Ministère de l'Enseignement Supérieur
et de la Recherche Scientifique
*** * ***
Université de Carthage
*** * ***
**Institut National des Sciences
Appliquées et de Technologie**

Foundations of Language Artificial Intelligence : A Guide to NLP, LLMs, and Recurrent Neural Networks"

- Presented by : Malek Zitouni

- Academic Year : 2024-2025

Book 1: Attention Is All You Need

The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks show these models to be superior in quality while being more parallelizable and requiring significantly less time to train.

Recurrent neural networks, long short-term memory and gated recurrent neural networks in particular, have been firmly established as state of the art approaches in sequence modeling and transduction problems such as language modeling and machine translation . Numerous efforts have since continued to push the boundaries of recurrent language models and encoder-decoder architectures.Recurrent models typically factor computation along the symbol positions of the input and output sequences. Aligning the positions to steps in computation time, they generate a sequence of hidden states ht, as a function of the previous hidden state ht−1 and the input for position t. This inherently sequential nature precludes parallelization within training examples, which becomes critical at longer sequence lengths, as memory constraints limit batching across examples. Recent work has achieved significant improvements in

computational efficiency through factorization tricks and conditional computation , while also improving model performance in case of the latter. The fundamental constraint of sequential computation, however, remains.Attention mechanisms have become an integral part of compelling sequence modeling and transduction models in various tasks, allowing modeling of dependencies without regard to their distance in the input or output sequences . In all but a few cases , however, such attention mechanisms are used in conjunction with a recurrent network.The Transformer, a model architecture eschewing recurrence and instead relying entirely on an attention mechanism to draw global dependencies between input and output. The Transformer allows for significantly more parallelization and can reach a new state of the art in translation quality after being trained for as little as twelve hours on eight P100 GPUs.The goal of reducing sequential computation also forms the foundation of the Extended Neural GPU , ByteNet and ConvS2S , all of which use convolutional neural networks as basic building block, computing hidden representations in parallel for all input and output positions. In these models, the number of operations required to relate signals from two arbitrary input or output positions grows in the distance between positions, linearly for ConvS2S and logarithmically for ByteNet. This makes it more difficult to learn dependencies between distant positions . In the Transformer this is reduced to a constant number of operations, albeit at the cost of reduced effective resolution due to averaging attention-weighted positions.Self-attention, sometimes called intra-attention is an attention mechanism relating different positions of a single sequence in order to compute a representation of

the sequence. Self-attention has been used successfully in a variety of tasks including reading comprehension, abstractive summarization, textual entailment and learning task-independent sentence representations.End-to-end memory networks are based on a recurrent attention mechanism instead of sequence aligned recurrence and have been shown to perform well on simple-language question answering and language modeling tasks.End-to-end memory networks are based on a recurrent attention mechanism instead of sequence aligned recurrence and have been shown to perform well on simple-language question answering and language modeling tasks

- **Model Architecture :**

Most competitive neural sequence transduction models have an encoder-decoder structure . Here, the encoder maps an input sequence of symbol representations (x1,...,xn) to a sequence of continuous representations z = (z1,...,zn). Given z, the decoder then generates an output sequence (y1,...,ym) of symbols one element at a time. At each step the model is auto-regressive , consuming the previously generated symbols as additional input when generating the next.

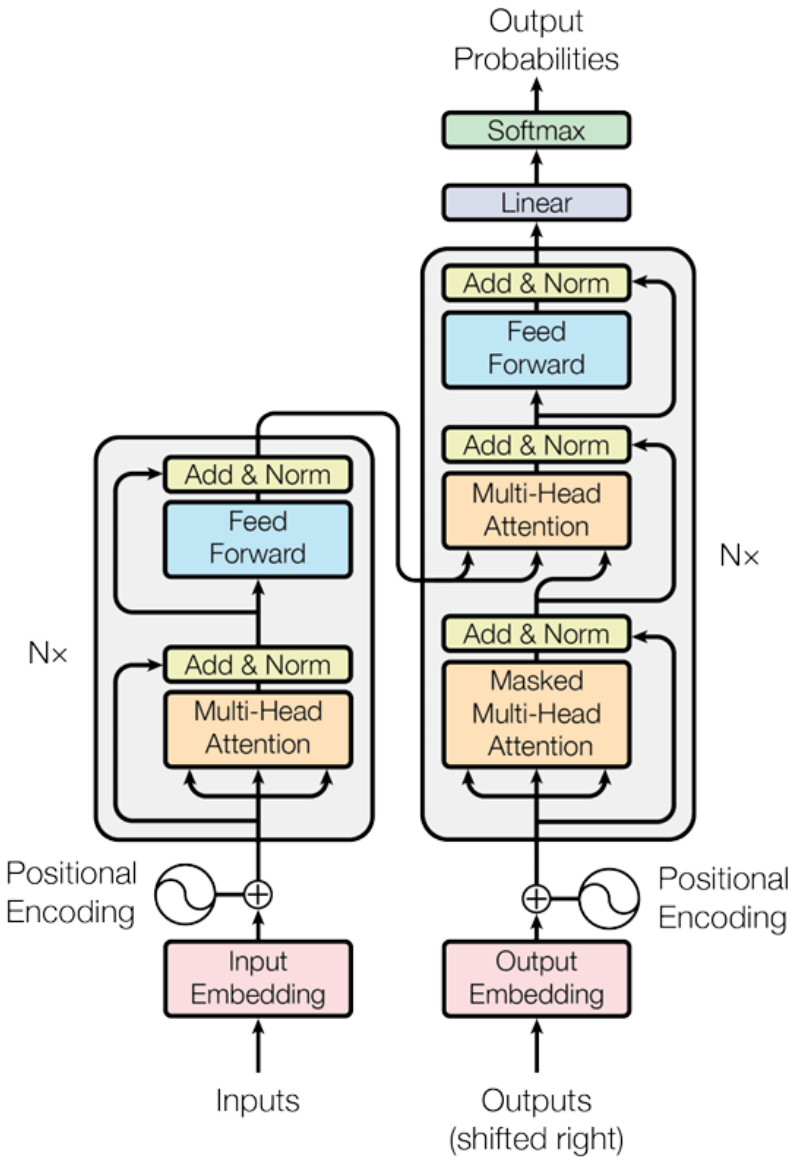


Figure 1: The Transformer - model architecture.

- **<u>Input Embedding :</u>**
    - The first step in the Transformer is to convert each word or token in the input sequence into a dense vector representation, known as an embedding.
    - This vector captures semantic information about the word and provides a fixed-size representation for each token.
    - The model does not operate on raw tokens (e.g., words or subwords) but rather on these dense embeddings.

- **Positional Encoder :**

    - Positional encoding is added to each input embedding to provide information about the order of tokens in the sequence.
    - Since Transformers do not have a built-in mechanism for tracking token order (like recurrent neural networks do), positional encodings allow the model to differentiate between words based on their position.
    - This encoding is typically achieved using sinusoidal functions or learned embeddings.
    - The positional encoding is then **added to** the input embeddings to form the final input to the encoder.

- **Encoder Block :**

    - The encoder is composed of multiple identical layers (typically six or twelve, represented as N×), each containing two main sub-layers: **Multi-Head Self-Attention** and **Feed Forward Network**.

        - **Multi-Head Self-Attention** :

            - This layer is responsible for allowing each token in the input sequence to pay attention to every other token, enabling the model to capture relationships between tokens.
            - **Self-attention** calculates attention scores by computing a dot product between the query (Q), key (K), and value (V) representations of the input tokens. This results in a weighted representation of the context for each token.
            - **Multi-head attention** involves running multiple self-attention layers in parallel, each focusing on different parts of the context. These multiple "heads" are then concatenated and linearly transformed.
            - This mechanism enables the model to capture different types of relationships and dependencies for each token in the sequence.

- **Add & Norm**

  - A residual connection (indicated by "Add") is added to the output of the multi-head attention layer, meaning the original input is added to the output.
  - The result is then passed through a layer normalization step ("Norm"), which stabilizes the network by normalizing activations, making training faster and improving model performance.

- **Feed Forward Network (FFN)**

  - This layer is a fully connected network applied independently to each position (token) in the sequence.It typically consists of two linear transformations with a ReLU activation in between. This network allows the model to learn complex transformations and mappings for each token. Like the multi-head attention layer, the output of the feed-forward network has a residual connection and layer normalization.

- **Stacking Encoder Layers :**

  - This entire process (multi-head attention, add & norm, feed forward, add & norm) is repeated across multiple layers.
  - Each layer further refines the token representations, enabling the model to build increasingly complex representations of the input sequence.

- Resume :

=> The encoder is composed of a stack of N = 6 identical layers. Each layer has two sub-layers. The first is a multi-head self-attention mechanism, and the second is a simple, position wise fully connected feed-forward network. We employ a residual connection around each of the two sub-layers, followed by layer normalization]. That is, the output of each sub-layer is LayerNorm(x + Sublayer(x)), where Sublayer(x) is the function implemented by the sub-layer itself. To facilitate these residual connections, all sub-layers in the model, as well as the embedding layers, produce outputs of dimension dim = 512.

- **Decoder Block** :

  - Each decoder layer has three main sub-layers: **Masked Multi-Head Self-Attention**, **Multi-Head Attention** (which attends to the encoder outputs), and a **Feed Forward Network**.

    - **Masked Multi-Head Self-Attention**

- In the decoder, the self-attention mechanism is masked to prevent attending to future tokens. This ensures that each position in the output sequence can only depend on previous tokens, which is crucial for autoregressive generation (predicting one token at a time).
Like in the encoder, this layer includes multiple heads for capturing diverse relationships.

- **Multi-Head Attention (with Encoder Output) :**

- After the masked self-attention, the next attention layer in the decoder attends to the encoder outputs, allowing each token in the target sequence to attend to all tokens in the input sequence.
- This encoder-decoder attention mechanism enables the decoder to focus on relevant parts of the input sentence while generating each word in the output.
- This layer operates similarly to self-attention but takes queries from the decoder and keys/values from the encoder output.

- **Add & Norm (for both attention layers and feed-forward)**

- Like in the encoder, each attention layer and the feed-forward network in the decoder are followed by residual connections and layer normalization steps.

- **Feed Forward Network (FFN)**

- Similar to the encoder, each decoder layer also has a feed-forward network that processes each token position independently, enabling further non-linear transformations.

- **Stacking Decoder Layers**

- These three sub-layers (masked attention, encoder-decoder attention, feed-forward) are stacked across multiple layers, allowing the model to capture more sophisticated dependencies for generating the output sequence.

- **Output Embedding and Shifted Right**

- In the decoder, the output tokens are also passed through an embedding layer, which provides dense representations similar to the input embeddings.
- The output sequence is **shifted right**, meaning the model sees the output sequence up to the current position for predicting the next token. This is essential for autoregressive generation.
- Positional encodings are added to the output embeddings to retain the positional information in the target sequence.

- **Linear and Softmax Layers**

- After passing through the final decoder layer, the output goes through a **linear layer**, which projects the decoder's final hidden states into the vocabulary size.

- This projection generates a score for each word in the vocabulary, representing the likelihood of each word being the next token in the sequence.

- Finally, a **softmax layer** is applied to convert these scores into probabilities, indicating the most likely next word in the sequence.

- The model then chooses the word with the highest probability (or applies beam search or sampling strategies) to generate the next token in the output.

- **Output Probabilities**

- The final output probabilities represent the likelihoods for each token in the vocabulary as the next word in the output sequence.
- During training, these probabilities are compared to the actual target tokens to calculate loss and update model weights.
- During inference, the decoder generates output tokens one at a time until it produces the end-of-sequence token, completing the output sentence.

- Resume :

The decoder is composed of a stack of N = 6 identical layers. In addition to the two sub-layers in each encoder layer, the decoder inserts a third sub-layer, which performs multi-head attention over the output of the encoder stack. Similar to the encoder, we employ residual connections around each of the sub-layers, followed by layer normalization. We also modify the self-attention sub-layer in the decoder stack

to prevent positions from attending to subsequent positions. This masking, combined with the fact that the output embeddings are offset by one position, ensures that the predictions for position i can depend only on the known outputs at positions less than

i .

=> An attention function can be described as mapping a query and a set of key-value pairs to an output, where the query, keys, values, and output are all vectors. The output is computed as a weighted sum of the values, where the weight assigned to each value is computed by a compatibility function of the query with the corresponding key

Key Characteristics of Autoregressive Generation

Autoregressive generation is a process where a model generates a sequence of outputs one step at a time, using previously generated tokens as part of the input to predict the next token. This approach is particularly useful in tasks like **text generation, machine translation, and language modeling**, where each output token depends on the context provided by the previous tokens in the sequence.

In simpler terms, autoregressive models produce text by **predicting the next word (or token) based on the preceding words** they have already generated. They rely on **sequential, step-by-step predictions**, where each new prediction depends on the cumulative context of previous predictions.

1. **Sequential Prediction**:
    - The model generates text one token at a time. For instance, to generate a sentence, it first generates the first word, then uses that word to predict the second word, and so on until the end of the sentence.
2. **Dependency on Previous Outputs**:
    - Each token prediction depends on all previously generated tokens. This "left-to-right" approach allows the model to condition the next token on the context built up from the sequence so far.
3. **Self-Feeding of Predictions**:
    - Once the model generates a token, it takes that token as part of the input for predicting the next one. This self-feeding mechanism continues until a specified end condition is met, like reaching an end-of-sequence token or hitting a maximum length.

Autoregressive Generation in Transformers

In the Transformer architecture (specifically the **decoder** part), autoregressive generation is achieved with the following mechanisms:

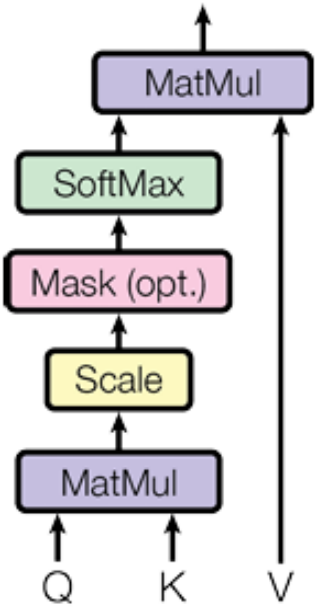
1. **Masked Self-Attention**:
    - To ensure that each token in the sequence is generated based only on previous tokens (not future ones), the Transformer's decoder applies **masked self-attention**. This mask blocks each position in the sequence from attending to future tokens, enforcing the left-to-right generation order.
    - This means when generating the third token, the model can only "see" the first and second tokens, but not any future tokens.
2. **Shifted Right Output**:
    - During training, the output sequence is "shifted right." This means that the model learns to predict the next token in the sequence, not the current token it has already seen.
    - For example, if the sentence is "I love cats," the input to the decoder would be "I love" and the model would be trained to predict "cats" next.
3. **End of Sequence Token**:
    - During generation, the model continues generating tokens one at a time until it produces a special **end-of-sequence (EOS)** token. This token indicates that the sentence or sequence has ended and stops the generation process.

Examples of Autoregressive Generation in Different Tasks

1. **Text Generation**:

    - For generating new text, an autoregressive model like GPT or a Transformer decoder starts with a prompt (e.g., "Once upon a time") and generates the next word based on the prompt, then the next word based on the previous two words, and so on, until a coherent story is formed.

