(SCTP) Advanced Professional Certificate

Data Science and AI

3.10 Natural Language Processing - Advanced

Module Overview

Lesson Objectives

- Deep Learning for NLP
- Recurrent Neural Networks (RNN)
- Attention Mechanism
- Transformers

Common NLP Tasks

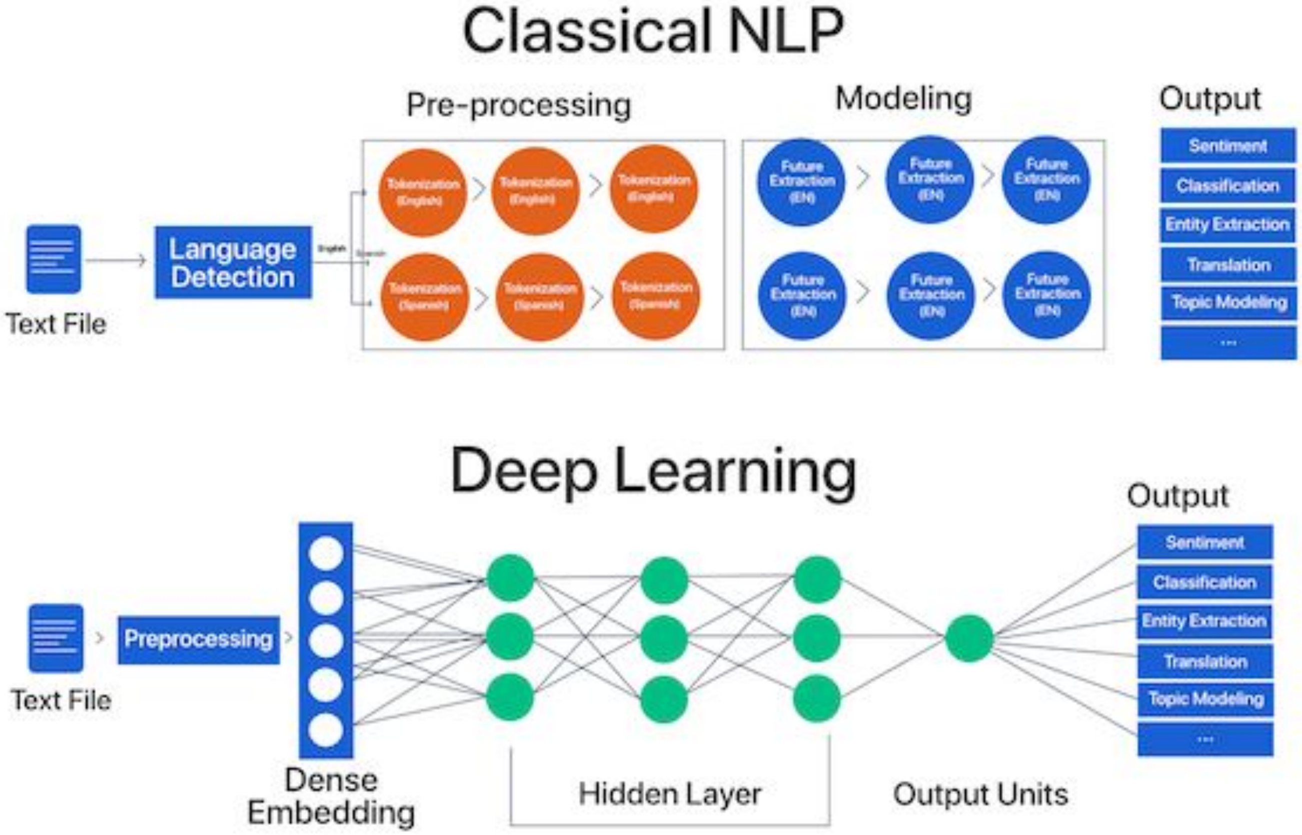
- Text Classification
- Information Extraction (e.g. NER)
- Question Answering
- Summarisation
- Text Generation
- Style Transfer
- Translation

…

Style Attribute	***Source Attribute / Sentence***	***Target Attribute / Sentence***
Politeness	Polite: *"**Could you please** send me the data**?**"*	Impolite: *"send me the data**!!**"*
Toxicity	Offensive: *"I hope they pay out the *******."*	Non-offensive: *"I hope they pay **what they deserve**."*
Simplicity	Expert: *"Many cause **dyspnea**, pleuritic chest pain, or both."*	Layman: *"The most common symptoms, regardless of the type of fluid in the pleural space or its cause, are **shortness of breath** and chest pain."*
Biasedness	Biased: *"A new downtown is being developed **which will bring back**..."*	Neutral: *"A new downtown is being developed **which its promoters hope** will bring back..."*
Authorship	Shakespearean: *"My lord, the queen **would** speak with you**, and presently.**"*	Contemporary: *"My lord, the queen **wants to** speak with you **right away.**"*

https://blog.fastforwardlabs.com/2022/03/22/an-introduction-to-text-style-transfer.html

Deep Learning for NLP



Deep Learning for NLP

Limitations of Traditional Models:

Sparsity: Traditional text representations like bag-of-words result in sparse vectors that are not efficient for capturing the semantic meaning of words.

Feature Engineering: Crafting features requires domain expertise and is often labor-intensive and time-consuming.

Contextual Information: Capturing context and word order is challenging with traditional models.

Deep Learning for NLP

Emergence of Deep Learning Models:

Deep learning models introduced end-to-end learning, where raw data can be fed into the model, and the model itself learns the features that are most predictive for the task.

Word Embeddings: Deep learning introduced word embeddings (e.g., Word2Vec, GloVe), which are dense vector representations of words capturing semantic meaning.

Sequence Models: RNNs and their variants like LSTMs and GRUs were developed to handle sequences and capture temporal dependencies in text.

Attention Mechanisms: Attention allows models to focus on different parts of the input sequence when predicting an output, improving performance on tasks like machine translation.

Transformer Models: The Transformer architecture, which relies solely on attention without recurrence, has achieved state-of-the-art results in many NLP tasks.

