Noam. 1957. *Syntactic Structures*. Mouton.

Devlin, Jacob, Chang, Ming-Wei, Lee, Kenton, & Toutanova, Kristina. 2018. BERT: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Elman, Jeffrey L. 1990. Finding structure in time. *Cognitive science*, **14**(2), 179–211.

Hochreiter, Sepp, & Schmidhuber, Jürgen. 1997. Long short-term memory. *Neural computation*, **9**(8), 1735–1780.

Kaplan, Jared, McCandlish, Sam, Henighan, Tom, Brown, Tom B, Chess, Benjamin, Child, Rewon, Gray, Scott, Radford, Alec, Wu, Jeffrey, & Amodei, Dario. 2020. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*.

Kingma, Diederik P, & Ba, Jimmy. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

LeCun, Yann, Bottou, Léon, Bengio, Yoshua, & Haffner, Patrick. 1998. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, **86**(11), 2278–2324.

LeCun, Yann, Bengio, Yoshua, & Hinton, Geoffrey. 2015. Deep learning. *Nature*, **521**(7553), 436–444.

Masters, Timothy. 1993. *Practical Neural Network recipes in C++*. New York: Academic Press.

Micikevicius, Paulius, Narang, Sharan, Alben, Jonah, Diamos, Gregory, Elsen, Erich, Garcia, David, Ginsburg, Boris, Houston, Michael, Kuchaiev, Oleksii, Venkatesh, Ganesh, & Wu, Hao. 2018. Mixed Precision Training. *In: International Conference on Learning Representations*.

References II

Mikolov, Tomas, Sutskever, Ilya, Chen, Kai, Corrado, Greg S, & Dean, Jeff. 2013a. Distributed representations of words and phrases and their compositionality. *Advances in neural information processing systems*, **26**.

Mikolov, Tomas, Chen, Kai, Corrado, Greg, & Dean, Jeffrey. 2013b. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Pennington, Jeffrey, Socher, Richard, & Manning, Christopher D. 2014. Glove: Global vectors for word representation. *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, 1532–1543.

Porter, Martin F. 1980. An algorithm for suffix stripping. *Program*, **14**(3), 130–137.

Radford, Alec, Narasimhan, Karthik, Salimans, Tim, & Sutskever, Ilya. 2018. Improving language understanding by generative pre-training.

Raffel, Colin, *et al.* 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, **21**, 1–67.

Rumelhart, David E, Hinton, Geoffrey E, & Williams, Ronald J. 1986. Learning representations by back-propagating errors. *Nature*, **323**(6088), 533–536.

Sennrich, Rico, Haddow, Barry, & Birch, Alexandra. 2016. Neural machine translation of rare words with subword units. *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 1715–1725.

Shoeybi, Mohammad, Patwary, Mostofa, Puri, Raul, LeGresley, Patrick, Casper, Jared, & Catanzaro, Bryan. 2019. Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism. *arXiv preprint arXiv:1909.08053*.

Strubell, Emma, Ganesh, Ananya, & McCallum, Andrew. 2019. Energy and policy considerations for deep learning in NLP. *arXiv preprint arXiv:1906.02243*.

Sutton, Richard S, & Barto, Andrew G. 2018. *Reinforcement learning: An introduction*. MIT press.

Thompson, Neil C, Greenewald, Kristjan, Lee, Keeheon, & Manso, Gabriel F. 2020. The computational limits of deep learning. *arXiv preprint arXiv:2007.05558*.

Vaswani, Ashish, *et al.* 2017. Attention is all you need. *Pages 5998–6008 of: Advances in neural information processing systems*.

Wei, Jason, Tay, Yi, Bommasani, Rishi, Raffel, Colin, Zoph, Barret, Borgeaud, Sebastian, Yogatama, Dani, Bosma, Maarten, Zhou, Denny, Metzler, Donald, *et al.* 2022. Emergent abilities of large language models. *arXiv preprint arXiv:2206.07682*.