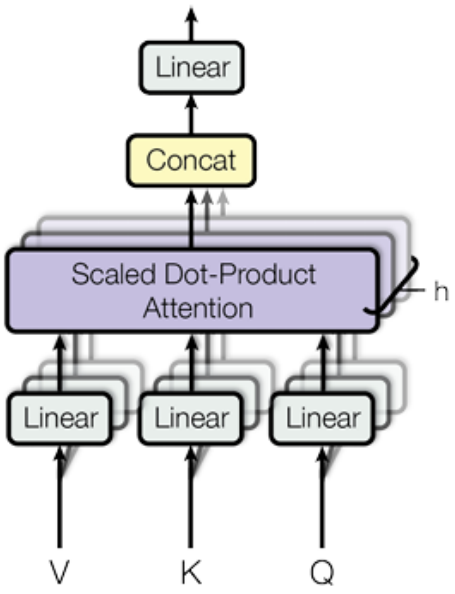
- Scaled Dot-Product Attention :



- The input consists of queries and keys of dimension dk, and values of dimension dv. We compute the dot products of the query with all keys, divide each by √dk, and apply a softmax function to obtain the weights on the values. In practice, we compute the attention function on a set of queries simultaneously, packed together into a matrix Q. The keys and values are also packed together into matrices K and V . We compute the matrix of outputs as:

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

- **Multi-Head Attention**



Instead of performing a single attention function with dmodel-dimensional keys, values and queries, we found it beneficial to linearly project the queries, keys and values h times with different, learned linear projections to dk, dk and dv dimensions, respectively. On each of these projected versions of queries, keys and values we then perform the attention function in parallel, yielding dv-dimensional

- The Transformer uses multi-head attention in three different ways: • In "encoder-decoder attention" layers, the queries come from the previous decoder layer, and the memory keys and values come from the output of the encoder. This allows every position in the decoder to attend over all positions in the input sequence. This mimics the typical encoder-decoder attention mechanisms in sequence-to-sequence models • The encoder contains self-attention layers. In a self-attention layer all of the keys, values and queries come from the same place, in this case, the output of the previous layer in the encoder. Each position in the encoder can attend to all positions in the previous layer of the encoder. • Similarly, self-attention layers in the decoder allow each position in the decoder to attend to all positions in the decoder up to and including that position. We need to prevent leftward information flow in the decoder to preserve the auto-regressive property. We implement this inside of scaled dot-product attention by masking out (setting to $-\infty$) all values in the input of the softmax which correspond to illegal connections

- Position-wise Feed-Forward Networks :

In addition to attention sub-layers, each of the layers in our encoder and decoder contains a fully connected feed-forward network, which is applied to each position separately and identically. This consists of two linear transformations with a ReLU activation in between.

$$\mathrm{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2$$

Table 1: Maximum path lengths, per-layer complexity and minimum number of sequential operations for different layer types. n is the sequence length, d is the representation dimension, k is the kernel size of convolutions and r the size of the neighborhood in restricted self-attention.

Layer Type	Complexity per Layer	Sequential Operations	Maximum Path Length
Self-Attention	$O(n^2 \cdot d)$	$O(1)$	$O(1)$
Recurrent	$O(n \cdot d^2)$	$O(n)$	$O(n)$
Convolutional	$O(k \cdot n \cdot d^2)$	$O(1)$	$O(log_k(n))$
Self-Attention (restricted)	$O(r \cdot n \cdot d)$	$O(1)$	$O(n/r)$

- **Training Data and Batching**

- The model was trained on the standard WMT 2014 English-German dataset, which contains approximately 4.5 million sentence pairs. Sentences were encoded using byte-pair encoding (BPE), resulting in a shared source-target vocabulary of around 37,000 tokens. For English-French, the model used the larger WMT 2014 English-French dataset, consisting of 36 million sentence pairs. Tokens were split into a 32,000-word-piece vocabulary. Sentence pairs were grouped into batches based on approximate sequence length, with each batch containing around 25,000 source tokens and 25,000 target tokens.

- **About Dataset :**

The WMT 2014 English-German dataset is a cornerstone resource for researchers developing and evaluating machine translation (MT) systems. It's widely used in the annual WMT shared task, serving as a standard benchmark to compare different approaches and track progress in the field.

Key Features

- **Size: 4.5 million parallel sentence pairs, providing ample data for training and testing MT models**
- **Origin: Comprises high-quality news articles from Europarl, News Commentary, and TED Talks, offering realistic and diverse text domains.**

- **Preprocessing: Cleaned and normalized for consistency, ensuring model compatibility and training efficiency.**
- **Task Diversity: Originally used for the WMT 2014 News Translation Task, but applicable to various MT research areas**

=====> The Transformer, the first sequence transduction model based entirely on attention, replaced the recurrent layers traditionally used in encoder-decoder architectures with multi-head self-attention. For translation tasks, the Transformer can be trained significantly faster than architectures based on recurrent or convolutional layers. It achieved new state-of-the-art results on both the WMT 2014 English-to-German and WMT 2014 English-to-French translation tasks.

• Metric BLEU :

- BLEU (Bilingual Evaluation Understudy) is an algorithm for evaluating the quality of text which has been machine-translated from one natural language to another. Quality is considered to be the correspondence between a machine's output and that of a human: "the closer a machine translation is to a professional human translation, the better it is" – this is the central idea behind BLEU. BLEU was one of the first metrics to claim a high correlation with human judgements of quality, and remains one of the most popular automated and inexpensive metrics.
- Scores are calculated for individual translated segments—generally sentences—by comparing them with a set of good quality reference translations. Those scores are then averaged over the whole corpus to reach an estimate of the translation's overall quality. Neither intelligibility nor grammatical correctness are not taken into account.

Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [18]	23.75			
Deep-Att + PosUnk [39]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [38]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [9]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [32]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [39]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [38]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [9]	26.36	**41.29**	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$
Transformer (base model)	27.3	38.1	**$3.3 \cdot 10^{18}$**	
Transformer (big)	**28.4**	**41.8**	$2.3 \cdot 10^{19}$	

Article 2 : Transformers United

++ Foundation Models ++

-AI is undergoing a paradigm shift with the rise of models (e.g., BERT, DALL-E, GPT-3) trained on broad data (generally using self-supervision at scale) that can be adapted to a wide range of downstream tasks. We call these models foundation models

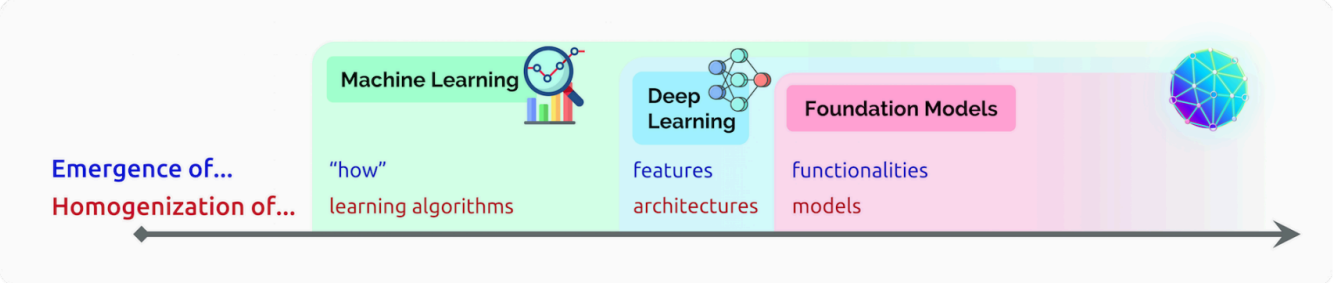


Fig. 1. The story of AI has been one of increasing *emergence* and *homogenization*. With the introduction of machine learning, *how* a task is performed emerges (is inferred automatically) from examples; with deep learning, the high-level features used for prediction emerge; and with foundation models, even advanced functionalities such as in-context learning emerge. At the same time, machine learning homogenizes learning algorithms (e.g., logistic regression), deep learning homogenizes model architectures (e.g., Convolutional Neural Networks), and foundation models homogenizes the model itself (e.g., GPT-3).

- Most AI systems today are powered by machine learning, where predictive models are trained on historical data and used to make future predictions. The rise of machine learning within AI started in the 1990s, representing a marked shift from the way AI systems were built previously: rather than specifying how to solve a task, a learning algorithm would induce it based on data — i.e., how it emerges from the dynamics of learning

- Around 2010, a revival of deep neural networks under the moniker of deep learning [LeCun et al. 2015] started gaining traction in the field of machine learning. Deep learning was fueled by larger datasets, more computation (notably, the availability of GPUs), and greater audacity. Deep neural networks would be trained on the raw inputs (e.g., pixels), and higher-level features would emerge through training (a

process dubbed “representation learning”). This led to massive performance gains on standard benchmarks, for example, in the seminal work of AlexNet [Krizhevsky et al. 2012] on the ImageNet dataset [Deng et al. 2009]. Deep learning also reflected a further shift towards homogenization: rather than having bespoke feature engineering pipelines for each application, the same deep neural network architecture could be used for many applications.

-Foundation models have taken shape most strongly in NLP, so we focus our story there for the moment. That said, much as deep learning was popularized in computer vision but exists beyond it, we understand foundation models as a general paradigm of AI, rather than specific to NLP in any way. By the end of 2018, the field of NLP was about to undergo another seismic change, marking the beginning of the era of foundation models. On a technical level, foundation models are enabled by transfer learning [Thrun 1998] and scale. The idea of transfer learning is to take the “knowledge” learned from one task (e.g., object recognition in images) and apply it to another task (e.g., activity recognition in videos). Within deep learning, pretraining is the dominant approach to transfer learning: a model is trained on a surrogate task (often just as a means to an end) and then adapted to the downstream task of interest via fine-tuning

-Transfer learning is what makes foundation models possible, but scale is what makes them powerful. Scale required three ingredients: (i) improvements in computer hardware — e.g., GPU throughput and memory have increased 10× over the last four years (systems); (ii) the development of the Transformer model architecture [Vaswani et al. 2017] that leverages the parallelism of the hardware to train much more expressive models than before(modeling); and (iii) the availability of much more training data

-The next wave of developments in self-supervised learning — BERT [Devlin et al. 2019] GPT-2 [Radford et al. 2019], RoBERTa [Liu et al. 2019], T5 [Raffel et al. 2019], BART [Lewis et al. 2020a] — quickly followed, embracing the Transformer architecture, incorporating more powerful deep bidirectional encoders of sentences, and scaling up to larger models and datasets. While one can view this last wave of technical developments purely through the lens of self supervised learning, there was a sociological inflection point around the introduction of BERT. Before 2019,

self-supervised learning with language models was essentially a subarea in NLP, which progressed in parallel to other developments in NLP. After 2019, self-supervised learning with language models became more of a substrate of NLP, as using BERT has become the norm. The acceptance that a single model could be useful for such a wide range of tasks marks the beginning of the era of foundation models. Foundation models have led to an unprecedented level of homogenization: Almost all state-of-the-art NLP models are now adapted from one of a few foundation models, such as BERT, RoBERTa, BART, T5, etc. While this homogenization produces extremely high leverage (any improvements in the foundation models can lead to immediate benefits across all of NLP), it is also a liability; all AI systems might inherit the same problematic biases of a few foundation models . Besides the homogenization of approaches, we also see the homogenization of actual models across research communities in the form of multimodal models — e.g., foundation models trained on language and vision data . Data is naturally multimodal in some domains—e.g., medical images, structured data, clinical text in healthcare . Thus, multimodal foundation models are a natural way of fusing all the relevant information about a domain, and adapting to tasks that also span multiple modes

Foundation models have also led to surprising emergence which results from scale. For example, GPT-3 [Brown et al. 2020], with 175 billion parameters compared to GPT-2's 1.5 billion, permits in-context learning, in which the language model can be adapted to a downstream task simply by providing it with a prompt (a natural language description of the task), an emergent property that was neither specifically trained for nor anticipated to arise

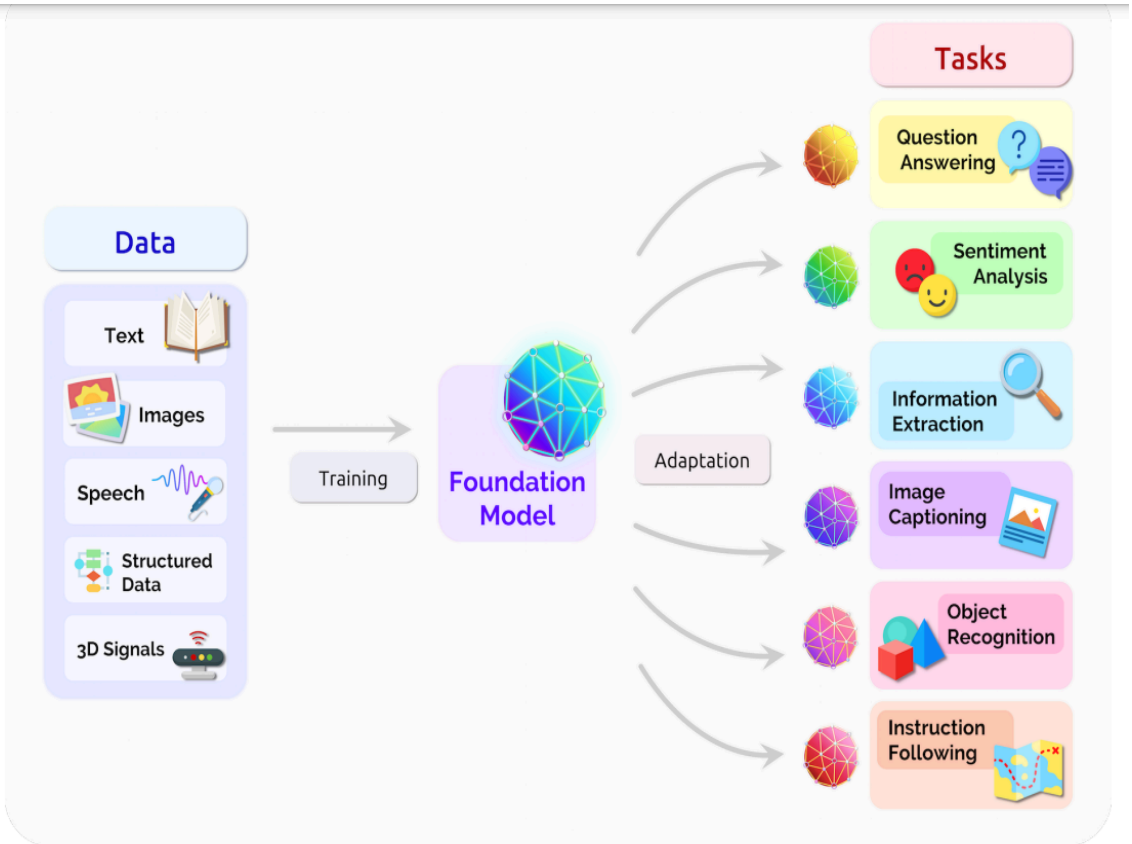


Fig. 2. A foundation model can centralize the information from all the data from various modalities. This one model can then be adapted to a wide range of downstream tasks.

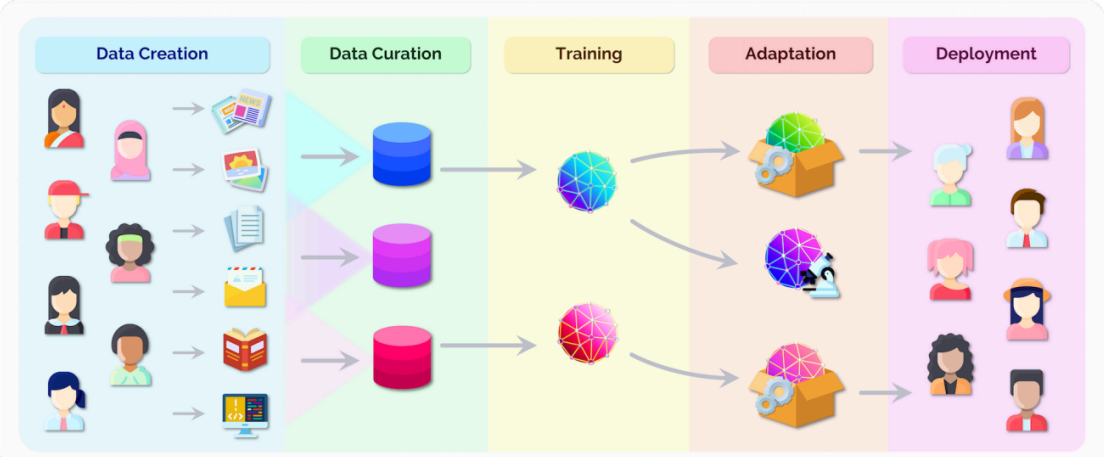


Fig. 3. Before reasoning about the social impact of foundation models, it is important to understand that they are part of a broader ecosystem that stretches from data creation to deployment. At both ends, we highlight the role of people as the ultimate source of data into training of a foundation model, but also as the downstream recipients of any benefits and harms. Thoughtful data curation and adaptation should be part of the responsible development of any AI system. Finally, note that the deployment of adapted foundation models is a decision separate from their construction, which could be for research.

● BERT

BERT (Bidirectional Encoder Representations from Transformers) is a pre-trained deep learning model developed by Google for natural language processing (NLP) tasks. Since its release in 2018, BERT has revolutionized NLP by enabling models to understand language context in a more sophisticated, nuanced way. Here's an overview of what makes BERT unique:

1. Bidirectional Context Understanding

- Traditional NLP models, like RNNs or earlier transformers, often processed text in a single direction (either left-to-right or right-to-left). BERT, however, reads text in both directions simultaneously, which allows it to understand the context of a word by looking at both its preceding and following words.
- This bidirectional approach enables BERT to capture deeper contextual relationships, especially in complex sentences where context may depend on multiple words and phrases.