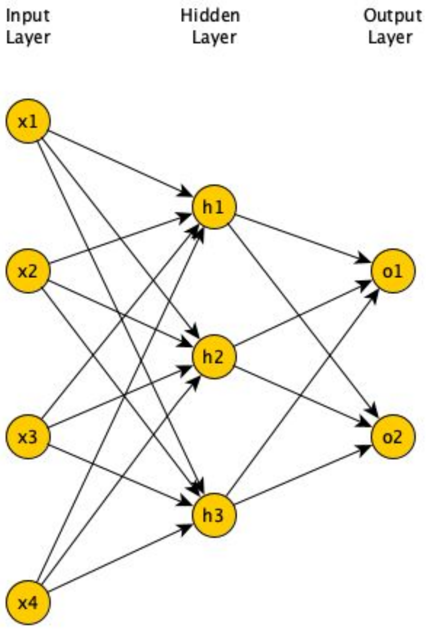
Text Generation

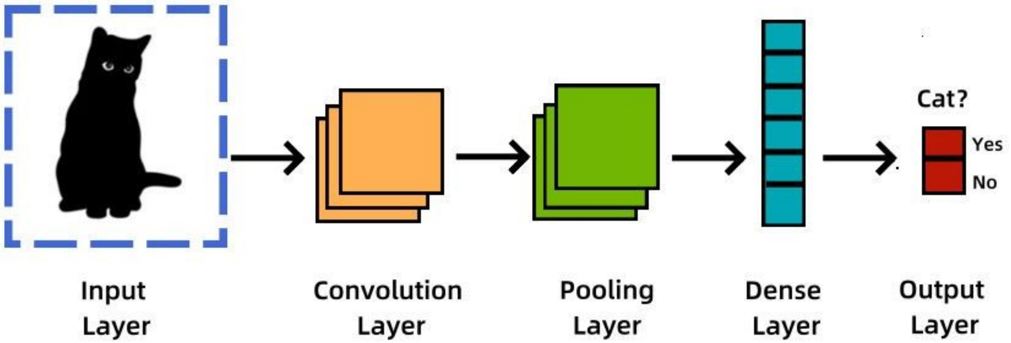
Mary → had

Mary had → a

Mary had a → little

Mary had a little → lamb



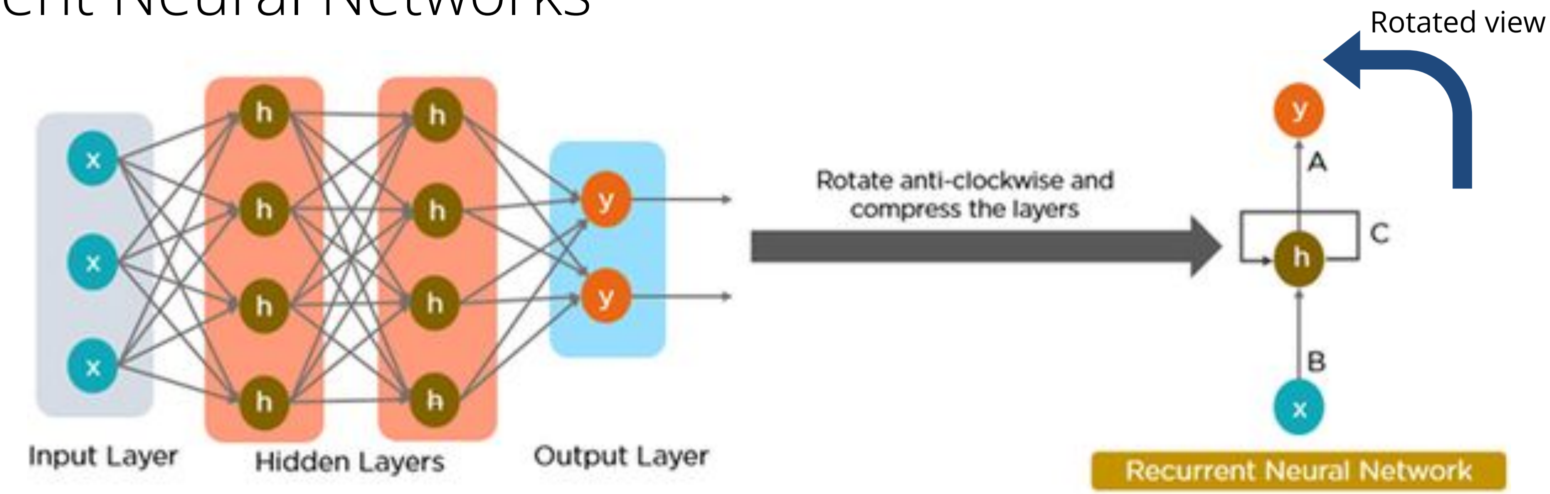


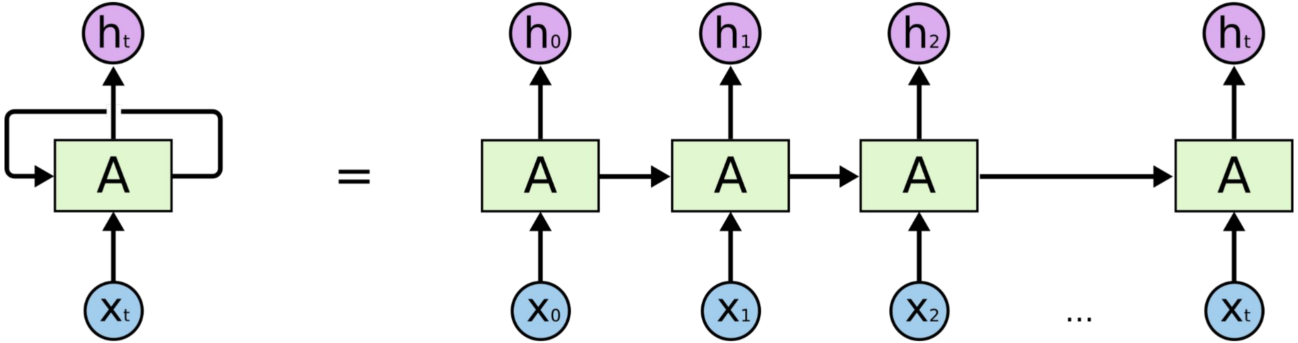
Feedforward networks lack Spatial awareness → **CNN**

Recurrent Neural Networks

Feedforward networks lack memory → **RNN**

Recurrent Neural Networks





https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Recurrent Neural Network

Step 1: Prepare the Data

You have a big corpus of text (like Penn Treebank).

- Tokenize the text → turn words into integers (using a tokenizer + vocab)
- Create sequences of tokens → input and target pairs

Example:

Suppose the tokenized data is:

["The", "quick", "brown", "fox", "jumps", "over"]

Inputs

`["The", "quick", "brown", "fox", "jumps"]`

Targets

`["quick", "brown", "fox", "jumps", "over"]`

Recurrent Neural Network

Step 2: Initialize the Model

Create an RNN model:

- **Embedding layer** → to map tokens into dense vectors.
- **RNN layer** → to process sequences.
- Fully connected layer → to map hidden states to vocab predictions.

Also set up:

- Loss function → typically CrossEntropyLoss (for classification).
- Optimizer → like Adam (to update weights).

Define vocab size and embedding size

Memory: RNNs maintain a hidden state (or memory) at each time step, which is passed to the next time step.

This hidden state allows the network to retain information about past inputs, which is crucial for learning sequential data (e.g., time series, language).

Typically - tanh activation function

Recurrent Neural Network

Step 3: Set the Initial Hidden State

Before feeding each batch into the RNN, initialize the hidden state:

```
hidden = model.init_hidden(batch_size)
```

- This is usually a tensor of zeros at the start.

Step 4: Feed a Batch of Inputs

Step 5: Calculate the Loss

Step 6: Backpropagate the Loss

Step 7: Update the Model Weights

Step 8: Repeat for Many Epochs

RNNs maintain a hidden state (or memory) that gets updated at each time step based on the input sequence, and it needs to be initialized before processing the first batch.

Code-along

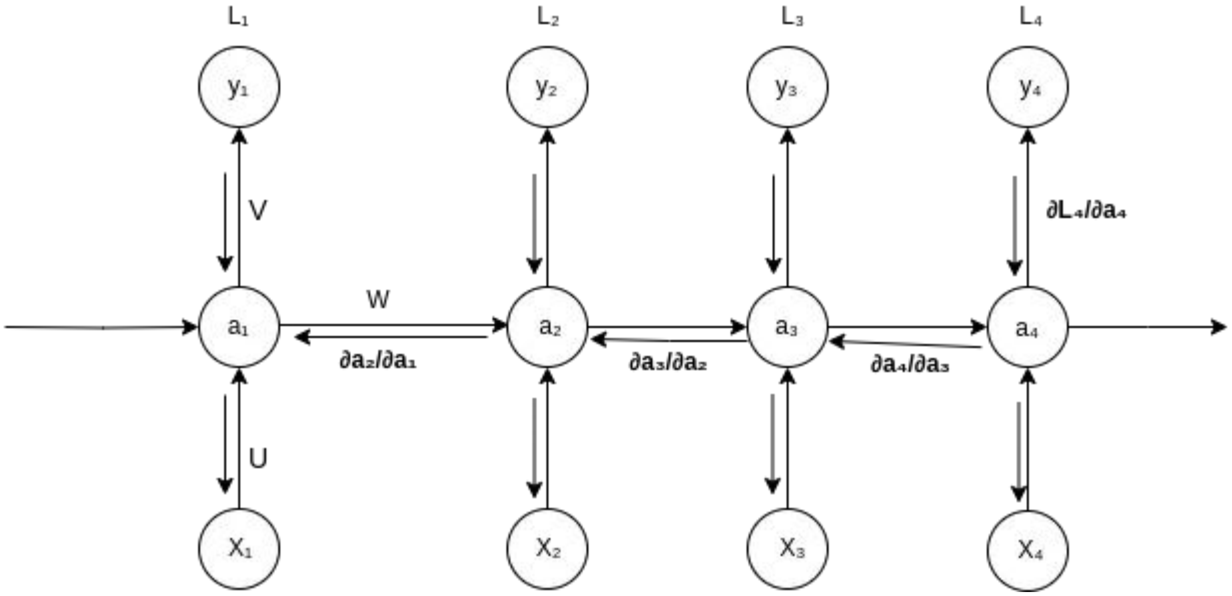
Jupyter Notebook

Part 1: Recurrent Neural Networks for Text Generation

Backpropagation Through Time

The key difference is that the gradients are "propagated back in time" across multiple time steps, not just one layer like in standard networks.