2. Transformer Architecture

- BERT is built on the Transformer architecture, which is designed to handle long-range dependencies in text more effectively than traditional RNNs. Transformers use self-attention mechanisms to weigh the importance of each word in relation to every other word in a sequence, allowing them to capture intricate relationships.
- In BERT, each word's meaning is represented through a sequence of transformer "layers," where each layer refines the representation based on the context of the surrounding words.

3. Pre-training and Fine-tuning Process

- BERT is pre-trained on a massive amount of text (like Wikipedia and books) using two specific tasks:
    - **Masked Language Modeling (MLM)**: Random words in a sentence are masked, and the model tries to predict them, forcing it to learn contextual representations.
    - **Next Sentence Prediction (NSP)**: BERT learns to predict if two given sentences logically follow each other, helping it learn sentence-level relationships and coherence.
- After pre-training, BERT can be fine-tuned on various specific NLP tasks with relatively small amounts of task-specific data. These tasks might include sentiment analysis, question answering, named entity recognition, or summarization, among others.

● GPT-3 :

Generative Pre-trained Transformer 3, is a large-scale language model created by OpenAI. Released in 2020, it is one of the most advanced language models available, capable of generating highly coherent and contextually relevant text. Here's a breakdown of GPT-3's unique characteristics and capabilities:

1. Transformer Architecture

- Like BERT and many other recent NLP models, GPT-3 is based on the transformer architecture. Transformers are especially good at handling long-range dependencies in text, thanks to their attention mechanism, which allows each word to be weighted based on its relationship to every other word in the input.
- However, unlike BERT, GPT-3 is a unidirectional model, meaning it generates text in a left-to-right manner. It predicts the next word in a sequence based only on the words that came before it, rather than using a bidirectional approach.

2. Massive Scale

- GPT-3 has 175 billion parameters, which are essentially the adjustable parts of the model that enable it to "learn" from data. This massive scale allows GPT-3 to capture a vast amount of language nuances, making it one of the largest NLP models ever developed (at the time of its release).
- For comparison, GPT-2, the previous version, had only 1.5 billion parameters, while models like BERT have around 340 million parameters.

3. Pre-training on Diverse Data

- GPT-3 was trained on a diverse set of internet text, which included web pages, books, and other publicly available data sources. Its training corpus was designed to cover a wide range of knowledge, enabling it to generate text on various topics, including language, science, history, and even creative writing.
- It was pre-trained in a generative manner, which means it learned to generate plausible continuations of text given an initial prompt.

4. Few-shot, Zero-shot, and One-shot Learning

- One of GPT-3's standout features is its ability to perform tasks in few-shot, one-shot, or zero-shot settings:

    - **Zero-shot**: The model can perform a task without any examples. For instance, if asked to summarize a paragraph, GPT-3 can attempt it directly based on the prompt.
    - **One-shot**: GPT-3 can learn from a single example provided in the prompt before performing a task.
    - **Few-shot**: The model is given a few examples of the task in the prompt to understand the format or style needed.
- This flexibility is due to GPT-3's large scale, which allows it to generalize well to various tasks even without specialized training.

5. Text Generation Capabilities

- GPT-3 can generate coherent, contextually relevant, and creative text in a wide variety of formats and styles. It can:
    - Write essays, stories, and poetry
    - Answer questions, explain concepts, and summarize information
    - Generate code snippets and assist with programming tasks
    - Emulate conversational dialogue and play characters
    - Translate languages and correct grammar
- GPT-3's versatility has made it useful in many areas, from content creation and customer support to programming assistance and educational tools.

6. Applications of GPT-3

- **Chatbots and Virtual Assistants**: GPT-3 powers chatbots that provide more human-like interactions and understand complex user intents.
- **Code Assistance**: Codex, a GPT-3-based model, powers GitHub Copilot, which helps developers by suggesting code and completing programming tasks.
- **Content Creation**: GPT-3 is used for creating written content like articles, social media posts, product descriptions, and even marketing copy.
- **Education and Tutoring**: It serves as an interactive tutor for students, answering questions on a wide range of subjects.
- **Research and Summarization**: GPT-3 can assist researchers by summarizing papers or generating hypotheses based on provided information.

7. Limitations and Challenges

- **Bias and Misinformation**: GPT-3 can reflect biases present in its training data and sometimes generate misleading or inaccurate information.
- **Lack of Common Sense Reasoning**: While GPT-3 is powerful, it sometimes struggles with logical consistency or real-world reasoning that humans find intuitive.

- **Dependency on Large-Scale Computation**: The vast scale of GPT-3 requires substantial computational resources, making it costly to deploy at scale

● GRU

A **GRU** (Gated Recurrent Unit) is a type of recurrent neural network (RNN) architecture designed to handle sequential data. GRUs were introduced by Kyunghyun Cho et al. in 2014 as an improvement over traditional RNNs, providing many of the benefits of LSTMs (Long Short-Term Memory networks) with a simplified structure. Here's an overview of GRUs and their significance in NLP and time series applications:

1. Recurrent Neural Network Basics

- Traditional RNNs are designed for sequential tasks, where the input data has a temporal or sequential order. They're commonly used for language modeling, machine translation, and time-series forecasting.
- RNNs process data one step at a time, with each step's output dependent on the previous step's output. However, basic RNNs often suffer from the **vanishing gradient problem**, which limits their ability to retain information over long sequences.

2. GRU Structure and Mechanism

- GRUs address the vanishing gradient issue by using **gates**—mechanisms that control the flow of information to make the model more adaptable to long sequences.
- Unlike LSTMs, which have three gates (input, forget, and output), GRUs have only two gates:
    - **Update Gate**: Controls how much of the previous information is passed to the next step.
    - **Reset Gate**: Decides how much of the past information to forget.
- These gates help the GRU to selectively retain or discard information over long sequences, which makes it effective for tasks that require memory of distant information while being simpler and computationally lighter than LSTMs.

3. How GRUs Work

- The reset gate allows the model to forget irrelevant information from previous steps, making it useful for shorter-term dependencies.
- The update gate, on the other hand, controls how much past information needs to be carried forward, helping the GRU retain important information from earlier steps.
- This setup allows GRUs to learn both short- and long-term dependencies within the data without requiring as many parameters as LSTMs.

4. Advantages of GRUs

- **Simpler Architecture**: With fewer gates and parameters than LSTMs, GRUs are computationally efficient and often faster to train, especially on larger datasets.
- **Effective Memory Retention**: GRUs retain relevant information over long sequences and handle sequential dependencies well, making them suitable for tasks requiring both short- and long-term memory.
- **Reduced Risk of Overfitting**: Fewer parameters in GRUs can lead to reduced overfitting in some cases, especially on smaller datasets, while still providing comparable performance to LSTMs.

5. Applications of GRUs

- **Natural Language Processing (NLP)**: GRUs are commonly used in NLP tasks like machine translation, sentiment analysis, and language modeling due to their ability to handle variable-length text sequences.
- **Time-Series Forecasting**: GRUs are widely used in financial modeling, weather forecasting, and other time-series applications, as they capture temporal dependencies effectively.
- **Speech Recognition**: GRUs are used in speech recognition and synthesis because they handle long sequences of audio data efficiently.

6. Comparison with LSTM

- **Parameter Efficiency**: GRUs have fewer parameters than LSTMs, making them lighter and faster to train.
- **Performance**: GRUs generally perform similarly to LSTMs on many tasks, especially when the dataset isn't overly complex or memory-intensive.
- **Training Speed**: Due to fewer parameters, GRUs can be faster than LSTMs in training, which can be an advantage when working with large-scale datasets or applications that require quick iteration.

● Multimodal Tasks

A **multimodal task** refers to tasks or problems that involve multiple types or modalities of data (e.g., text, images, audio, video, etc.) to provide a more comprehensive understanding or solution. In machine learning and artificial

intelligence, multimodal tasks aim to process and integrate information from these different modalities to improve the accuracy, efficiency, and generalization of models.

Key Concepts in Multimodal Tasks

1. **Modalities**:
    - A **modality** refers to a specific type or form of data. Common modalities in multimodal tasks include:
        - **Text**: Natural language (e.g., documents, tweets, descriptions).
        - **Images**: Visual data (e.g., photos, illustrations).
        - **Audio**: Sound data (e.g., speech, music, environmental sounds).
        - **Video**: Combined visual and audio data.
        - **Sensor Data**: Data from devices like wearables or IoT sensors (e.g., temperature, motion).
2. **Multimodal Integration**:
    - The main challenge in multimodal tasks is to **integrate** information from different modalities in a way that allows the model to use them synergistically. This requires careful design to ensure that relevant features from each modality are appropriately combined and that the model can effectively learn from them.
3. **Types of Multimodal Tasks**:
    - **Multimodal Classification**: Classifying input data that comes from multiple modalities. For example, determining the sentiment of a video based on both its audio (spoken words) and visual (facial expressions) content.
    - **Multimodal Generation**: Generating output based on multiple types of input. For instance, generating a textual description for an image or video (image captioning) or creating a realistic video based on textual descriptions (text-to-video).
    - **Multimodal Retrieval**: Retrieving relevant information from one modality based on a query in another modality. For example, finding images based on a text query or finding videos based on an audio clip.
    - **Multimodal Translation**: Translating from one modality to another, such as converting speech to text, or translating sign language gestures (video) into text or speech.

Examples of Multimodal Tasks

1. **Image Captioning:**

    - The task involves generating a textual description of an image, combining visual information (from the image) with language understanding (to generate the caption).
2. **Visual Question Answering (VQA)**:
    - Given an image and a natural language question about that image, the model must generate a relevant answer. For example, "How many people are in the picture?" requires both image recognition and language comprehension.
3. **Speech-to-Text**:
    - Converting spoken language (audio) into text. This involves understanding the acoustic signals and translating them into readable language.
4. **Multimodal Sentiment Analysis**:
    - Analyzing sentiment from data that involves both visual and textual information. For example, detecting sentiment from a video, which includes both the facial expressions of the person speaking (visual) and the spoken words (text).
5. **Audio-Visual Emotion Recognition**:
    - Recognizing the emotional state of a person in a video based on both their voice (audio) and facial expressions (visual).
6. **Cross-Modal Retrieval**:
    - Searching for images that match a text description or finding a relevant audio clip based on a visual query.

Challenges in Multimodal Tasks

1. **Data Alignment**:
    - One of the biggest challenges is aligning data from different modalities. For example, in video, the alignment between visual and audio data (e.g., ensuring that a spoken word corresponds to the speaker's lip movements) is crucial for effective learning.
2. **Fusion of Modalities**:
    - Deciding how to fuse the information from different modalities is not straightforward. The model can combine features at different stages of processing (early fusion, late fusion, or joint embedding space) depending on the task.
3. **Missing Data**:
    - In real-world scenarios, one modality may be missing or incomplete. For example, a video might have missing audio, or an image may have an unclear background. Handling such missing data is essential for creating robust multimodal models.
4. **Scalability**:

- Working with multiple modalities often requires dealing with large and complex datasets, which can be computationally intensive and require significant storage and processing power.

Techniques for Multimodal Learning

1. **Early Fusion**:
   - In early fusion, features from different modalities are combined at the input level before being fed into the model. For example, you could concatenate the feature vectors from images and text before passing them into a neural network.
2. **Late Fusion**:
   - In late fusion, each modality is processed separately by different models or networks, and the results are combined at the decision level (e.g., by averaging the outputs or voting).
3. **Joint Embedding Space**:
   - In this method, data from different modalities (like text and images) are projected into a shared space where the relationships between modalities can be learned. For example, both text and image data might be embedded into vector representations, and the model learns to associate related pairs of text and images in this shared space.
4. **Attention Mechanisms**:
   - Attention mechanisms, like those used in transformers, can be adapted for multimodal tasks to focus on the most relevant parts of each modality. For instance, in visual question answering, attention could allow the model to focus on specific parts of the image relevant to answering a question.

- RA :

Reinforcement Learning (RL) is a type of machine learning in which an agent learns to make a sequence of decisions by interacting with an environment to achieve a specific goal. Unlike supervised learning, where models learn from

labeled data, RL is a trial-and-error process in which the agent learns through rewards and penalties that it receives based on its actions in the environment.

Key Components of Reinforcement Learning

1. **Agent**:
    - The learner or decision-maker in the RL system. The agent's goal is to find an optimal way to make decisions to maximize the total reward over time.
2. **Environment**:
    - The external system or world in which the agent operates. The environment responds to the agent's actions and provides feedback (rewards or penalties).
3. **State (s)**:
    - A representation of the current situation or condition of the environment. The state provides information about where the agent currently stands in relation to the environment.
4. **Action (a)**:
    - Any move or decision the agent makes in a given state. The set of all possible actions depends on the current state and the specific RL task.
5. **Reward (r)**:
    - A scalar value that the agent receives after taking an action. Rewards guide the agent's learning by indicating whether an action was good or bad. The agent's objective is to maximize the cumulative reward it receives over time.
6. **Policy (π)**:
    - The strategy that the agent follows to decide which action to take in each state. Policies can be deterministic (always taking the same action for a given state) or stochastic (assigning probabilities to actions).
7. **Value Function (V)**:
    - A function that estimates the expected cumulative reward an agent can achieve from a given state, following a particular policy. It helps the agent evaluate how good or bad each state is in terms of potential future rewards.
8. **Q-Value (Q)**:
    - Also called the action-value function, it estimates the expected reward for taking a specific action in a specific state and then following a policy. It's a key concept in Q-learning, an RL algorithm.

How Reinforcement Learning Works

1. **Interaction Loop**:
    - The agent interacts with the environment in a loop. For each time step:
        - The agent observes the current state.
        - Based on its policy, the agent takes an action.
        - The environment responds with a reward and provides the next state.
        - The agent updates its knowledge (policy, value functions) based on the reward and the new state.
2. **Exploration vs. Exploitation**:
    - The agent needs to balance between:
        - **Exploitation**: Using its current knowledge to maximize rewards by choosing the best-known action.
        - **Exploration**: Trying new actions to gather more information about the environment, which may lead to better long-term rewards.
    - The **exploration-exploitation trade-off** is a central challenge in RL.
3. **Learning Goal**:
    - The agent's objective is to learn a policy that maximizes the **cumulative reward** over time. This cumulative reward is often called the **return**, which can either be finite (for a fixed number of steps) or infinite (for continuing tasks).

Types of Reinforcement Learning

1. **Model-Free vs. Model-Based RL**:
    - **Model-Free RL**: The agent learns the optimal policy based on rewards received without knowing the transition dynamics of the environment. Common algorithms include Q-learning and SARSA.
    - **Model-Based RL**: The agent attempts to model the environment's dynamics (e.g., transition probabilities between states) and uses this model to plan its actions.
2. **On-Policy vs. Off-Policy RL**:
    - **On-Policy**: The agent learns the value of the policy it is currently following. For example, SARSA (State-Action-Reward-State-Action) is an on-policy algorithm.
    - **Off-Policy**: The agent learns the value of the optimal policy independently of the agent's actions. Q-learning is an off-policy algorithm.
3. **Value-Based, Policy-Based, and Actor-Critic Methods**:
    - **Value-Based**: The agent learns value functions (like Q-values) to decide actions. Q-learning is a popular value-based method.

- **Policy-Based**: The agent directly learns the policy (a mapping from states to actions). Policy Gradient methods are examples.
- **Actor-Critic**: Combines both value-based and policy-based approaches. The “actor” updates the policy, and the “critic” estimates the value function to help guide the actor.

Popular Reinforcement Learning Algorithms

1. **Q-Learning**:
   - A value-based, model-free RL algorithm that learns the Q-values for each action-state pair. Q-learning aims to learn the optimal action-value function Q(s,a)Q(s, a)Q(s,a) by updating the Q-values based on received rewards.
2. **SARSA**:
   - An on-policy version of Q-learning, where the agent updates its Q-values based on the action it actually takes, rather than the optimal action.
3. **Deep Q-Network (DQN)**:
   - An extension of Q-learning that uses deep neural networks to approximate Q-values, enabling it to work with high-dimensional inputs like images.
4. **Policy Gradient Methods**:
   - A class of algorithms where the policy itself is optimized directly, usually through gradient ascent on expected rewards. Common examples include REINFORCE and PPO (Proximal Policy Optimization).
5. **Actor-Critic Methods**:
   - Combines value-based and policy-based methods. The actor decides actions, and the critic evaluates them. This setup helps improve training stability. Examples include A3C (Asynchronous Advantage Actor-Critic) and DDPG (Deep Deterministic Policy Gradient).