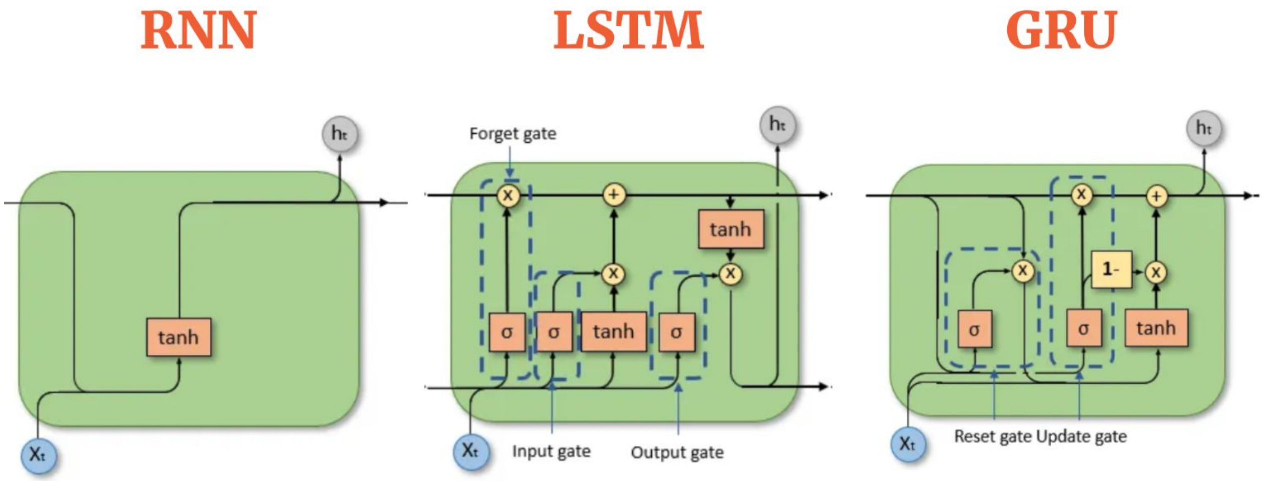
This allows the RNN to learn from long sequences, but it also means that it's harder to train on long sequences due to issues like vanishing or exploding gradients.



Source link: Medium article (RNN Architecture Explained)

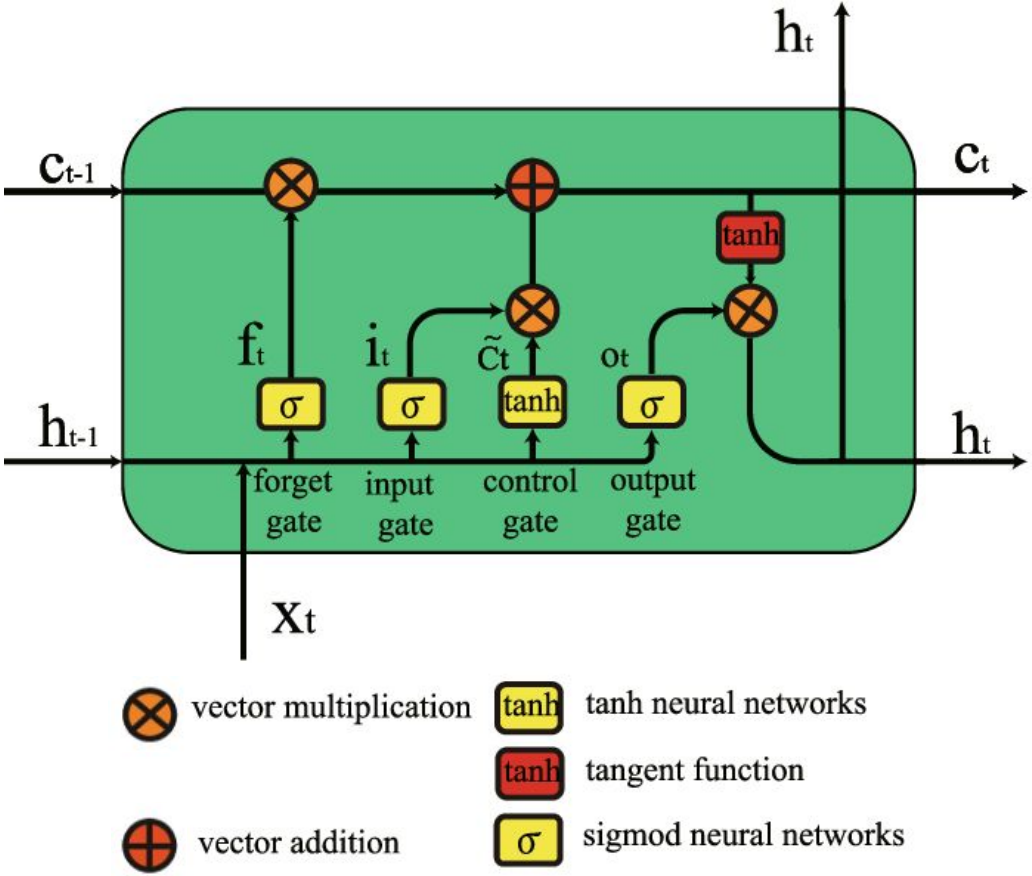
RNN Variations

LSTM is better than RNN because it can remember long-term dependencies with its memory cells, reducing the vanishing gradient problem, while GRU is better than RNN because it uses fewer gates and is computationally more efficient while still handling long-term dependencies well.

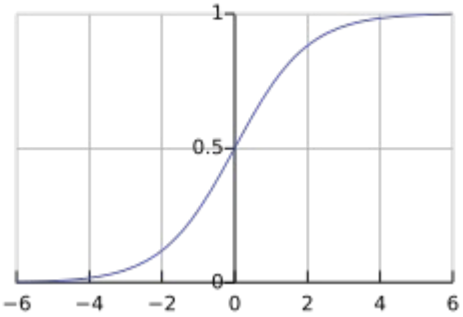


[Source Article: Sequence Models]