• Universal Models :

A **universal model** in artificial intelligence (AI) is a highly generalized model designed to handle a wide range of tasks across different domains without requiring significant task-specific customization. Unlike conventional models that are trained for a specific function (like language translation or image classification), universal models are built to work across diverse applications,

often achieving generalization and transferability by leveraging large-scale pre-training and multimodal capabilities.

Characteristics of Universal Models

1. **General-Purpose Architecture**:
    - Universal models are typically built on architectures that can process various data types and tasks, like transformers. This flexibility enables the model to handle tasks across different domains (e.g., text, vision, audio) and adapt to new tasks with minimal changes.
2. **Multimodal Capability**:
    - Universal models are often designed to handle multiple data modalities (text, image, audio, video) simultaneously or in combination. By integrating multimodal learning, they can perform tasks like image captioning, text-to-image generation, and more.
3. **Extensive Pre-training**:
    - Universal models are typically pre-trained on vast, diverse datasets across different domains and modalities. This broad exposure allows the model to learn foundational knowledge, which can then be applied to various downstream tasks without requiring extensive task-specific data or training.
4. **Transfer Learning and Adaptability**:
    - These models can transfer learned knowledge from one domain or task to another, showing high adaptability. The pre-training stage enables the universal model to serve as a foundation for fine-tuning on specific tasks with minimal data, similar to the concept of foundation models but with an emphasis on broader, universal applications.
5. **Zero-Shot and Few-Shot Learning**:
    - Universal models often excel in zero-shot and few-shot learning, where they perform new tasks with little or no additional task-specific data. This capability stems from their general training and vast exposure to varied information.
6. **Scalability and Efficiency**:
    - Scalability is crucial for universal models, as they need to be able to accommodate extensive datasets and complex architectures. Efficiency in both computation and memory is also critical, especially as these models grow larger.

Examples of Universal Models

1. **GPT-4**:

    - The latest generation of OpenAI's language model, GPT-4, is designed with capabilities across multiple domains (e.g., text, image understanding) and performs well on diverse language tasks. Although initially trained on text data, GPT-4 has expanded its utility across tasks like generating code, summarizing documents, answering questions, and interpreting images.
2. **Gato (DeepMind)**:
    - Gato by DeepMind is an experimental universal model that is trained to perform a variety of tasks, from controlling robotic arms to playing video games to performing language-based tasks. By using a single neural network that can handle different input types and tasks, Gato is a step toward developing a general-purpose AI model.
3. **CLIP (Contrastive Language-Image Pretraining)**:
    - CLIP, also by OpenAI, is a universal model that learns from both text and image pairs. It can be used in zero-shot learning to recognize objects in images based on textual descriptions, allowing it to perform image recognition without explicit task-specific training.
4. **FLAN (Google)**:
    - FLAN is a series of instruction-fine tuned models by Google that are pre-trained on a mixture of different tasks. It is intended to generalize across various NLP tasks and can perform reasonably well without task-specific adjustments.
5. **DALL·E**:
    - Another universal model by OpenAI, DALL·E generates images based on textual descriptions. While primarily a text-to-image generator, it combines multimodal understanding (text and visual data) in a way that has broad applications, from creative content generation to aiding accessibility in image descriptions.

Q : how do we have a usable meaning of a word in a computer ?

In traditional Natural Language Processing (NLP) systems, meaning has often been handled by leveraging lexical resources, with **WordNet** being one of the most prominent. WordNet is a large lexical database that organizes English words into sets of synonyms called "synsets" and records their semantic

relationships, such as hypernyms (generalization relationships), hyponyms (specificity relationships), antonyms, and more.

Traditional Approaches to Meaning in NLP

For many years, the primary approach to understanding and representing meaning in NLP relied on structured linguistic resources such as:

1. **WordNet**:
    - WordNet organizes words into synsets, where each synset represents a distinct concept or meaning. For example, the word "bank" has several synsets, one related to a financial institution and another to a riverbank. By providing such structured meanings, WordNet helps systems disambiguate word senses and understand semantic similarity.
    - WordNet also includes relationships like hypernymy (e.g., "dog" is a type of "animal"), meronymy (e.g., "wheel" is a part of "car"), and antonymy, which helps NLP systems understand how words relate conceptually.
2. **Dictionaries and Thesauri**:
    - Dictionaries provide definitions and part-of-speech tags, giving basic information about each word's meaning, while thesauri group similar words. While not as rich as WordNet, they give NLP systems a foundation to understand synonyms and basic word relationships.
3. **FrameNet and ConceptNet**:
    - **FrameNet** is a resource based on frame semantics, which groups words by conceptual frames or scenarios (e.g., "buying" has roles like buyer, seller, goods, and money). ConceptNet is a semantic network that connects words and phrases across multiple languages with relationships like "is a part of" and "used for."
    - These resources contribute additional context and help NLP systems interpret words within larger structures and scenarios, useful in tasks like sentiment analysis and question answering.
4. **Rule-Based Systems**:
    - Early NLP systems also relied on manually crafted rules to parse sentences and determine relationships between words. These rules might include specific patterns for identifying verbs, subjects, and objects, as well as patterns for common phrases. However, rule-based systems struggled to scale due to the ambiguity and variability of natural language.

Limitations of Traditional Approaches : Localist Representations

While these lexical resources provided a structured way to interpret meaning, they also came with significant limitations:

1. **Lack of Contextual Understanding**:
    - Traditional resources like WordNet treat word meanings as fixed and isolated. They lack contextual awareness, meaning they cannot adapt their interpretations based on the sentence or broader context. For instance, "bank" as a financial institution versus "bank" as a riverbank could only be distinguished by strict rules rather than nuanced context.
2. **Static Nature**:
    - WordNet and similar resources are static and cannot dynamically adapt to new word senses or emerging slang. This creates a bottleneck in modern NLP applications where language evolves rapidly.
3. **Limited Coverage for Specialized Domains**:
    - Lexical resources often lack coverage in specialized fields like medical or legal terminology. They also lack the flexibility to cover less common languages and dialects, further restricting their use across diverse applications.
4. **Manual Construction and Maintenance**:
    - Resources like WordNet were manually curated, which made them labor-intensive to create and maintain. Updating them with new meanings or relationships was time-consuming and costly.

Modern Approaches to Handling Meaning in NLP : Distributional Semantics : The word's meaning is given by the words that frequently appear close by

-When a word w appears in a sequence ,its context is the set of words that appear nearby
-Word vector=word embedding =word representation =>Distributed Representation

With advancements in deep learning and natural language models, the focus has shifted towards data-driven and context-aware methods to handle meaning, addressing many of the limitations of traditional resources like WordNet:

1. **Word Embeddings**:
    - **Word2Vec**, **GloVe**, and **fastText** introduced a new way to represent words as dense vectors in continuous vector spaces. Words that appear in similar contexts tend to have similar embeddings, capturing nuanced meanings and semantic similarities based on real usage rather than rigid definitions.

    - Word embeddings have the advantage of being trained on large corpora, meaning they automatically capture relationships and common patterns across large amounts of data.
2. **Contextualized Word Embeddings**:
    - Models like **ELMo**, **BERT**, and **GPT** introduced contextual embeddings, which generate different representations for a word depending on the context in which it appears. This allows these models to disambiguate words like "bank" dynamically, distinguishing their meanings based on surrounding text.
    - Contextual embeddings allow NLP models to perform well on tasks that require nuanced meaning understanding, such as sentiment analysis, named entity recognition, and question answering.
3. **Pre-trained Language Models as Knowledge Sources**:
    - Large pre-trained language models (e.g., **BERT**, **RoBERTa**, **GPT-3**) have become “foundation models” that capture vast amounts of world knowledge from diverse text sources, making them adaptable to various tasks. These models can often "understand" meaning, infer relationships, and answer questions without needing explicit ontologies or lexical resources.
    - These models have learned relationships, factual knowledge, and even certain reasoning patterns, allowing them to generalize across tasks like summarization, translation, and entity recognition with high accuracy.
4. **Transfer Learning and Fine-Tuning**:
    - Fine-tuning these large models on specific datasets or tasks allows them to adapt to particular domains or purposes. For example, fine-tuning a BERT model on medical data can make it more accurate in medical NLP tasks.
    - This method has proven far more efficient and scalable than manually updating lexical resources like WordNet.
5. **Multimodal and Multitask Learning**:
    - Newer models like **CLIP** and **DALL·E** incorporate multimodal learning, integrating both text and image data, while multitask models like **FLAN** handle numerous NLP tasks simultaneously. These advances are pushing the boundaries of meaning representation by connecting linguistic meaning with visual, auditory, and even sensory understanding.
6. **Neural Knowledge Bases and Retrieval-Augmented Models**:
    - Models like **REALM** (Retrieval-Augmented Language Model) and **RAG** (Retrieval-Augmented Generation) combine deep language models with external knowledge bases, allowing them to query up-to-date information dynamically, which is especially useful for

facts that change over time or new information that was not in the training corpus.

- **Natural Language Toolkit** :

NLTK is a leading platform for building Python programs to work with human language data. It provides easy-to-use interfaces to over 50 corpora and lexical resources such as WordNet, along with a suite of text processing libraries for classification, tokenization, stemming, tagging, parsing, and semantic reasoning, wrappers for industrial-strength NLP libraries, and an active discussion forum.

Thanks to a hands-on guide introducing programming fundamentals alongside topics in computational linguistics, plus comprehensive API documentation, NLTK is suitable for linguists, engineers, students, educators, researchers, and industry users alike. NLTK is available for Windows, Mac OS X, and Linux. Best of all, NLTK is a free, open source, community-driven project.

NLTK has been called “a wonderful tool for teaching, and working in, computational linguistics using Python,” and “an amazing library to play with natural language.”

- Example of code :

```
   # Import the Natural Language Toolkit (NLTK) library
import nltk

# Download the necessary resources
nltk.download('punkt')              # Tokenizer model for word
tokenization
nltk.download('vader_lexicon')      # Sentiment analysis lexicon
nltk.download('stopwords')          # Stopwords list
nltk.download('wordnet')            # WordNet lexical database

# Import necessary modules from NLTK
from nltk.tokenize import word_tokenize            # For
tokenizing text into words
from nltk.stem import PorterStemmer, WordNetLemmatizer  # For
stemming and lemmatizing words
from nltk.probability import FreqDist              # For
calculating word frequency distribution
```

```
from nltk.corpus import stopwords                    # For accessing
the list of English stopwords
from nltk.corpus import wordnet as wn                # For accessing
WordNet for synonyms, antonyms, etc.
from nltk.sentiment import SentimentIntensityAnalyzer  # For
sentiment analysis

# Sample text for processing
text = """I did not love NLP #playing"""

# Tokenize the text into words
tokens = word_tokenize(text)
print("Tokenized text:", tokens)  # Output the list of tokens

# Initialize the PorterStemmer and WordNetLemmatizer
stemmer = PorterStemmer()         # Stemming removes suffixes to
reduce words to their root form
lemmatizer = WordNetLemmatizer()   # Lemmatization reduces words
to their base form (lemma)

# Stem each word in the tokenized text
stemmed_words = [stemmer.stem(word) for word in tokens]
print("Stemmed words:", stemmed_words)  # Output the stemmed words

# Lemmatize each word in the tokenized text, treating each word as
a verb ('v')
lemmatized = [lemmatizer.lemmatize(word, "v") for word in tokens]
print("Lemmatized words:", lemmatized)  # Output the lemmatized
words

# Calculate the frequency distribution of tokens
fd = FreqDist(tokens)
print("Frequency Distribution:", fd)     # Output the frequency
distribution object
print("Most common 3 words:", fd.most_common(3))  # Display the
top 3 most common words

# Uncomment the following lines to plot the Frequency Distribution
if matplotlib is installed
# import matplotlib.pyplot as plt         # Import pyplot for
plotting
# fd.plot(30, cumulative=False)           # Plot the frequency
```

```
distribution for the top 30 words
# plt.show()                                # Display the plot

# Get the list of English stopwords
stop_words = stopwords.words('english')

# Remove stopwords from tokens
newtext = [word for word in tokens if word.lower() not in
stop_words]
print("Filtered text without stopwords:", newtext)  # Output the
filtered list

# Sentiment analysis using VADER
sentim = SentimentIntensityAnalyzer()
sentiment_scores = sentim.polarity_scores(text)
print("Sentiment scores:", sentiment_scores)

# Get synonyms of the word "NLP" from WordNet and print its
definition
syn = wn.synsets("NLP")  # Look up synonyms for "NLP"

for synset in syn:
    print(f"{synset.name()}: {synset.definition()}")
print("\n")

# Step 2: Extract Synonyms and Antonyms
synonyms = set()
antonyms = set()

for synset in syn:
    # Add synonyms from each synset
    for lemma in synset.lemmas():
        synonyms.add(lemma.name())
        # Check for antonyms
        if lemma.antonyms():
            antonyms.add(lemma.antonyms()[0].name())

print("Synonyms of 'NLP':", synonyms)
print("Antonyms of 'NLP':", antonyms)
print("\n")

# Step 3: Find Hypernyms and Hyponyms (semantic relations)
```

```
first_synset = syn[0] if syn else None

if first_synset:
    # Hypernyms (more general concepts)
    hypernyms = first_synset.hypernyms()
    print("Hypernyms of 'NLP':")
    for hypernym in hypernyms:
        print(f"- {hypernym.name()}: {hypernym.definition()}")

    # Hyponyms (more specific concepts)
    hypernyms = first_synset.hyponyms()
    print("\nHyponyms of 'NLP':")
    for hyponym in hyponyms:
        print(f"- {hyponym.name()}: {hyponym.definition()}")
else:
    print("No synsets found for 'NLP'.")
```

-Outputs :

Tokenized text: ['I', 'did', 'not', 'love', 'NLP', '#', 'playing']

Stemmed words: ['i', 'did', 'not', 'love', 'nlp', '#', 'play']

Lemmatized words: ['I', 'do', 'not', 'love', 'NLP', '#', 'play']

Frequency Distribution: <FreqDist with 7 samples and 7 outcomes>

Most common 3 words: [('I', 1), ('did', 1), ('not', 1)]

Filtered text without stopwords: ['love', 'NLP', '#', 'playing']

Sentiment scores: {'neg': 0.457, 'neu': 0.543, 'pos': 0.0, 'compound': -0.5216}

natural_language_processing.n.01: the branch of information science that deals with natural language information

Synonyms of 'NLP': {'natural_language_processing', 'human_language_technology', 'NLP'}

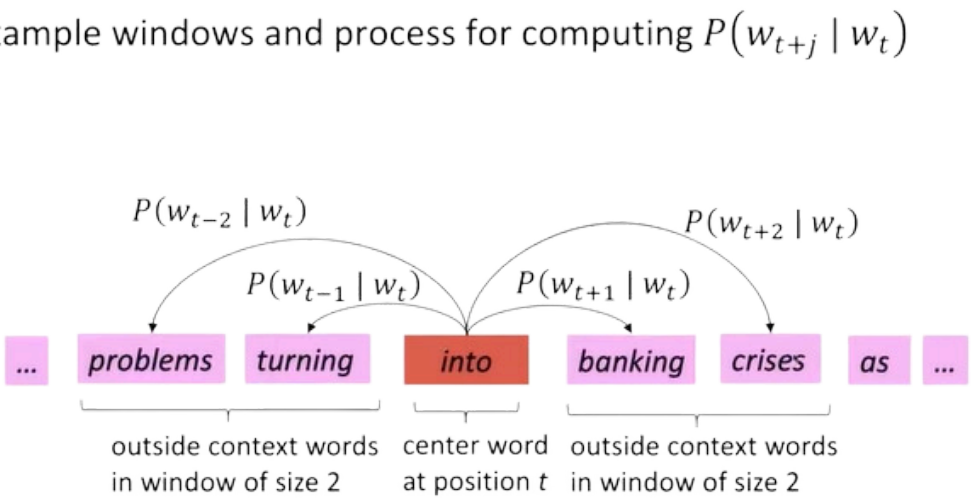
Antonyms of 'NLP': set()

Hypernyms of 'NLP':

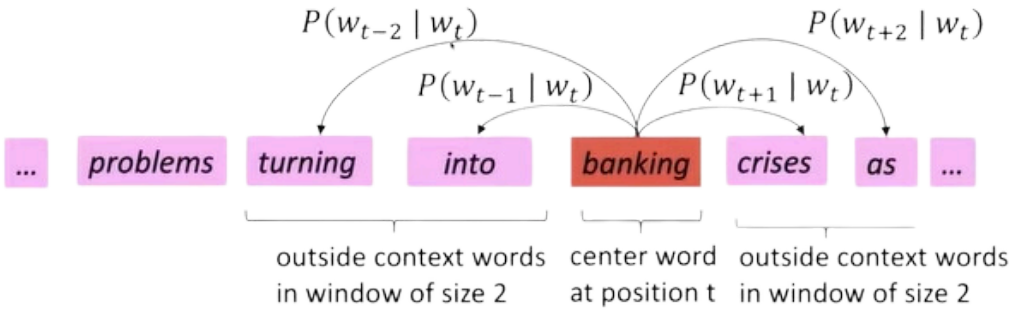
- information_science.n.01: the sciences concerned with gathering, manipulating, storing, retrieving, and classifying recorded information

- **Word2Vec :**

- **Word2Vec** is a neural network-based framework for learning **word embeddings**, which are dense vector representations of words in a fixed vocabulary. Each word in this vocabulary is represented as a continuous, low-dimensional vector, with the goal of capturing the semantic relationships and similarities between words based on their contexts in large text corpora. In a Word2Vec model, words that frequently appear in similar contexts are mapped closer together in the vector space, which helps the model understand relationships like synonyms, analogies, and other linguistic patterns.