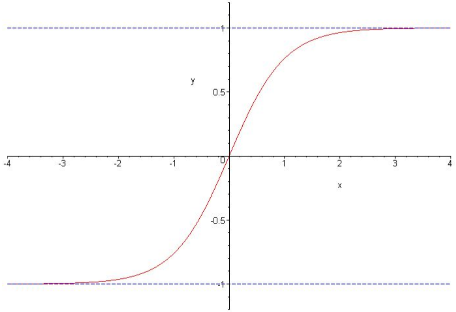
LSTM Cell



Sigmoid



tanh



Transformers

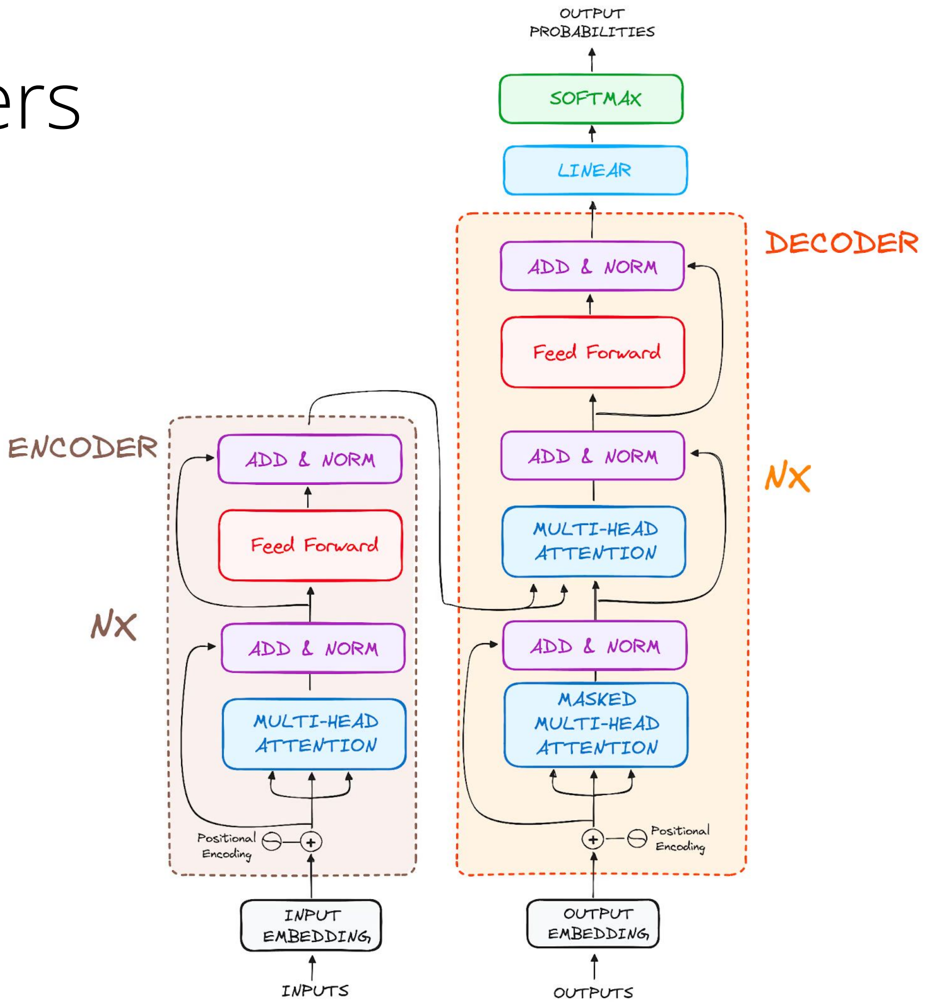
Positional Encoding

Self and Multi-Headed Attention

Encoder-Decoder

https://www.datacamp.com/tutorial/how-transformers-work

Transformers



How do Transformers process language*

"The fisherman walked to the bank across the river bank to deposit his earnings."

1 Word Embeddings

Each word activates its learned semantic representation.
<u>"bank"</u> (first) → Vector encoding financial institution concept
<u>"bank"</u> (second) → Same base vector as first "bank"

2 Position Encoding

Position information is added to help the model understand sequence order:
<u>"bank"</u> (position 6) gets positional encoding #6
<u>"bank"</u> (position 9, in "river bank") gets positional encoding #9

3 Attention (Self / Multihead)

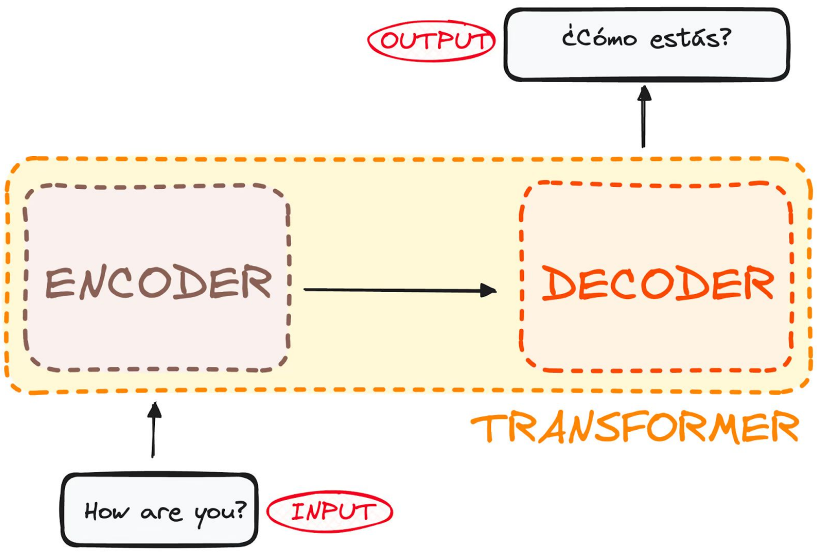
Learns relationships between words in text:
First "bank" strongly attends to "to" (destination/location)
Second "bank" strongly attends to "river" (context of 2nd bank)
"deposit" attends to "earnings" and the first "bank" (financial context)
"fisherman" attends to "walked" (subject-verb relationship)

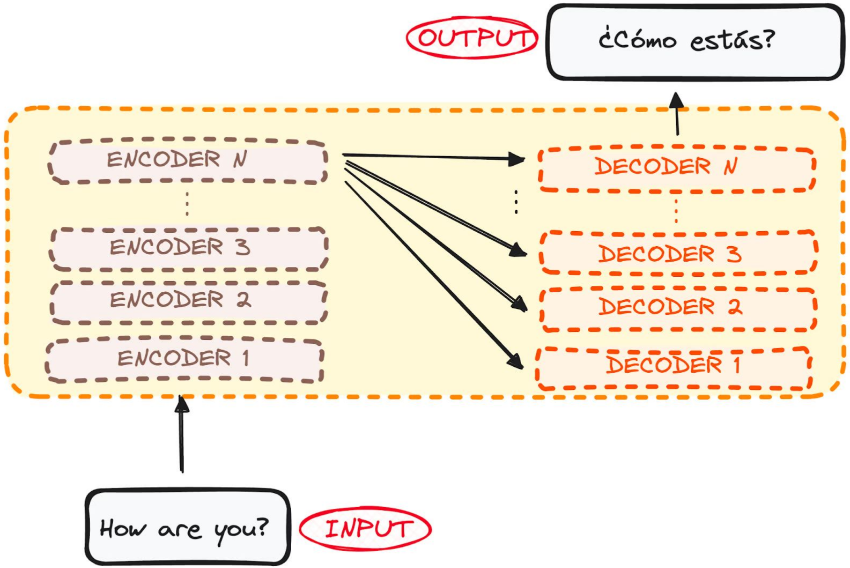
4 Encoder layers (multiple layers)

Applies incremental conceptual understanding (like "convolution" layers)
Early layers: Learn that "river bank" is a compound noun, "walked from" indicates movement
Middle layers: Disambiguate that first "bank" is financial (due to "deposit" and "earnings"), second "bank" is geographical (due to "river")
Later layers: Understand the narrative: a fisherman is going from a riverbank to a financial institution

* Intuitive illustration, does not fully describe mathematical workings

Transformers





Each layer learns different types of patterns, starting simple and getting more complex as you go deeper

Illustration / Simplification : "The bank by the river was steep."

- Encoder 1: Recognizes individual words - "bank," "river," "steep"
- Encoder 2: Understands "by the river" describes location
- Encoder 3: Realizes "bank" here means riverbank, not a financial institution
- Encoder N: Combines everything to understand the full meaning

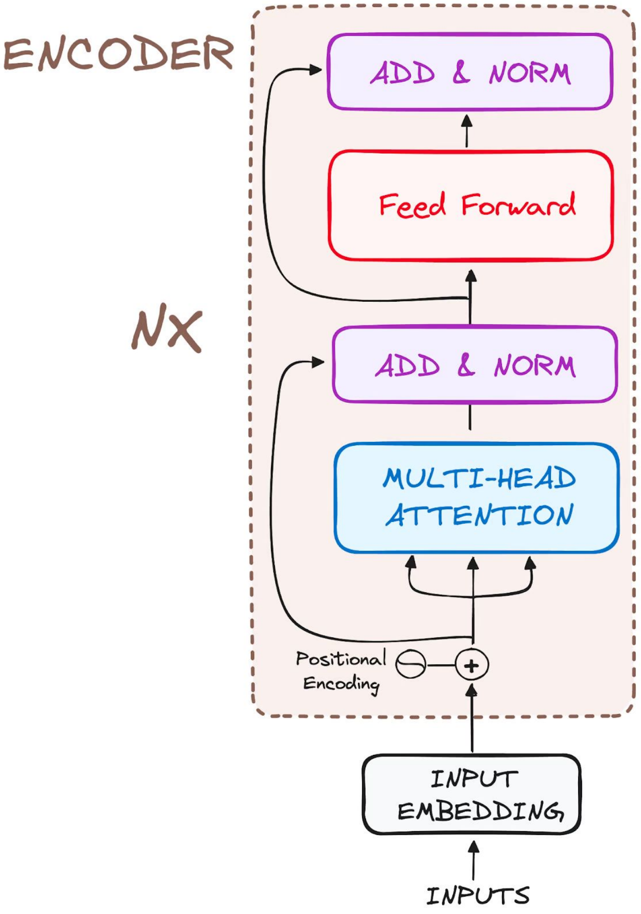
Transformers

Encoder Layer

- Purpose: The encoder reads the input data and turns it into a useful representation.
- It takes in the input sequence (like a sentence) and processes it one word at a time. But instead of just looking at one word in isolation, it looks at all the words at once to understand the relationships between them.