Example windows and process for computing $P(w_{t+j} \mid w_t)$



For each position $t = 1, \ldots, T$, predict context words within a window of fixed size m, given center word w_j. Data likelihood:

$$\text{Likelihood} = L(\theta) = \prod_{t=1}^{T} \prod_{\substack{-m \le j \le m \\ j \neq 0}} P(w_{t+j} \mid w_t; \theta)$$

θ is all variables to be optimized

sometimes called a *cost* or *loss* function

The objective function $J(\theta)$ is the (average) negative log likelihood:

$$J(\theta) = -\frac{1}{T} \log L(\theta) = -\frac{1}{T} \sum_{t=1}^{T} \sum_{\substack{-m \le j \le m \\ j \neq 0}} \log P(w_{t+j} \mid w_t; \theta)$$

Minimizing objective function ⟺ Maximizing predictive accuracy

- The goal of calculating the probability P(wt+1 | wt) , or similar probabilities in word embedding models like Word2Vec, is to **learn the relationships** and **semantic similarities** between words based on their co-occurrence patterns in text.
- "Learn word embeddings" refers to the process of training a machine learning model to represent words as dense vectors in a continuous vector space, such that words with similar meanings or words that frequently appear in similar contexts are represented by similar vectors. This allows the model to capture semantic and syntactic relationships between words.
- Word embeddings are learned based on the **context** in which words appear. The context is usually defined as the surrounding words within a window size

- Word2Word typically refers to a concept or model where words are mapped to vector representations (embeddings) in a continuous vector space, enabling the model to learn and understand the relationships between words based on their co-occurrence patterns in a large corpus of text. While Word2Word is not a standard term in NLP, it often refers to models like Word2Vec, which learn these word embeddings.

1. Goal: The goal is to learn word embeddings—dense vector representations of words—such that similar words (in meaning or context) have similar vector representations.
2. Skip-gram Model: Given a target word, the model predicts the surrounding context words.
3. CBOW (Continuous Bag of Words) Model: Given a set of context words, the model predicts the target word.
4. Learning Process: The model adjusts the embeddings during training to maximize the likelihood of predicting the correct context or target word.
5. Output : The model outputs word embeddings that capture semantic and syntactic relationships, allowing for applications like word similarity, analogy solving, and various NLP tasks.
6. Training : This is done on a large text corpus using optimization techniques like stochastic gradient descent to minimize the prediction error and learn high-quality embeddings.

Word2Vec's word embeddings have wide applications in NLP, such as improving text classification, machine translation, and sentiment analysis.

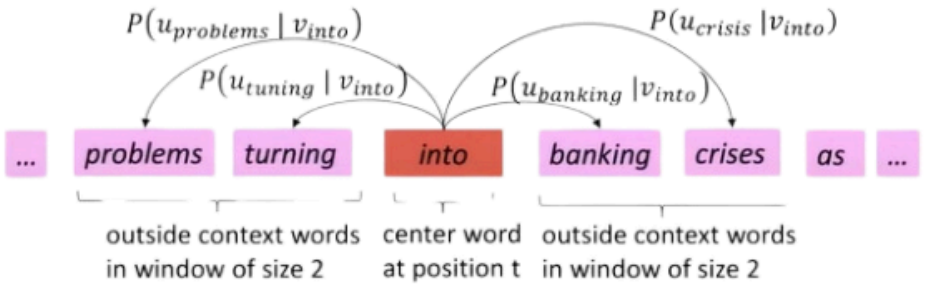
- We want to minimize the objective function:

$$J(\theta) = -\frac{1}{T}\sum_{t=1}^{T}\sum_{\substack{-m\le j\le m \\ j\neq 0}} \log P\left(w_{t+j} \mid w_t; \theta\right)$$

- **Question:** How to calculate $P\left(w_{t+j} \mid w_t; \theta\right)$?
- **Answer:** We will *use two* vectors per word *w*:
  - v_w when *w* is a center word
  - u_w when *w* is a context word
- Then for a center word *c* and a context word *o*:

$$P(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w\in V}\exp(u_w^T v_c)}$$

- Example windows and process for computing $P\left(w_{t+j} \mid w_t\right)$
- $P\left(u_{problems} \mid v_{into}\right)$ short for $\mathrm{P}\left(problems \mid into\ ; u_{problems}, v_{into}, \theta\right)$



② Exponentiation makes anything positive

$$P(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w \in V} \exp(u_w^T v_c)}$$

① Dot product compares similarity of o and c.
$u^T v = u.v = \sum_{i=1}^{n} u_i v_i$
Larger dot product = larger probability

③ Normalize over entire vocabulary to give probability distribution

- This is an example of the **softmax function** $\mathbb{R}^n \to (0,1)^n$ ← Open region

$$\text{softmax}(x_i) = \frac{\exp(x_i)}{\sum_{j=1}^{n} \exp(x_j)} = p_i$$

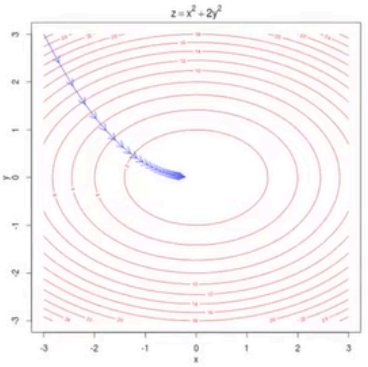
- The softmax function maps arbitrary values x_i to a probability distribution p_i
  - "max" because amplifies probability of largest x_i
  - "soft" because still assigns some probability to smaller x_i
  - Frequently used in Deep Learning

But sort of a weird name because it returns a distribution!

To train a model, we gradually adjust parameters to minimize a loss

- Recall: θ represents **all** the model parameters, in one long vector
- In our case, with d-dimensional vectors and V-many words, we have:
- Remember: every word has two vectors

$$\theta = \begin{bmatrix} v_{aardvark} \\ v_a \\ \vdots \\ v_{zebra} \\ u_{aardvark} \\ u_a \\ \vdots \\ u_{zebra} \end{bmatrix} \in \mathbb{R}^{2dV}$$



- We optimize these parameters by walking down the gradient (see right figure)
- We compute all vector gradients!

-word2vec is not a singular algorithm, rather, it is a family of model architectures and optimizations that can be used to learn word embeddings from large datasets. Embeddings learned through word2vec have proven to be successful on a variety of downstream natural language processing tasks.

- Notebook

The **Continuous Bag-of-Words (CBOW)** and **Continuous Skip-Gram** models are two popular architectures used in natural language processing, especially for generating word embeddings in neural networks. While they both aim to capture word meanings based on their context, they approach the task from opposite perspectives. Here's a breakdown of their key differences:

1. Objective of Prediction

- **CBOW Model**: The goal is to predict a target (middle) word given the surrounding context words. For instance, if we have a sentence, "The cat sits on the mat," and want to predict "sits," we use "The," "cat," "on," "the," and "mat" as context words.
- **Skip-Gram Model**: The objective here is to predict the context words surrounding a given target word. Using the same example sentence, if "sits" is the target word, the model tries to predict "The," "cat," "on," "the," and "mat."

2. Training Focus

- **CBOW**: Trained to maximize the likelihood of predicting the target word based on a "bag" of context words, ignoring the order of the surrounding words. This approach is computationally efficient and suitable when working with large datasets.
- **Skip-Gram**: Trained to maximize the likelihood of predicting context words within a specified range around a target word. Since the model predicts multiple words for each target, this approach can learn more nuanced relationships, especially for rarer words, but is typically slower than CBOW.

3. Handling Word Order in Context

- **CBOW**: Treats the context words as a "bag of words," meaning the order of the context words around the target word doesn't matter. This makes the CBOW model simpler but may lose some information about word sequence.
- **Skip-Gram**: Maintains directional context by predicting words within a specified range before and after the target word, preserving a sense of word order around the target.

4. Use Cases and Performance

- **CBOW**: Generally better for larger corpora and is computationally efficient, as it aggregates the context words into a single representation. This model often works well for tasks involving high-frequency words.
- **Skip-Gram**: Tends to work better with smaller datasets and can capture better representations of infrequent or rare words, as it explicitly models the directional relationship between the target and context words.

• Efficient Estimation of Word Representations in Vector Space :

- Syntactic and semantic word similarities :

Syntactic and semantic word similarities are both measures of how closely related two words are, but they approach similarity from different perspectives:

1. Syntactic Similarity

- Syntactic similarity is based on the structure and form of words.
- It considers how words appear or function in sentences, such as their morphological similarity (like prefixes or suffixes) and part-of-speech similarity (e.g., nouns with nouns, verbs with verbs).
- Words like *running* and *runner* have high syntactic similarity because they share the root *run* and are morphological variants of the same base word. However, their meanings differ: *running* describes an action, while *runner* refers to a person.
- Syntactic similarity is useful for **text normalization** (e.g., stemming and lemmatization), **part-of-speech tagging**, and **spell-checking**, where understanding the structure of words is more important than their meanings.

2. Semantic Similarity

- Semantic similarity is based on the **meaning** or **conceptual relation** between words.
- It measures the closeness of the meanings of two words, regardless of their structure or form, often using techniques like **word embeddings** (e.g., Word2Vec or GloVe) or **ontologies** (e.g., WordNet).
- Words like *car* and *automobile* have high semantic similarity because they refer to the same concept, even though they look and sound different. Words like *cold* and *chilly* also have high semantic similarity because they describe similar sensations.
- Semantic similarity is widely used in **information retrieval**, **natural language understanding**, **sentiment analysis**, and **machine translation**, where understanding the actual meaning of words or phrases is crucial.

-Many current NLP systems treat words as atomic units, without accounting for similarity between words. Words are often represented as indices in a vocabulary, a choice driven by simplicity, robustness, and the effectiveness of simple models trained on vast amounts of data. For example, N-gram models are popular in statistical language modeling, and today, it's feasible to train N-grams on virtually all available data. However, these simple techniques have limitations for many tasks. In automatic speech recognition, for instance, performance is largely constrained by the quantity of high-quality, transcribed speech data (often limited to millions of words).

Similarly, in machine translation, existing corpora for many languages contain only a few billion words or fewer. In such cases, merely scaling up basic techniques yields limited improvements, necessitating more advanced approaches.

To address these challenges, various techniques have been proposed to improve the quality of vector representations. The goal is to ensure that similar words are positioned close to each other and that different degrees of similarity can be represented. Interestingly, it was discovered that word vector similarities go beyond simple syntactic regularities. For example, using the word offset technique—where simple algebraic operations are performed on word vectors—it was found that vector(“King”) - vector(“Man”) + vector(“Woman”) results in a vector closest to the vector representation of *Queen*.

-A feedforward neural network with a linear projection layer and a non-linear hidden layer has been used to jointly learn word vector representations and a statistical language model. One notable architecture first learns word vectors using a neural network with a single hidden layer, and these word vectors are then used to train the neural network language model (NNLM), making it possible to learn word vectors without constructing the full NNLM.

-These word vectors have been shown to significantly enhance and simplify numerous NLP applications. Word vector estimation itself has been performed using various model architectures and trained on diverse corpora, with some of the resulting word vectors made publicly available for research and comparison. However, as far as we know, training these architectures remains computationally expensive, with the exception of certain versions of the log-bilinear model, which use diagonal weight matrices to reduce computational demands.

-Many different types of models were proposed for estimating continuous representations of words, including the well-known Latent Semantic Analysis (LSA) and Latent Dirichlet Allocation (LDA). Distributed representations of words learned by neural networks perform significantly better than LSA for preserving linear regularities among words . Moreover LSA becomes computationally very expensive on large data sets.

- To compare different model architectures first the computational complexity of a model as the number of parameters that need to be accessed to fully train the model was defined . Next, we will try to maximize the accuracy, while minimizing the computational complexity.
- The training complexity is proportional to $O = E \times T \times Q$

where E is the number of the training epochs, T is the number of the words in the training set and Q is defined further for each model architecture. Common choice is E = 3 to 50 and T up to one billion. All models are trained using stochastic gradient descent and backpropagation

- Model Architectures :

● Feedforward Neural Net Language Model (NNLM) :

-The goal of NNLM is to predict the next word in a sentence .It consists of input, projection, hidden and output layers. At the input layer, N previous words are encoded using 1-of-V coding, where V is the size of the vocabulary.

-The input layer is then projected to a projection layer P that has dimensionality N × D, using a shared projection matrix. As only N inputs are active at any given time, composition of the projection layer is a relatively cheap operation. The NNLM architecture becomes complex for computation between the projection and the hidden layer, as values in the projection layer are dense. For a common choice of N = 10, the size of the projection layer (P) might be 500 to 2000, while the hidden layer size H is typically 500 to 1000 units. Moreover, the hidden layer is used to compute probability distribution over all the words in the vocabulary, resulting in an output layer with dimensionality V .
Thus, the computational complexity per each training example is

$$Q = N \times D + N \times D \times H + H \times V$$

where the dominating term is H × V . However, several practical solutions were proposed for avoiding it; either using hierarchical versions of the softmax or avoiding normalized models completely by using models that are not normalized during training . With binary tree representations of the vocabulary, the number of output units that need to be evaluated can go down to around log2(V). Thus, most of the complexity is caused by the term N × D × H.

- When the vocabulary size is one million words, this results in about two times speedup in evaluation. While this is not crucial speedup for neural network LMs as the computational bottleneck is in the N ×D×H term. Architectures that do not have hidden layers were proposed and thus depend heavily on the efficiency of the softmax normalization

● Recurrent Neural Net Language Model (RNNLM)

-Recurrent neural network based language model has been proposed to overcome certain limitations of the feedforward NNLM, such as the need to specify the context length (the order of the model N), and because theoretically RNNs can efficiently represent more complex patterns than the shallow neural networks . The RNN model does not have a projection layer; only input, hidden and output layers. What is special for this type of model is the recurrent matrix that connects the hidden layer to itself, using time-delayed connections. This allows the recurrent model to form some kind of short term memory, as information from the past can be represented by the hidden layer

state that gets updated based on the current input and the state of the hidden layer in the previous time step.
-The complexity per training example of the RNN model is

$$Q = H \times H + H \times V$$

where the word representations D have the same dimensionality as the hidden layer H. Again, the term H × V can be efficiently reduced to H × log2(V) by using hierarchical softmax. Most of the complexity then comes from H × H

-To train models on huge data sets, several models were implemented on top of a large-scale distributed framework called DistBelief , including the feedforward NNLM and the new models

- DistBelief

DistBelief is a large-scale distributed framework developed by Google to train deep neural networks efficiently across multiple machines. By leveraging distributed computing, DistBelief allows for the parallel processing of large datasets, enabling the training of complex models that would otherwise be computationally prohibitive on a single machine. This framework has been foundational in advancing neural network scalability, facilitating the development of deep learning models for various applications, including image recognition, natural language processing, and more.

DistBelief also paved the way for subsequent frameworks like TensorFlow, which integrated similar distributed computing principles with broader accessibility and flexibility for researchers and developers.

● New Log-linear Models :

-The main observation from the previous section was that most of the complexity is caused by the non-linear hidden layer in the model. While this is what makes neural networks so attractive, we decided to explore simpler models that might not be able to represent the data as precisely as neural networks, but can possibly be trained on much more data efficiently

● Continuous Bag-of-Words Model :

-The proposed architecture is similar to the feedforward NNLM, where the nonlinear hidden layer is removed and the projection layer is shared for all words (not just the projection matrix) thus, all words get projected into the same position (their vectors are averaged). We call this architecture a bag-of-words model as the order of words in the history does not influence the projection. Furthermore, we also use words from the future; we have obtained the best performance on the task introduced in the next section by building a log-linear classifier with four future and four history words at the input, where the training criterion is to correctly classify the current (middle) word.

Training complexity is then $Q = N \times D + D \times \log2(V)$

-This model further is denoted as CBOW, as unlike the standard bag-of-words model, it uses continuous distributed representation of the context. The model architecture is shown at Figure 1. Note that the weight matrix between the input and the projection layer is shared for all word positions in the same way as in the NNLM. Furthermore, The CBOW architecture predicts the current word based on the context

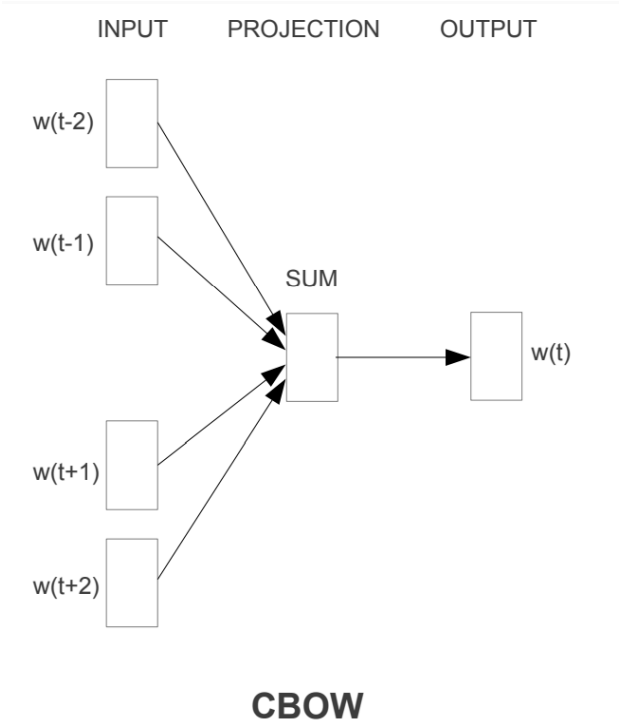


Figure 1 : The CBOW architecture

● Continuous Skip-gram Model

- The second architecture is similar to CBOW, but instead of predicting the current word based on the context, it tries to maximize classification of a word based on another word in the same sentence. More precisely, we use each current word as an input to a log-linear classifier with continuous projection layer, and predict words within a certain range before and after the current word. We found that increasing the range improves the quality of the resulting word vectors, but it also increases the computational complexity. Since the more distant words are usually less related to the current word than those close to it, we give less weight to the distant words by sampling less from those words in our training examples. The training complexity of this architecture is proportional to $Q = C \times (D + D \times \log2(V))$

where C is the maximum distance of the words. Thus, if we choose C = 5, for each training word we will select randomly a number R in range < 1; C >, and

then use R words from history and R words from the future of the current word as correct labels. This will require us to do R × 2 word classifications, with the current word as input, and each of the R + R words as output. In the following experiments, we use C = 10

- The Skip-gram predicts surrounding words given the current word.

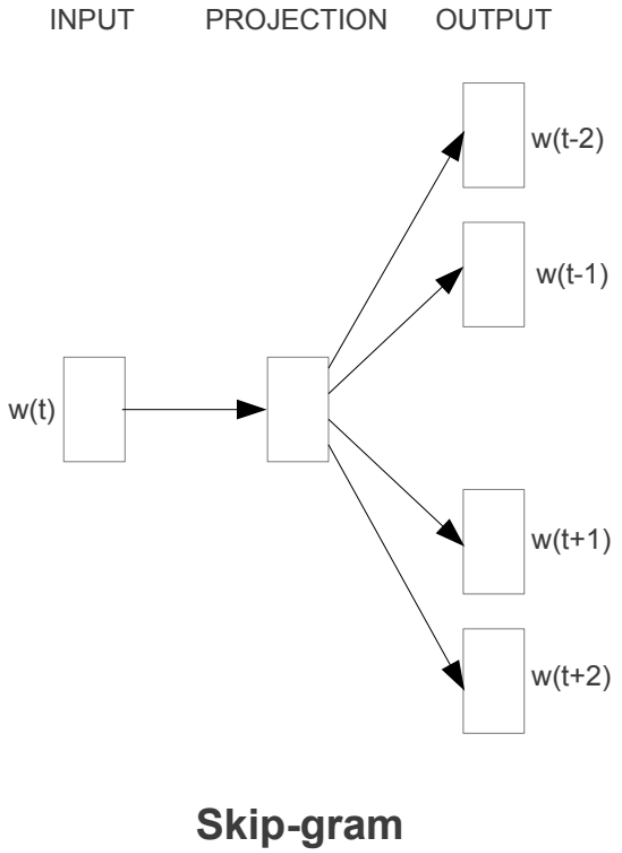


Figure 2: The Skip_gram architecture

Table 3: *Comparison of architectures using models trained on the same data, with 640-dimensional word vectors. The accuracies are reported on our Semantic-Syntactic Word Relationship test set, and on the syntactic relationship test set of [20]*

Model Architecture	Semantic-Syntactic Word Relationship test set		MSR Word Relatedness Test Set [20]
	Semantic Accuracy [%]	Syntactic Accuracy [%]	
RNNLM	9	36	35
NNLM	23	53	47
CBOW	24	64	61
Skip-gram	55	59	56

● Distributed Representations of Words and Phrases and their Compositionality

-The recently introduced continuous Skip-gram model is an efficient method for learning high-quality distributed vector representations that capture a large number of precise syntactic and semantic word relationships. In this section we present several extensions that improve both the quality of the vectors and the training speed. By subsampling the frequent words we obtain significant speedup and also learn more regular word representations. We also describe a simple alternative to the hierarchical softmax called negative sampling. An inherent limitation of word representations is their indifference to word order and their inability to represent idiomatic phrases. For example, the meanings of “Canada” and “Air” cannot be easily combined to obtain “Air Canada”

-Distributed representations of words in a vector space help learning algorithms to achieve better performance in natural language processing tasks by grouping similar words. One of the earliest uses of word representations dates back to 1986 due to Rumelhart, Hinton, and Williams . This idea has since been applied to statistical language modeling with considerable success

● The Skip-gram Model :

The training objective of the Skip-gram model is to find word representations that are useful for predicting the surrounding words in a sentence or a document. More formally, given a sequence of training words w1, w2, w3, . . . , wT , the objective of the Skip-gram model is to maximize the average log probability

$$\frac{1}{T}\sum_{t=1}^{T}\sum_{-c\leq j\leq c, j\neq 0}\log p(w_{t+j}|w_t)$$

where c is the size of the training context (which can be a function of the center word wt). Larger c results in more training examples and thus can lead to a higher accuracy, at the expense of the training time. The basic Skip-gram formulation defines p(wt+j |wt) using the softmax function:

$$p(w_O|w_I) = \frac{\exp\left({v'_{w_O}}^\top v_{w_I}\right)}{\sum_{w=1}^{W} \exp\left({v'_w}^\top v_{w_I}\right)}$$

where vw and v ′ w are the “input” and “output” vector representations of w, and W is the number of words in the vocabulary. This formulation is impractical because the cost of computing ∇ log p(wO|wI) is proportional to W, which is often large (105–107 terms).

• Negative Sampling

An alternative to the hierarchical softmax is Noise Contrastive Estimation (NCE), which was introduced by Gutmann and Hyvarinen and applied to language modeling by Mnih and Teh . NCE posits that a good model should be able to differentiate data from noise by means of logistic regression. This is similar to hinge loss used by Collobert and Weston who trained the models by ranking the data above noise. While NCE can be shown to approximately maximize the log probability of the softmax, the Skip Gram model is only concerned with learning high-quality vector representations, so we are free to simplify NCE as long as the vector representations retain their quality. We define Negative sampling (NEG) by the objective

$$\log \sigma({v'_{w_O}}^\top v_{w_I}) + \sum_{i=1}^{k} \mathbb{E}_{w_i \sim P_n(w)} \left[\log \sigma(-{v'_{w_i}}^\top v_{w_I})\right]$$

which is used to replace every log P(wO|wI) term in the Skip-gram objective. Thus the task is to distinguish the target word wO from draws from the noise distribution Pn(w) using logistic regression, where there are k negative samples for each data sample. Our experiments indicate that values of k in the range 5–20 are useful for small training datasets, while for large datasets the k can be as small as 2–5. The main difference between the Negative sampling and NCE is that NCE needs both samples and the numerical probabilities of the noise distribution, while Negative sampling uses only samples. And while NCE approximately maximizes the log probability of the softmax, this property is not important for our application. Both NCE and NEG have the noise distribution Pn(w) as a free parameter. We investigated a

number of choices for Pn(w) and found that the unigram distribution U(w) raised to the 3/4rd power (i.e., U(w) 3/4/Z) outperformed significantly the unigram and the uniform distributions, for both NCE and NEG on every task we tried including language modeling

Context Window and Positive Samples

- In Skip-Gram, the **context window** defines the range of words around a target word that are considered related or meaningful. For example, with a context window of 2, if the target word is "cat" in the sentence "The cat sat on the mat," the context words (positive samples) would be "The," "sat," "on," and "the."
- **Positive samples** are thus pairs formed by the target word and its actual context words within this window.

Negative Samples Outside the Window

- **Negative samples** are words that are **not within the context window** of the target word. In other words, these are random words selected from the vocabulary that do not co-occur with the target word in this specific context.
- For instance, if we have "cat" as the target word, a negative sample might be "car," "banana," or any other word that does not appear near "cat" in the current sentence. These words are sampled randomly to ensure they don’t have any actual association with the target word in the current context.

Why Negative Samples Are Outside the Window

By choosing words that are not in the context window as negative samples, the model learns to distinguish meaningful word relationships (positive samples) from random associations (negative samples). This helps the model position related words closer together in the embedding space, while unrelated words are positioned farther apart.

In short:

- **Positive samples** are words within the context window of the target word.
- **Negative samples** are words outside of the context window, selected to be unassociated with the target word in the current context.

How Negative Sampling Works in Skip-Gram

During training, the Skip-Gram model:

1. **Learns from Positive Samples**: The model adjusts the embeddings to increase the similarity between the target word and its context words, reinforcing the associations between them.
2. **Learns from Negative Samples**: The model also adjusts the embeddings to decrease the similarity between the target word and the negative samples, discouraging associations between unrelated words.

-The goal of **Noise-Contrastive Estimation (NCE)** is to transform the problem of **density estimation** (estimating probabilities) into a **binary classification problem** to make training large-scale probabilistic models more efficient. This technique is especially useful in training word embeddings and neural language models by helping them estimate probabilities without having to normalize across the entire vocabulary, which would be computationally expensive.

Key Goals of Negative Sampling:

1. **Efficient Learning**:
    - **Avoid Full Softmax Calculation**: In tasks like word embeddings, calculating the softmax over the entire vocabulary for each word prediction is computationally expensive. Negative sampling reduces this by only updating a small number of negative samples (randomly chosen words that do not appear in the context) alongside the positive sample (the actual context word).
    - Instead of considering all words in the vocabulary, negative sampling reduces the problem to binary classification (real vs. noise), making it much more computationally efficient.
2. **Speed Up Training**:
    - By limiting the number of word pairs the model needs to process, negative sampling speeds up the training process, allowing models to be trained on larger datasets in less time. In this way, it makes it feasible to train models with large vocabularies.
3. **Learn Word Representations (Embeddings)**:
    - Negative sampling encourages the model to learn meaningful word representations by distinguishing between **"positive samples"** (actual context words) and **"negative samples"** (randomly sampled

non-context words). This helps in positioning related words close to each other in the embedding space and unrelated words farther apart.
    - The model learns which words should be close (semantic or syntactic relationships) and which words should not (unrelated words).
4. **Reduce Memory Usage**:
    - Negative sampling helps keep memory usage low, as it does not require storing or computing the entire probability distribution across all words in the vocabulary. Instead, it focuses on a small number of negative samples for each positive word-context pair.

How Negative Sampling Works:

1. **Positive Sample**: The model is trained on a **positive sample** (a word pair where the target word actually appears with the context word).
2. **Negative Samples**: For each positive sample, several **negative samples** (random word pairs) are generated from words that do **not** appear in the context of the target word. The goal is to train the model to distinguish between real word pairs (positive) and randomly chosen ones (negative).

Why Negative Sampling Works:

- By focusing on just a few "negative" examples, the model can learn the relationship between words more efficiently than if it were forced to calculate and consider all possible word pairs.
- This technique is effective because it enables the model to **emphasize learning the most informative pairs** (both positive and negative) rather than processing the entire vocabulary.

- ***Word2Vec Notebook :***
  https://github.com/malekzitouni/Word2Vec_NLP/tree/main/docs/tutorials

• Gensim word vector visualization of various word vectors

- Gensim isn't really a deep learning package. It's a package for word and text similarity modeling, which started with (LDA-style) topic models and grew into SVD and neural word representations. But it's efficient and scalable, and quite widely used.
- GloVe is an unsupervised learning algorithm for obtaining vector representations for words. Training is performed on aggregated global word-word co-occurrence statistics from a corpus, and the resulting representations showcase interesting linear substructures of the word vector space.

- Nearest neighbors

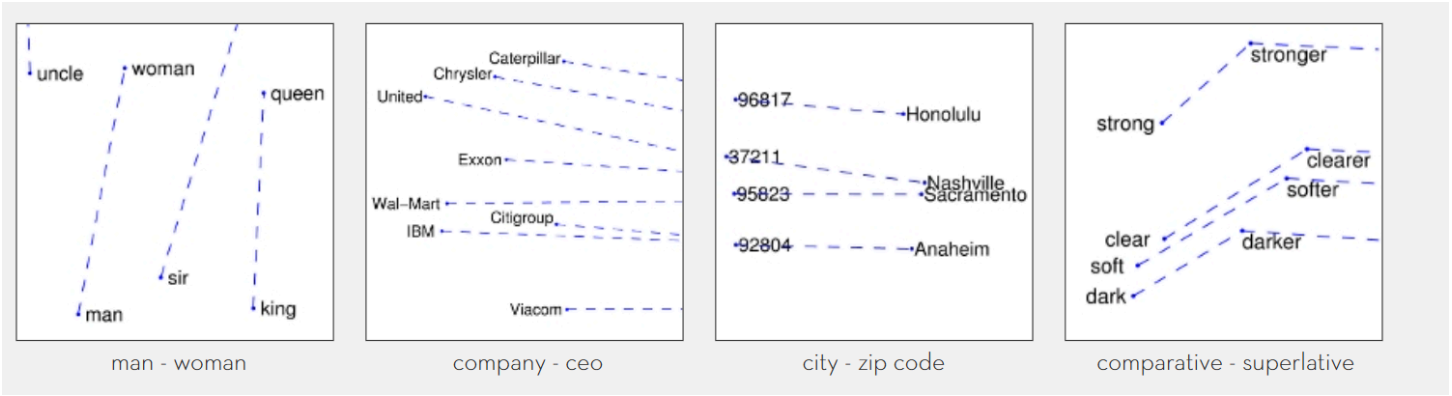
The Euclidean distance (or cosine similarity) between two word vectors provides an effective method for measuring the linguistic or semantic similarity of the corresponding words. Sometimes, the nearest neighbors according to this metric reveal rare but relevant words that lie outside an average human's vocabulary. For example, here are the closest words to the target word *frog*:

1. *frog*
2. frogs
3. toad
4. litoria
5. leptodactylidae
6. rana
7. lizard
8. eleutherodactylus

- Linear substructures

The similarity metrics used for nearest neighbor evaluations produce a single scalar that quantifies the relatedness of two words. This simplicity can be problematic since two given words almost always exhibit more intricate relationships than can be captured by a single number. For example, *man* may be regarded as similar to *woman* in that both words describe human beings; on the other hand, the two words are often considered opposites since they highlight a primary axis along which humans differ from one another.

In order to capture in a quantitative way the nuance necessary to distinguish *man* from *woman*, it is necessary for a model to associate more than a single number to the word pair. A natural and simple candidate for an enlarged set of discriminative numbers is the vector difference between the two word vectors. GloVe is designed in order that such vector differences capture as much as possible the meaning specified by the juxtaposition of two words.