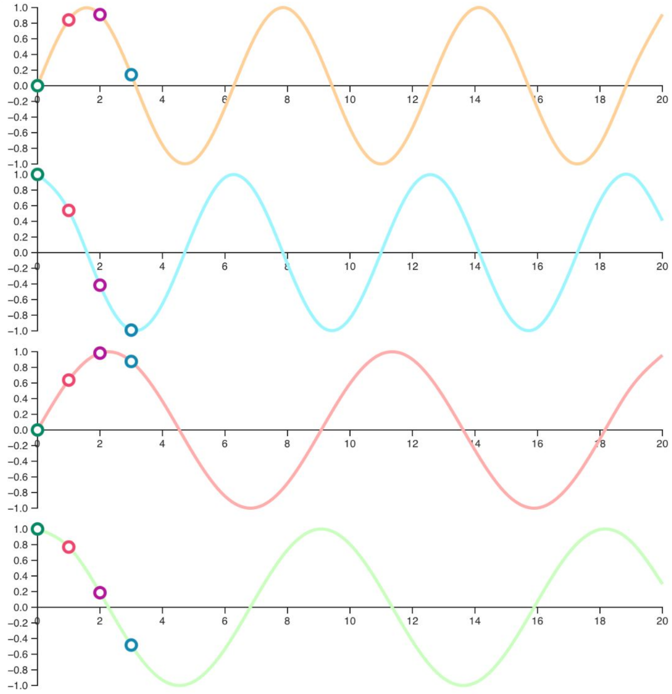
It does this in two steps:

1. **Self-attention:** It checks how each word is related to every other word in the sentence. This way, it can better understand the context of the words.
2. **Feed-forward network:** After that, it processes the information through a simple neural network to transform the data into something more useful for the next step.



Positional encoding

Positional encoding visualization



	p0	p1	p2	p3
i=0	0.000	0.841	0.909	0.141
i=1	1.000	0.540	-0.416	-0.990
i=2	0.000	0.638	0.983	0.875
i=3	1.000	0.770	0.186	-0.484

Positional Encoding

$$PE_{(pos,2i)} = sin(\frac{pos}{10000^{2i/d_{\text{model}}}})$$

$$PE_{(pos,2i+1)} = cos(\frac{pos}{10000^{2i/d_{\text{model}}}})$$

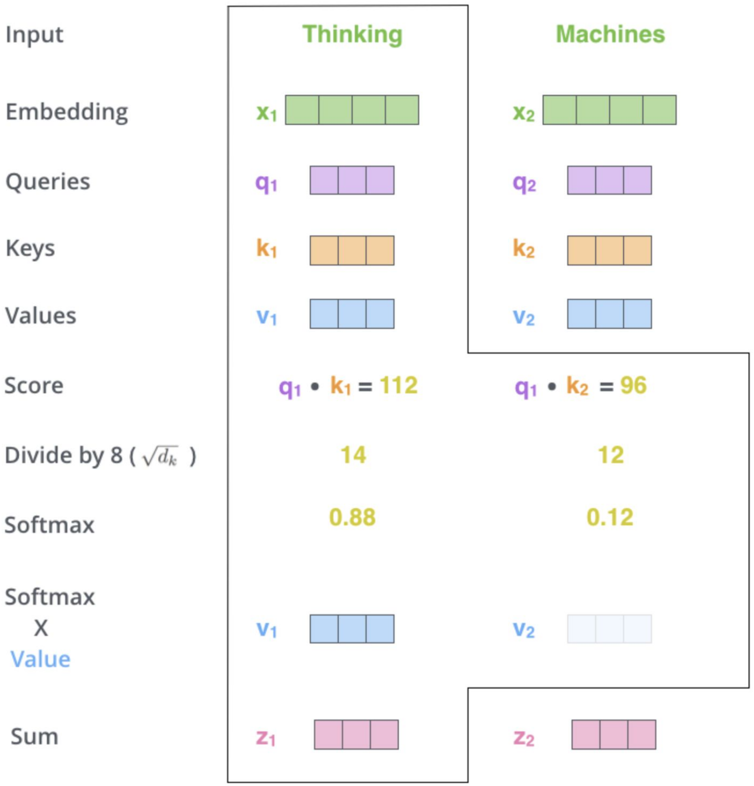
Settings: d = 50

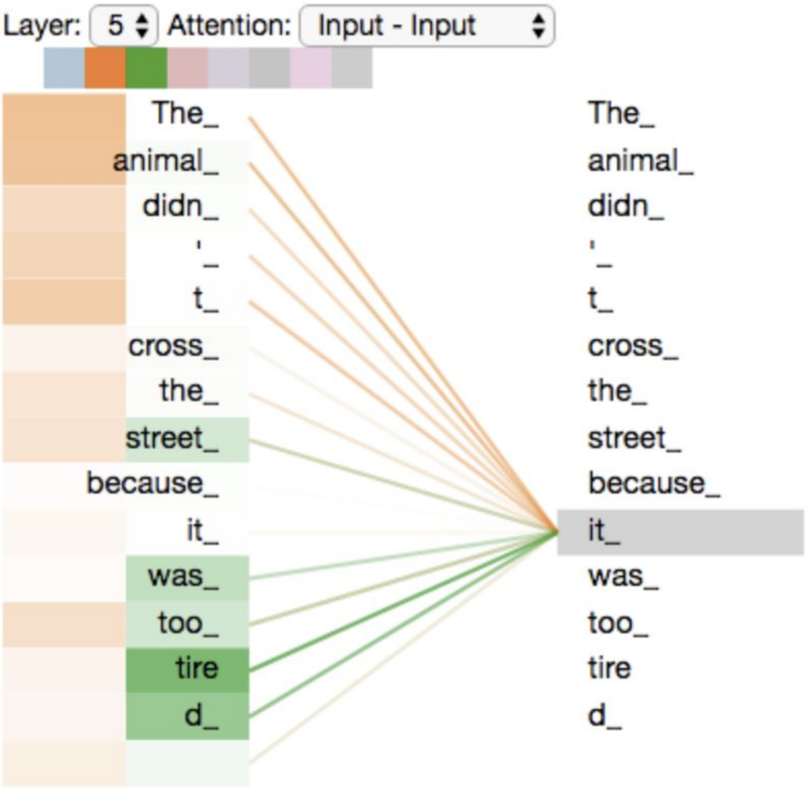
The value of each positional encoding depends on the *position* (*pos*) and *dimension* (*d*). We calculate result for every *index* (*i*) to get the whole vector.

pos0 pos1 pos2 pos3

https://erdem.pl/2021/05/understanding-positional-encoding-in-transformers

Self Attention (Query, Key, Value)





https://jalammar.github.io/illustrated-transformer/

Transformers

Decoder Layer

- Purpose: The decoder takes the output from the encoder and turns it into the final result.