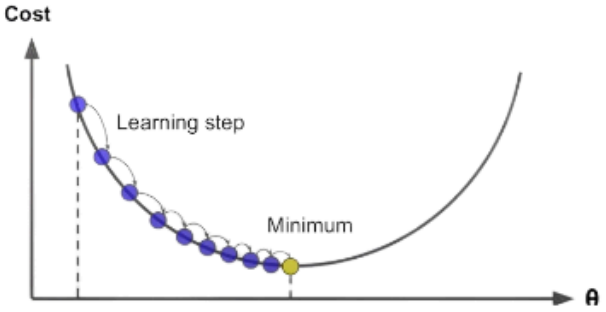
- Notebook Link :

https://github.com/malekzitouni/Word2Vec_NLP/blob/main/gensim_word_visualization.ipynb

● Neural Network Classifiers :

- Word2Vec maximizes objective function by putting similar words nearby in space
- Bag of words model makes the same prediction at each position
- Skip Gram model gives a reasonably high probability estimate to all words that occur in the context

- To learn good word vectors: We have a cost function $J(\theta)$ we want to minimize
- **Gradient Descent** is an algorithm to minimize $J(\theta)$ by changing θ
- **Idea**: from current value of θ, calculate gradient of $J(\theta)$, then take **small** step in the direction of negative gradient. Repeat.



Note: Our objectives may not be convex like this ☹

But life turns out to be okay ☺

Stochastic Gradient Descent

- **Problem**: $J(\theta)$ is a function of **all** windows in the corpus (often, billions!)
  - So $\nabla_\theta J(\theta)$ is very expensive to compute
- You would wait a very long time before making a single update!

- **Very** bad idea for pretty much all neural nets!
- **Solution: Stochastic gradient descent (SGD)**
  - Repeatedly sample windows, and update after each one, or each small batch

- Algorithm:

```
while True:
    window = sample_window(corpus)
    theta_grad = evaluate_gradient(J,window,theta)
    theta = theta - alpha * theta_grad
```

Why two vectors? → Easier optimization. Average both at the end

- But can implement the algorithm with just one vector per word … and it helps

Two model variants:

1. Skip-grams (SG)
   Predict context ("outside") words (position independent) given center word
2. Continuous Bag of Words (CBOW)
   Predict center word from (bag of) context words

We presented: **Skip-gram model**

Additional efficiency in training:

1. Negative sampling

The skip-gram model with negative sampling (HW2)

- The normalization term is computationally expensive

- $P(o|c) = \dfrac{\exp(u_o^T v_c)}{\sum_{w \in V} \exp(u_w^T v_c)}$

- From paper: "Distributed Representations of Words and Phrases and their Compositionality" (Mikolov et al. 2013)
- Overall objective function (they maximize): $J(\theta) = \frac{1}{T}\sum_{t=1}^{T} J_t(\theta)$

$$J_t(\theta) = \log \sigma\left(u_o^T v_c\right) + \sum_{i=1}^{k} \mathbb{E}_{j \sim P(w)}\left[\log \sigma\left(-u_j^T v_c\right)\right]$$

- Notation more similar to class and HW2:

$$J_{neg-sample}(\boldsymbol{u}_o, \boldsymbol{v}_c, U) = -\log \sigma(\boldsymbol{u}_o^T \boldsymbol{v}_c) - \sum_{k \in \{K \text{ sampled indices}\}} \log \sigma(-\boldsymbol{u}_k^T \boldsymbol{v}_c)$$

- We take *k* negative samples (using word probabilities)
- Maximize probability that real outside word appears, minimize probability that random words appear around center word

Towards GloVe: Count based vs. direct prediction

- LSA, HAL (Lund & Burgess),
- COALS, Hellinger-PCA (Rohde et al, Lebret & Collobert)

- Fast training
- Efficient usage of statistics
- Primarily used to capture word similarity
- Disproportionate importance given to large counts

- Skip-gram/CBOW (Mikolov et al)
- NNLM, HLBL, RNN (Bengio et al; Collobert & Weston; Huang et al; Mnih & Hinton)

- Scales with corpus size
- Inefficient usage of statistics
- Generate improved performance on other tasks
- Can capture complex patterns beyond word similarity

● Machine Learning I Building Large Language Models (LLMs)

● Pre-Training in NLP, LLMs, and Deep Learning :

Definition: Pre-training refers to the initial phase of training a language model on a large, general-purpose dataset, often unsupervised. The goal is to teach the model foundational knowledge about language, such as grammar, semantics, and common patterns.
Process:

- Uses techniques like masked language modeling (e.g., BERT) or causal language modeling (e.g., GPT).
- Trains on massive text datasets like Wikipedia, Common Crawl, or books.
- Models learn general linguistic structures, word relationships, and context.

Advantages:

- Provides a strong baseline understanding of language.
- Reduces the need for large labeled datasets for specific tasks.
- Enables transfer learning: the pre-trained model can be adapted to various downstream tasks.

Examples:

- **BERT** (Bidirectional Encoder Representations from Transformers): Pre-trained using masked language modeling and next sentence prediction.
- **GPT** (Generative Pre-trained Transformer): Pre-trained with autoregressive language modeling.

How It Works:

- **Self-supervised learning**: Models create pseudo-labels from the data itself (no manual labeling required). Common approaches include:
    - **Masked Language Modeling (MLM)**: Some tokens are masked, and the model predicts them (e.g., BERT).

    - **Causal Language Modeling**: Predicts the next token in a sequence (e.g., GPT).
    - **Next Sentence Prediction (NSP)**: Determines if two sentences follow each other (used in BERT pre-training).

Key Models and Techniques:

- **Transformer architecture**: Most LLMs use Transformers, a deep learning model designed for handling sequential data with self-attention mechanisms.
- **LLMs**: Examples include GPT, BERT, T5, and PaLM. These models are trained on diverse datasets like Common Crawl, BooksCorpus, and Wikipedia.

Outcomes:

- The model learns generalized embeddings, representing words and sentences in high-dimensional vector spaces.
- Pre-training results in a model that can generate coherent, context-aware text or analyze language patterns.

Advantages:

- **Transferability**: The pre-trained model serves as a base for many tasks.
- **Efficiency**: Reduces the need for task-specific labeled data.
- **Scalability**: Larger datasets and models improve generalization

● Post-Training (Fine-Tuning) in NLP, LLMs, and Deep Learning

Definition: Post-training, often called **fine-tuning**, is the process of adapting a pre-trained model to a specific NLP task or domain. This phase uses task-specific labeled datasets.
Process:

- The pre-trained model is initialized with its learned weights.
- Additional training is performed on task-specific data (e.g., sentiment analysis, named entity recognition, question answering).
- The model may also undergo hyperparameter optimization or structural modifications (e.g., adding task-specific layers).

Advantages:

- Tailors the pre-trained model to solve specific problems effectively.
- Requires less computational resources and labeled data compared to training a model from scratch.
- Adapts the model to domain-specific language or jargon (e.g., medical or legal texts).

How It Works:

- **Supervised learning**: Fine-tuning requires labeled data specific to the target task.
- **Task-specific objectives**: Examples include:
    - **Classification**: Assigning labels (e.g., sentiment analysis).
    - **Sequence-to-sequence tasks**: Translating or summarizing text (e.g., T5, BART).
    - **Span prediction**: Extracting answers from passages (e.g., SQuAD dataset for QA).

Key Methods:

1. **Learning rate adjustment**:
    - Fine-tuning uses smaller learning rates to preserve pre-trained knowledge.
2. **Regularization**:
    - Prevents overfitting by applying techniques like dropout or weight decay.
3. **Adapters**:
    - Lightweight modules added to pre-trained models to reduce fine-tuning costs (e.g., LoRA, prefix-tuning).
4. **Data augmentation**:
    - Expands labeled datasets with paraphrasing or synthetic data generation.

Outcomes:

- A specialized model optimized for specific tasks or industries, such as customer support, legal document analysis, or creative writing.

● Model Tuning vs Model fine_tuning :

Model Fine-Tuning

- **What it is**: Fine-tuning starts with a **pre-trained model**, which is a model that has already been trained on a large dataset. It takes the **learned weights** from this model as the starting point and adjusts them by retraining the model on a new, smaller, and typically task-specific dataset.
- **Steps**:
    - Load a pre-trained model (e.g., ResNet for image recognition, GPT for text).
    - Use the weights of the pre-trained model as **initial weights**.
    - Train the model on a custom dataset for a specific task.
        - This training updates the weights to make the model specialize in the task (e.g., medical diagnosis, sentiment analysis).
    - After retraining, the fine-tuned model is ready for **inference** on the specific task.
- **Key Features**:
    - Saves time and computational resources compared to training from scratch.
    - Requires a **smaller dataset** since the model already contains generalized knowledge from pre-training.
    - Focuses on adapting the model to a specific task (e.g., from general text generation to legal text generation).
- **Use Case**: Fine-tune a large language model like GPT for customer service by training it on customer query-response pairs.

=> Fine-tuning is the most resource-intensive, involving a comprehensive re-training of the model on a specific dataset for a specific purpose. This adjusts the weights of the pre-trained model, optimizing it for detailed nuances of the data but requiring substantial computational resources and increasing the risk of overfitting. Many LLMs like Chat GPT undergo fine-tuning after their initial generic training on the next word prediction task. Fine-tuning teaches these models how to function as digital assistants, making them significantly more useful than a generally trained model.

Tuning

- **What it is**: In this context, "tuning" refers to **training a model from scratch**, which involves starting with **randomly initialized weights** and training the model on a dataset to learn everything from the ground up.
- **Steps**:
    - Define the model architecture (e.g., a neural network).
    - Initialize all weights randomly.
    - Train the model on a large dataset, adjusting the weights from scratch.
    - After training, use the fully trained model for inference.
- **Key Features**:
    - Requires a **large dataset** to allow the model to learn from zero.
    - Computationally expensive and time-consuming compared to fine-tuning.
    - Used when no suitable pre-trained model exists, or when developing a model with a unique architecture or purpose.
- **Use Case**: Train a brand-new image recognition model from scratch to classify rare astronomical objects where no pre-trained model applies.

Efficient Methods of Tuning

Efficient tuning methods aim to adapt pre-trained models to specific tasks or domains while minimizing computational costs, data requirements, and time. These methods often avoid retraining the entire model, instead making targeted adjustments to a smaller subset of parameters or using novel strategies to guide the model's output.

---

1. Prompt Tuning

Définition:

Prompt tuning is a **parameter-efficient fine-tuning technique** that focuses on optimizing **soft prompts** (learnable embeddings) while keeping the pre-trained model weights frozen. Instead of retraining the entire model, only the prompts are learned and adjusted to guide the model toward the desired task-specific behavior.

How It Works:

1. **Learnable Prompt Embeddings**:

    - Instead of using static, manually crafted text prompts, prompt tuning introduces trainable embeddings as the "prompt."
    - These embeddings are concatenated to the input tokens and are optimized during training.
2. **Model Weights Remain Fixed**:
    - The pre-trained model's weights are not updated.
    - This drastically reduces the number of trainable parameters compared to full fine-tuning.
3. **Task-Specific Adaptation**:
    - By training the soft prompts on a task-specific dataset, the model can specialize in the task without modifying its core knowledge.

Advantages:

- **Parameter-Efficient**: Only a small number of parameters (the soft prompts) are updated.
- **Scalable**: Easily adapts large models like GPT-3 or T5 without requiring the massive computational resources needed for full fine-tuning.
- **Reusable Model**: The pre-trained model remains unchanged, allowing it to be reused for other tasks.

Use Case:

- Adapting a language model to classify customer sentiment by optimizing task-specific prompts without altering the model's general language understanding capabilities.

=> **Prompt tuning adjusts a set of extra parameters, known as "soft prompts," which are integrated into the model's input processing. This method modifies how the model interprets input prompts without a complete overhaul of its weights, offering a balance between performance enhancement and resource efficiency. It is particularly valuable when computational resources are limited or when flexibility across multiple tasks is required, because after applying the technique the original model weights remain unchanged.**

---

2. Other Efficient Tuning Methods

a. LoRA (Low-Rank Adaptation)

- Instead of fine-tuning all the model's weights, LoRA introduces **low-rank trainable matrices** that are added to the model's layers.

- These matrices adjust the outputs of specific layers, enabling task-specific adaptation without updating most of the original weights.

b. Adapter Layers

- Trainable layers (adapters) are inserted between the frozen layers of the pre-trained model.
- These layers are optimized for the new task while leaving the original model weights intact.

c. Prefix Tuning

- Similar to prompt tuning, but it prepends trainable embeddings (prefixes) to the attention layers of the model rather than just the input tokens

● PaLM (Pathways Language Model)

What is PaLM?

PaLM (Pathways Language Model) is a large-scale language model developed by **Google Research**, designed to handle diverse tasks with state-of-the-art performance. It is built using **Google's Pathways system**, which enables a single model to generalize across multiple tasks and modalities efficiently.

Key Features:

1. **Scalability**:
    - PaLM is one of the largest language models, trained with **540 billion parameters**.
    - It leverages **dense transformers** and is trained on a vast, multi-modal dataset.
2. **Pathways System**:
    - Pathways is a highly scalable infrastructure that allows a single model to perform multiple tasks (text, vision, etc.).
    - Efficiently routes tasks to the appropriate sub-models or components of the system.

3. **Few-shot and Zero-shot Learning**:
    - Excels in few-shot and zero-shot learning, meaning it can adapt to tasks with little or no additional fine-tuning.
4. **Task Generalization**:
    - Handles diverse NLP tasks like translation, summarization, question answering, reasoning, and more.
    - Demonstrates strong reasoning capabilities, including solving arithmetic word problems and logical puzzles.

Applications:

- **Conversational AI**: Enhanced chatbot capabilities.
- **Code Generation**: Writing and debugging code.
- **Creative Writing**: Generating poetry, stories, or other forms of text.
- **Scientific Applications**: Assisting in research with reasoning-based ta

• LaMDA (Language Model for Dialogue Applications)

What is LaMDA?

LaMDA is a **conversational AI model** developed by **Google AI**, specifically designed to handle **open-domain dialogue** with high fluency and relevance. Unlike traditional language models, LaMDA is optimized to generate coherent, contextually appropriate, and engaging responses in a conversational setting.

Key Features:

1. **Dialogue Specialization**:
    - Trained on dialogue-specific datasets to understand the nuances of conversational language.
    - Focuses on creating responses that are logical, contextually aware, and flow naturally.
2. **Open-ended Conversations**:
    - Can handle open-domain conversations, meaning it can discuss a wide range of topics without being confined to a specific task.

3. **Sense of Specificity and Safety**:
    - Designed to avoid generic or overly vague responses by ensuring specificity in replies.
    - Incorporates mechanisms to filter out harmful or inappropriate content, ensuring safe interactions.
4. **Knowledgeable Yet Adaptable**:
    - Generates responses that incorporate factual knowledge but can also engage in imaginative, exploratory dialogue.

Applications:

- **Chatbots**: Powering conversational agents for customer service or personal assistants.
- **Education**: Helping students by answering questions or tutoring interactively.
- **Entertainment**: Engaging in creative, open-ended discussions.
- **Companionship AI**: Providing meaningful and empathetic conversations.

The long road to LaMDA

LaMDA's conversational skills have been years in the making. Like many recent language models, including BERT and GPT-3, it's built on Transformer, a neural network architecture that Google Research invented and open-sourced in 2017. That architecture produces a model that can be trained to read many words (a sentence or paragraph, for example), pay attention to how those words relate to one another and then predict what words it thinks will come next.

But unlike most other language models, LaMDA was trained on dialogue. During its training, it picked up on several of the nuances that distinguish open-ended conversation from other forms of language. One of those nuances is sensibleness. Basically: Does the response to a given conversational context make sense? For instance, if someone says:

"I just started taking guitar lessons."

You might expect another person to respond with something like:

"How exciting! My mom has a vintage Martin that she loves to play."

That response makes sense, given the initial statement. But sensibleness isn't the only thing that makes a good response. After all, the phrase "that's nice" is a sensible response to nearly any statement, much in the way "I don't know" is

a sensible response to most questions. Satisfying responses also tend to be specific, by relating clearly to the context of the conversation.

• Prompt Tuning :

- Prompt tuning is a technique used to improve the performance of a pre-trained language model without modifying the model's internal architecture.
- In the fast evolving field of large language models (LLMs), staying on top of state of art approaches like prompt tuning is crucial. This technique, applied to already trained foundational models, enhances performance without the high computational costs associated with traditional model training.
- Instead of modifying the deep structural weights of the model, prompt tuning adjusts the prompts that guide the model's response. This method is based on the introduction of "soft prompts," a set of tunable parameters inserted at the beginning of the input sequence.