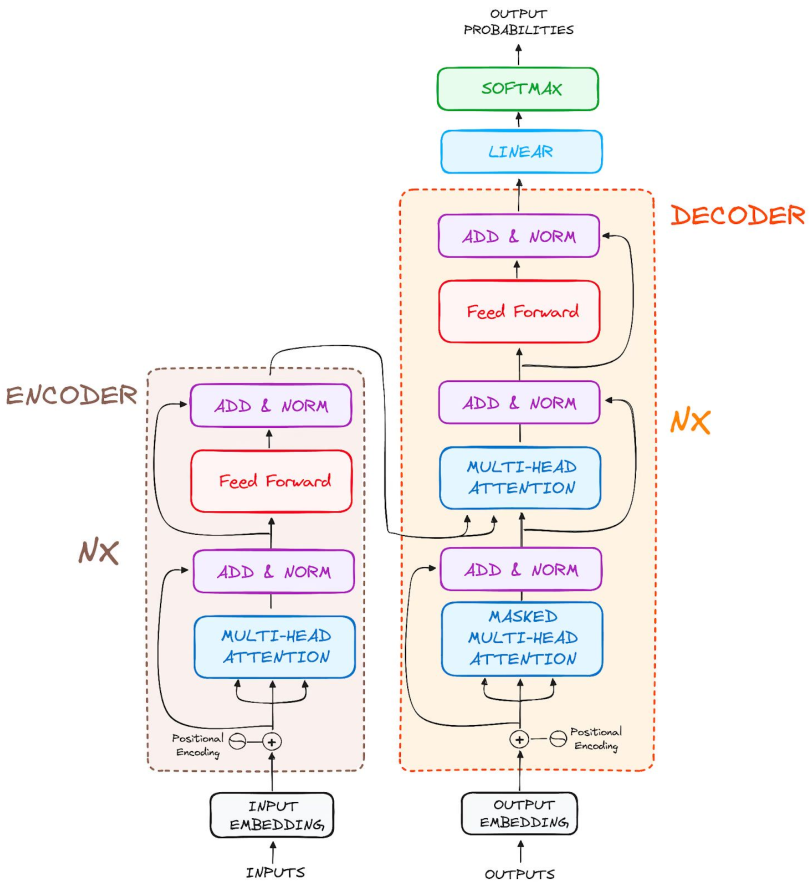
- It works similarly to the encoder, but in addition to the output from the encoder, it also uses the words it has already generated to create the next part of the output.

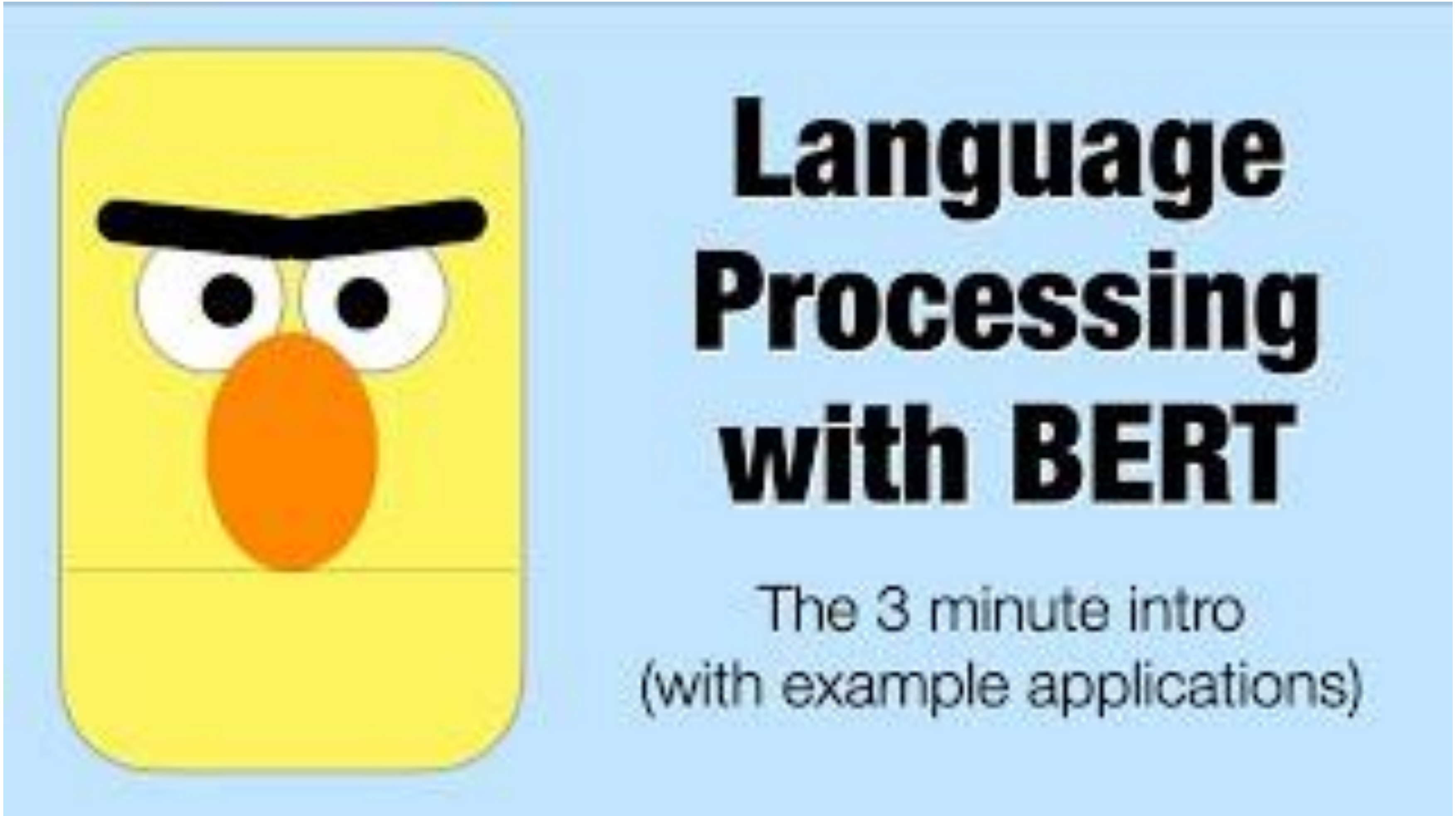
It has three steps:

1. **Self-attention:** Just like the encoder, it checks how the words in the output relate to each other.

2. **Encoder-decoder attention:** It looks at the encoder output to see which parts of the input sequence are most relevant for generating the next word.

3. **Feed-forward network:** Finally, it processes the information and generates the next word in the sequence.




Language
Processing
with BERT
The 3 minute intro
(with example applications)

BERT

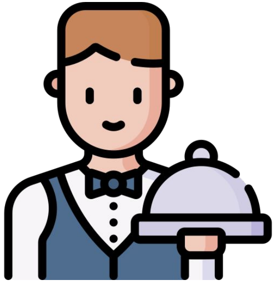
BERT (standing for Bidirectional Encoder Representations from Transformers) is an open-source model developed by Google in 2018

Google's release of BERT, an open-source natural language processing framework, revolutionized NLP with its unique bidirectional training, which enables the model to have more context-informed predictions about what the next word should be.

By understanding context from all sides of a word, BERT outperformed previous models in tasks like question-answering and understanding ambiguous language. Its core uses Transformers, connecting each output and input element dynamically.

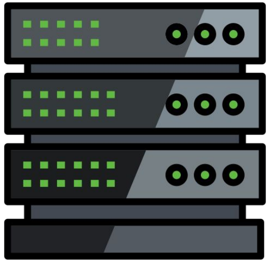
BERT, pre-trained on Wikipedia, excelled in various NLP tasks, prompting Google to integrate it into its search engine for more natural queries. This innovation sparked a race to develop advanced language models and significantly advanced the field's ability to handle complex language understanding.

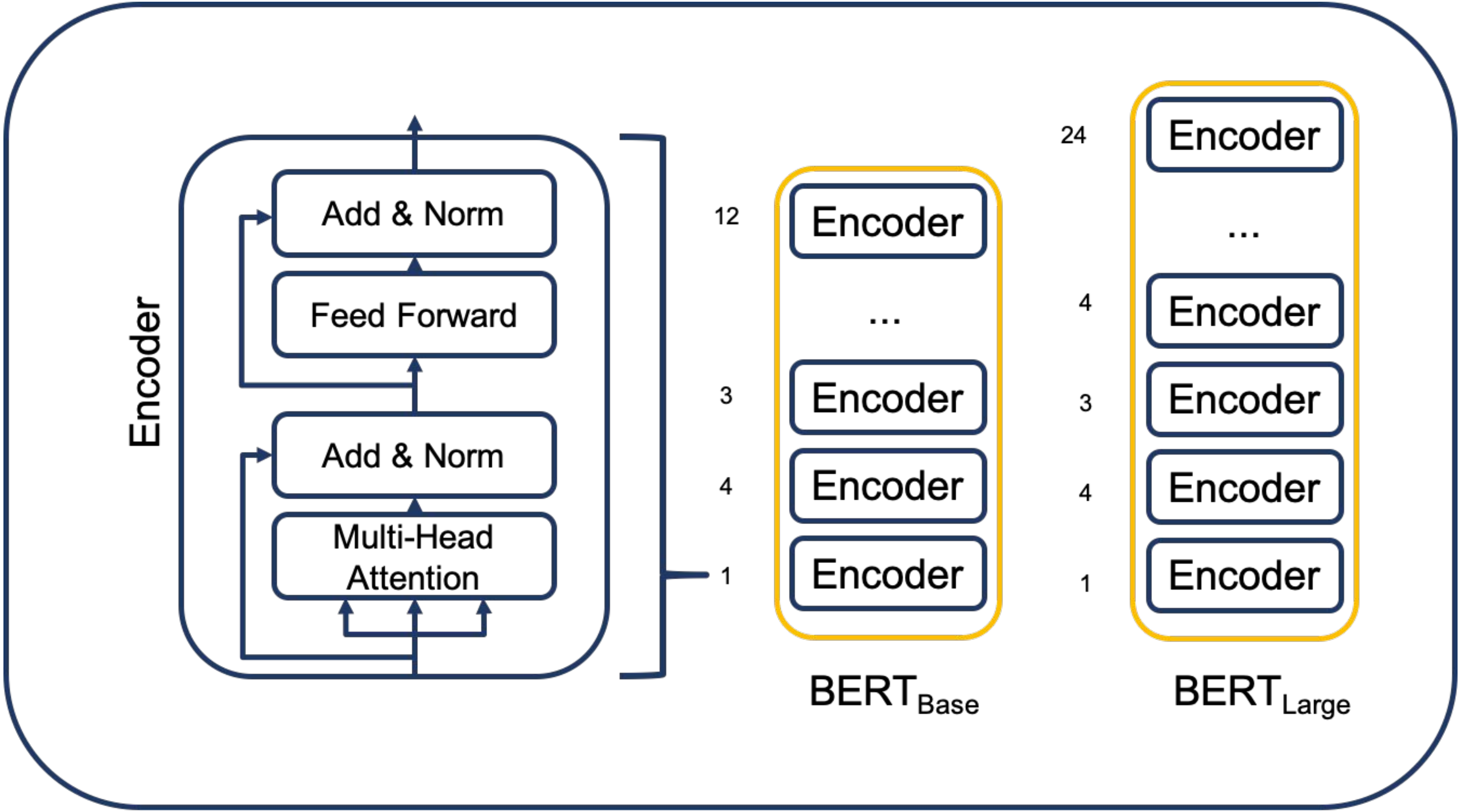
Bidirectional Encoder Representations



Server, can i have the check?

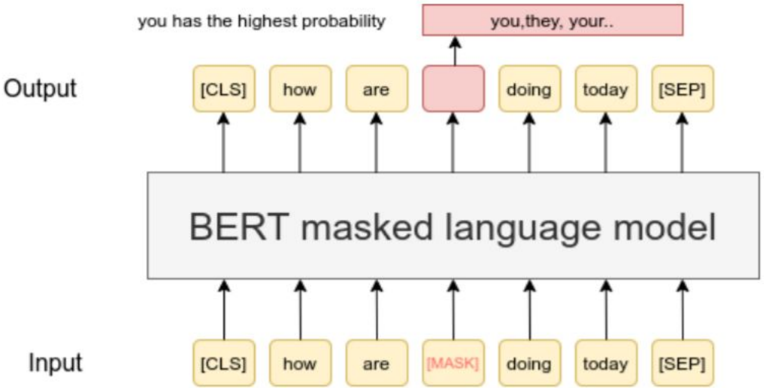
Ooops, the **server** just crashed.





https://pub.towardsai.net/demystifying-bert-efe9ac3c1c74

BERT Pretraining



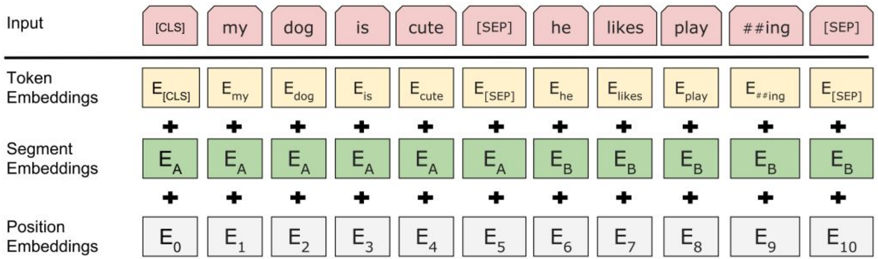


Figure 2: BERT input representation. The input embeddings is the sum of the token embeddings, the segmentation embeddings and the position embeddings.

Masked Language Modeling

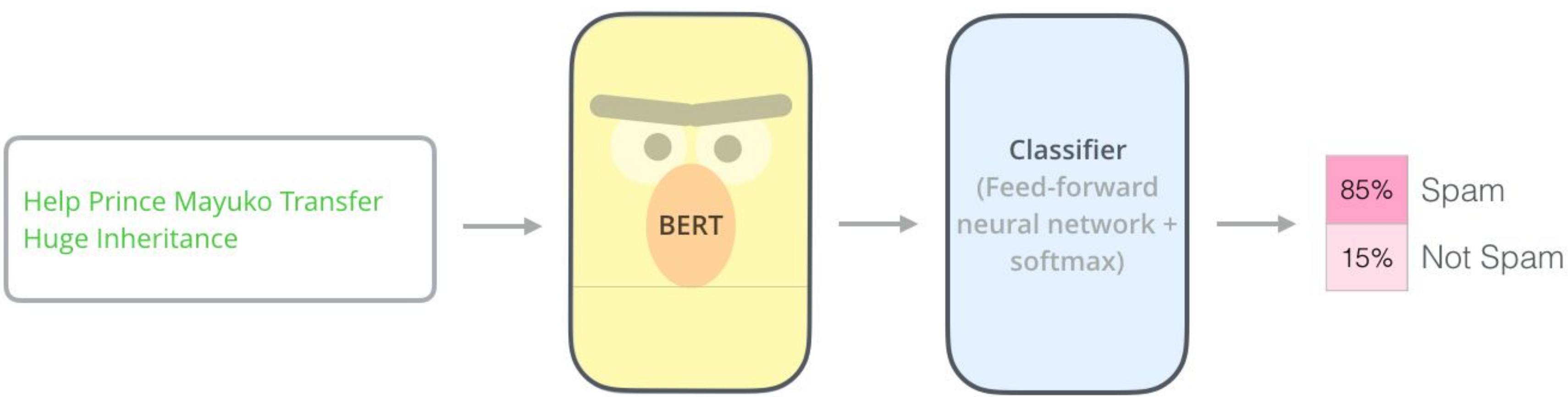
Next Sentence Prediction

Model		Parameters	Layers	Hidden	Embedding
	base	108M	12	768	768
BERT	large	334M	24	1024	1024
	xlarge	1270M	24	2048	2048

BERT in action - Finetuning

BERT has been successfully applied to many NLP tasks, including:

- Text classification
- Question answering
- Named entity recognition
- Sentiment analysis



https://jalammar.github.io/illustrated-bert/

BERT Model Variants: Comparison & Use Cases

BERT

BASE MODEL
Bidirectional Encoder Representations from Transformers (2018)

PARAMETERS
BERT-base: 110M | BERT-large: 340M

KEY FEATURES
- Pre-trained on MLM and NSP
- Bidirectional context
- Original transformer architecture

STRENGTHS
- Strong baseline performance
- Well-documented
- Extensive support

LIMITATIONS
- Static masking
- NSP adds complexity
- Slower training

WHEN TO USE
- General-purpose NLP

RoBERTa

FULL NAME
Robustly Optimized BERT Approach (2019)

PARAMETERS
RoBERTa-base: 125M | RoBERTa-large: 355M

KEY FEATURES
- Dynamic masking patterns
- Removes NSP task
- Trained on 10× more data

STRENGTHS
- Best overall accuracy
- Improved training
- Better generalization

LIMITATIONS
- Higher compute needs
- Longer training
- Larger model size

WHEN TO USE
- Max accuracy priority

DistilBERT

FULL NAME
Distilled BERT (2019)

PARAMETERS
66M (40% smaller than BERT-base)

KEY FEATURES
- Knowledge distillation
- 6 layers vs 12
- 97% of BERT performance
- 60% faster inference

STRENGTHS
- Great speed/accuracy trade-off
- Lower memory footprint
- Fast inference
- Production-ready

LIMITATIONS
- 3% accuracy reduction
- Not for complex tasks
- Less context depth

WHEN TO USE
- Real-time apps
- Speed critical

ALBERT

FULL NAME
A Lite BERT (2019)

PARAMETERS
ALBERT-base: 12M | xxlarge: 235M

KEY FEATURES
- Factorized embeddings
- Cross-layer sharing
- SOP instead of NSP
- Reduced parameters

STRENGTHS
- Smallest param count
- Memory efficient
- Scales well
- Good despite size

LIMITATIONS
- Slower than DistilBERT
- Sharing adds compute
- May underperform

WHEN TO USE
- Very limited memory

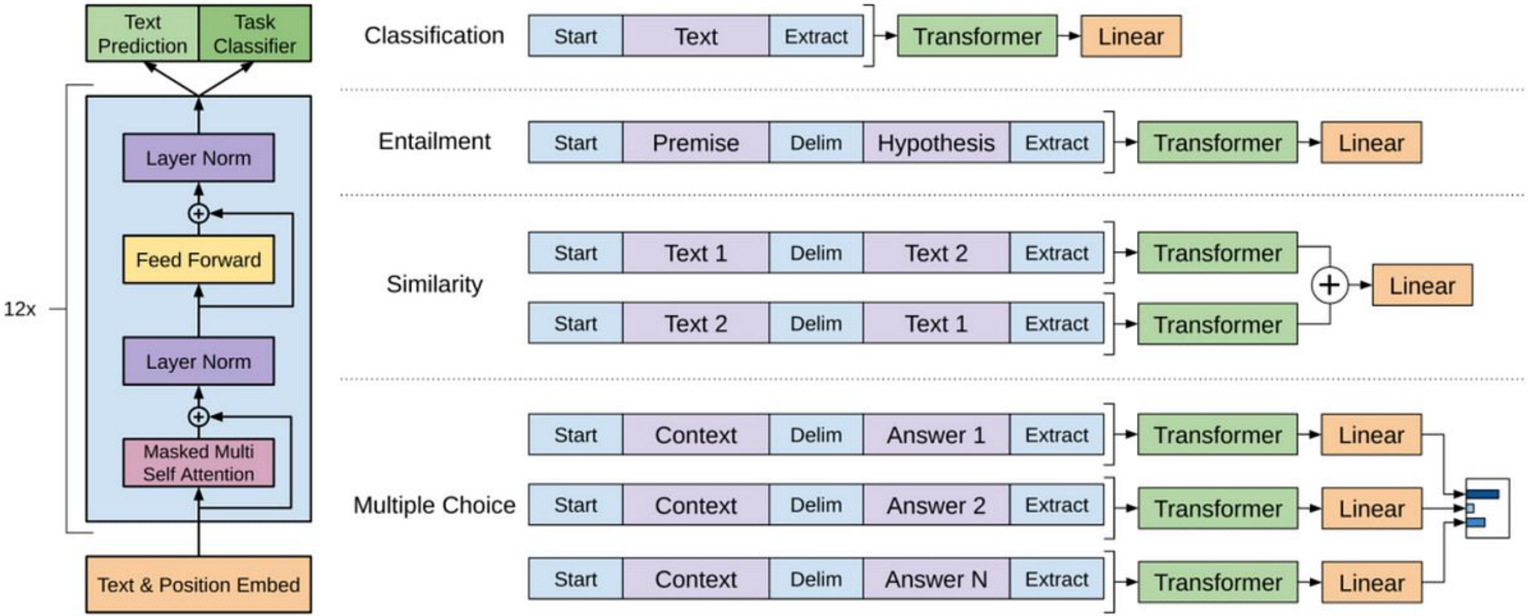
Code-along

Jupyter Notebook

Part 2: Sentiment Analysis and NER with BERT

GPT - Generative Pretrained Transformer

The GPT (Generative Pretrained Transformer) models are a series of deep learning models developed by OpenAI. These models are designed to generate human-like text by predicting the next word in a sequence of words. The architecture of GPT is similar to the decoder part of the Transformer model. It uses masked self-attention, where each token can only attend to previous tokens in the self-attention layers of the transformer.

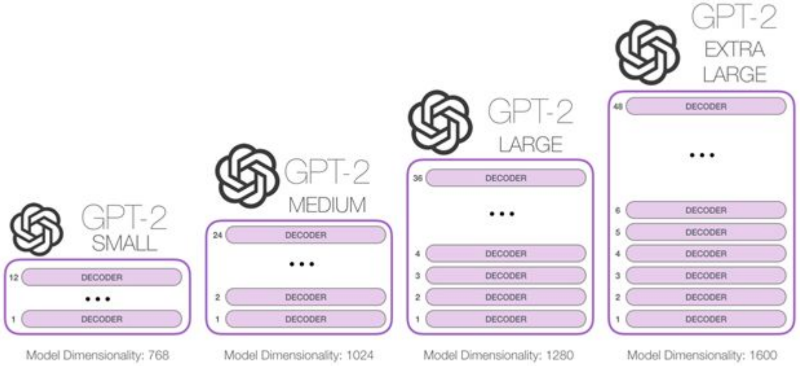


GPT-2

We've trained a large-scale unsupervised language model which generates coherent paragraphs of text, achieves state-of-the-art performance on many language modeling benchmarks, and performs rudimentary reading comprehension, machine translation, question answering, and summarization—all without task-specific training.

Our model, called GPT-2 (a successor to GPT), was trained simply to predict the next word in 40GB of Internet text. Due to our concerns about malicious applications of the technology, we are not releasing the trained model. As an experiment in responsible disclosure, we are instead releasing a much smaller model for researchers to experiment with, as well as a technical paper.

GPT-2 is a large transformer-based language model with 1.5 billion parameters, trained on a dataset[A] of 8 million web pages. GPT-2 is trained with a simple objective: predict the next word, given all of the previous words within some text. The diversity of the dataset causes this simple goal to contain naturally occurring demonstrations of many tasks across diverse domains. GPT-2 is a direct scale-up of GPT, with more than 10X the parameters and trained on more than 10X the amount of data.



https://openai.com/index/better-language-models/#sample1

GPT Evolution

Key Features Across All GPT Models:

- **Transformer Architecture:** All GPT models use the Transformer architecture, which allows for better handling of sequential data and long-range dependencies in text.
- **Pretraining and Fine-tuning:** These models are pre-trained on large datasets and can then be fine-tuned for specific tasks, making them highly adaptable.
- **Language Generation:** The models are generative, meaning they create text based on patterns and context learned during training.

https://platform.openai.com/tokenizer

https://medium.com/@vipul.koti333/evolution-of-gpt-models-gpt-1-to-gpt-4-0238ee07a29b

Feature	GPT-1	GPT-2	GPT-3	GPT-4
Year	2018	2019	2020	2023
Paper Title	"Improving Language Understanding by Generative Pre-training"	"Language Models are Unsupervised Multitask Learners"	"Language Models are Few-Shot Learners"	"GPT-4 Technical Report"
Parameters	117M	1.5B	175B	Undisclosed but larger than GPT-3
Architecture	12-layer transformer decoder	48-layer transformer decoder	96-layer transformer	Optimized transformer architecture, multimodal capabilities
Context Window	512 tokens	1024 tokens	2048 tokens	Up to 128k tokens depending on model
Key Objective	Generative pre-training for language modeling	Task conditioning for multitask learning	In-context learning and few-shot learning	Multimodal capabilities and advanced reasoning
Training Dataset	BooksCorpus	WebText (40GB from 8M documents)	Common Crawl, WebText2, Books1, Books2, Wikipedia	Multimodal datasets (text + images), higher quality and scale
Zero-Shot Learning	Limited	Improved over GPT-1	Major advancement, effective in many tasks	Further enhanced with stronger reasoning and visual input

Code-along

Jupyter Notebook

Part 3: Text Generation with GPT-2

End of Lesson - Exit Ticket

Survey Link

https://www.menti.com/al2yt8p1zw8i