- Initialization of soft prompts : Soft prompts are artificially constructed tokens that are added to the model's input sequence. These prompts can be initialized in several ways. Random initialization is common, however they can also be initialized based on certain heuristics. Once initialized, soft prompts are attached to the start of the input data. When the model processes this data, it considers both the soft prompts and the actual input.
- Forward Pass and loss evaluation : The training process is usually similar to that of training a standard deep neural network (DNN). It starts with a forward pass where the model processes the combined input through its layers, producing an output. This output is then evaluated against the desired outcome using a loss function, which measures the discrepancy between the model's output and the actual expected value. During backpropagation, the errors are propagated back through the network. However, instead of adjusting the network's weights, we only modify the soft prompt parameters. This process repeats across multiple epochs, with the soft prompts gradually learning to shape the model's processing of inputs in such a way that minimizes the error for the given task

- **Prompt engineering**

Prompt engineering involves no training or retraining at all. It is completely based on the user designing prompts for the model. It requires a nuanced understanding of the model's processing capabilities and leverages the intrinsic knowledge embedded within the model. Prompt engineering does not require any computational resources since it relies solely on the strategic formulation of inputs to achieve results.

-Prompt engineering is the technical process of constructing and optimizing prompts to maximize the performance of a language model. It involves strategic techniques to structure the prompt for precise, repeatable, and task-specific outputs.

- **Technical:**
  - **Focuses on how to structure the prompt technically for better model performance.**
  - **Involves detailed instructions, token placement, and formatting.**
- **Optimization-Oriented:**
  - **Ensures efficiency and accuracy by tweaking details like:**
    - **Explicit instructions.**
    - **Examples of desired input-output behavior (few-shot learning).**
    - **Context or delimiters to guide model behavior.**
- **Practical Application:**
  - **Prompt engineering often uses systematic testing and evaluation to tune performance for specialized tasks.**

● Prompt :

A prompt in the context of artificial intelligence, particularly large language models (LLMs), is an input instruction or query provided to the

model to elicit a specific output. Prompts act as the starting point for the model to generate responses, and they play a critical role in guiding the behavior of AI systems. The design of a prompt affects how the model interprets the input, structures its response, and performs a given task.

Types of Prompts

- **Discrete Prompts**: Natural language instructions written in text, like "Write a poem about autumn."
- **Soft Prompts**: Learnable embeddings in vector space that act as task-specific inputs. These are used in prompt tuning and are not human-readable.
- **Few-shot Prompts**: Provide a few examples of input-output pairs to guide the model's task performance.
- **Zero-shot Prompts**: Ask the model to perform a task without providing any examples, relying solely on its pre-trained knowledge.

How Prompts Work

Prompts guide the model by leveraging its underlying **knowledge and language patterns**. Language models, like GPT, are trained on vast datasets of text and learn to predict the next word or sequence based on the given input. A well-designed prompt ensures the model:

- Understands the task or goal.
- Retrieves relevant knowledge encoded in its parameters.
- Generates an output aligned with the user's expectations.

● Types of Large Language Models :

1. Autoregressive Models

- **Examples**: GPT (Generative Pre-trained Transformer) series, OpenAI's Codex.
- **How They Work**:
    - Predict the next word in a sequence, working left to right in a text.

    - Trained to maximize the likelihood of the next token based on previous tokens.
- **Key Strengths**:
    - Excellent for generative tasks like text generation, storytelling, or coding assistance.
- **Limitations**:
    - Struggle with bidirectional context (limited understanding of the "whole picture" at once).
- **Applications**:
    - Chatbots, content creation, creative writing, programming code generation.

---

2. Masked Language Models (Bidirectional Models)

- **Examples**: BERT (Bidirectional Encoder Representations from Transformers), RoBERTa, DistilBERT.
- **How They Work**:
    - Learn bidirectional context by masking parts of the input text and training the model to predict the masked tokens.
    - This allows the model to consider both the left and right contexts simultaneously.
- **Key Strengths**:
    - Strong performance on understanding and analysis tasks, such as sentiment analysis, text classification, and answering questions.
- **Limitations**:
    - Not well-suited for generating coherent long-form text.
- **Applications**:
    - Text classification, information retrieval, natural language understanding.

---

3. Sequence-to-Sequence (Seq2Seq) Models

- **Examples**: T5 (Text-to-Text Transfer Transformer), BART (Bidirectional and Auto-Regressive Transformers).
- **How They Work**:
    - Use an encoder-decoder structure: the encoder processes the input, and the decoder generates the output.
    - Trained to map one sequence to another, such as translating languages or summarizing text.

- **Key Strengths**:
  - Effective for structured transformations of text.
- **Limitations**:
  - Require task-specific tuning and are more resource-intensive than autoregressive models.
- **Applications**:
  - Machine translation, summarization, and question-answering.

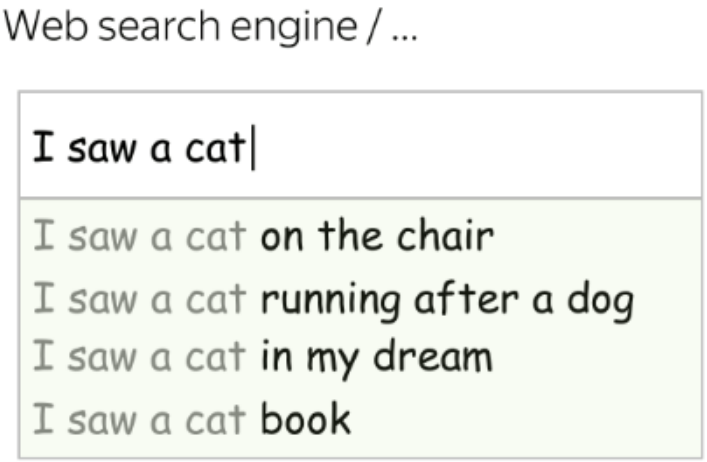
What about language?

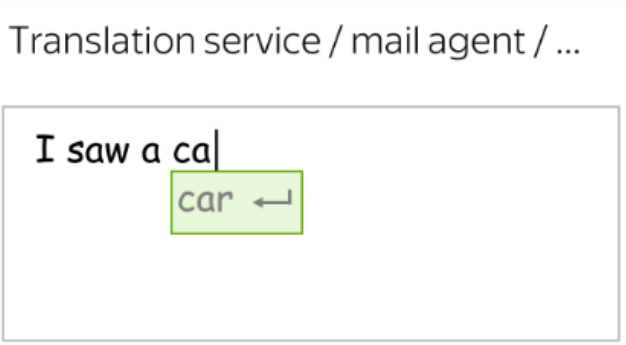
For language, the intuition is exactly the same! What is different, is the notion of an event. In language, an event is a linguistic unit (text, sentence, token, symbol), and a goal of a language model is to estimate the probabilities of these events.

Language Models (LMs) estimate the probability of different linguistic units: symbols, tokens, token sequences.

We deal with LMs every day!

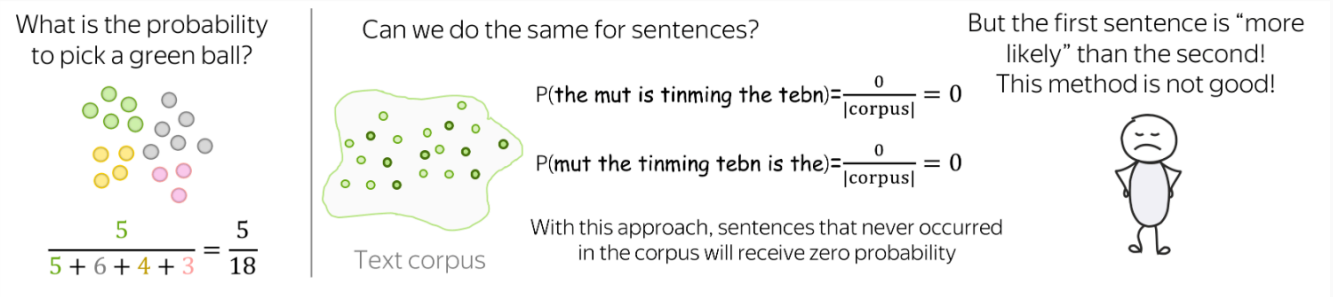
We see language models in action every day - look at some examples. Usually models in large commercial services are a bit more complicated than the ones we will discuss today, but the idea is the same: if we can estimate probabilities of words/sentences/etc, we can use them in various, sometimes even unexpected, ways.





But how is a machine supposed to understand human language ? A machine needs a language model, which estimates the probabilities of sentences. If a language model is good, it will assign a larger probability to the correct option.

How likely is a sentence to appear in a language?



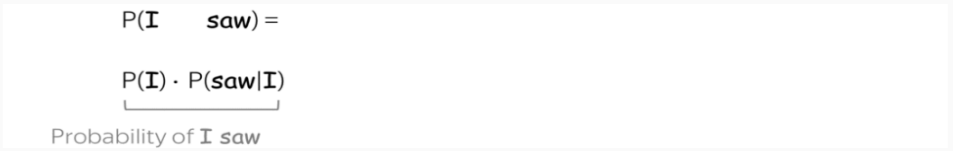
Let's check if simple probability theory can help. Imagine we have a basket with balls of different colors. The probability to pick a ball of a certain color (let's say green) from this basket is the frequency with which green balls occur in the basket.

What if we do the same for sentences? Since we can not possibly have a text corpus that contains all sentences in a natural language, a lot of sentences will not occur in the corpus. While among these sentences some are clearly more likely than the others, all of them will receive zero probability, i.e., will look equally bad for the model. This means, the method is not good and we have to do something more clever.

Sentence Probability: Decompose Into Smaller Parts

We can not reliably estimate sentence probabilities if we treat them as atomic units. Instead, let's decompose the probability of a sentence into probabilities of smaller parts.

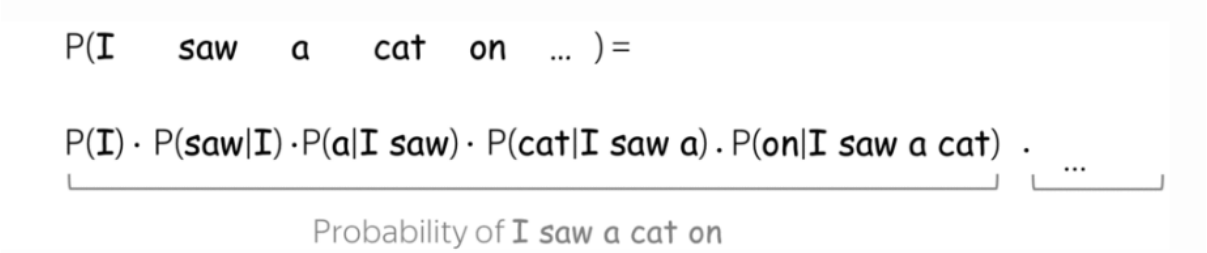
For example, let's take the sentence **I saw a cat on a mat** and imagine that we read it word by word. At each step, we estimate the probability of all seen so far tokens. We don't want any computations not to be in vain (no way!), so we won't throw away previous probability once a new word appears: we will update it to account for a new word. Look at the illustration.

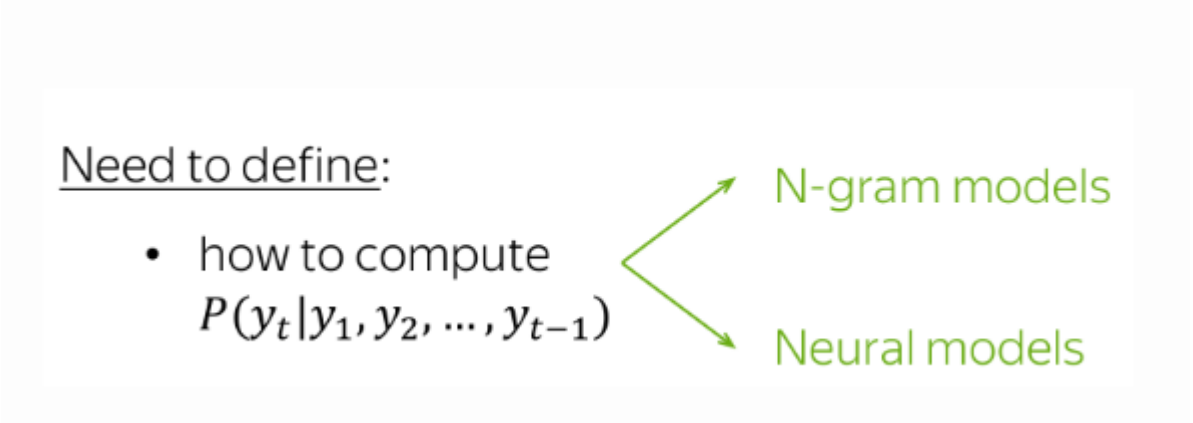


Formally, let $y_1, y_2, \ldots, y_n$ be tokens in a sentence, and $P(y_1, y_2, \ldots, y_n)$ the probability to see all these tokens (in this order). Using the product rule of probability (aka the chain rule), we get

$$P(y_1, y_2, \ldots, y_n) = P(y_1) \cdot P(y_2|y_1) \cdot P(y_3|y_1, y_2) \cdot \cdots \cdot P(y_n|y_1, \ldots, y_{n-1}) = \prod_{t=1}^{n} P(y_t|y_{<t}).$$

We decomposed the probability of a text into conditional probabilities of each token given the previous context.





Once we have a language model, we can use it to generate text. We do it one token at a time: predict the probability distribution of the next token given previous context, and sample from this distribution.

- Alternatively, you can apply greedy decoding: at each step, pick the token with the highest probability. However, this usually does not work well

Markov Property (Independence Assumption)

The straightforward way to compute $P(y_t|y_1, \ldots, y_{t-1})$ is

$$P(y_t|y_1, \ldots, y_{t-1}) = \frac{N(y_1, \ldots, y_{t-1}, y_t)}{N(y_1, \ldots, y_{t-1})},$$

where $N(y_1, \ldots, y_k)$ is the number of times a sequence of tokens $(y_1, \ldots, y_k)$ occur in the text.

For the same reasons we discussed before, this won't work well: many of the fragments $(y_1, \ldots, y_t)$ do not occur in a corpus and, therefore, will zero out the probability of the sentence. To overcome this problem, we make an independence assumption (assume that the Markov property holds):

> The probability of a word only depends on a **fixed** number of previous words.

Formally, n-gram models assume that

$$P(y_t|y_1, \ldots, y_{t-1}) = P(y_t|y_{t-n+1}, \ldots, y_{t-1}).$$

For example,

- n=3 (trigram model): $P(y_t|y_1, \ldots, y_{t-1}) = P(y_t|y_{t-2}, y_{t-1})$,
- n=2 (bigram model): $P(y_t|y_1, \ldots, y_{t-1}) = P(y_t|y_{t-1})$,
- n=1 (unigram model): $P(y_t|y_1, \ldots, y_{t-1}) = P(y_t)$.

Smoothing: Redistribute Probability Mass

Let's imagine we deal with a 4-gram language model and consider the following example:

$$\text{P(mat | I saw a cat on a)} = \text{P(mat | cat on a)} = \frac{\text{N(cat on a mat)}}{\text{N(cat on a)}}$$

What if either denominator or numerator is zero? Both these cases are not really good for the model. To avoid these problems (and some other), it is common to use smoothings. Smoothings redistribute probability mass: they "steal" some mass from seen events and give to the unseen ones.

Avoid zeros in the denominator

If the phrase cat on a never appeared in our corpus, we will not be able to compute the probability. Therefore, we need a "plan B" in case this happens.

$$P(\text{mat} \mid \text{cat on a}) = \frac{N(\text{cat on a mat})}{N(\text{cat on a})} = ?$$

$N(\text{cat on a})$: zero

not good: can not compute the probability

Backoff (aka Stupid Backoff)

One of the solutions is to use less context for context we don't know much about. This is called backoff:

- if you can, use trigram;
- if not, use bigram;
- if even bigram does not help, use unigram.

This is rather stupid (hence the title), but works fairly well.

$N(\text{cat on a}) = 0 \longrightarrow$ try "on a"

$P(\text{mat} \mid \text{cat on a}) \approx P(\text{mat} \mid \text{on a})$

$N(\text{on a}) = 0 \longrightarrow$ try "a"

$P(\text{mat} \mid \text{on a}) \approx P(\text{mat} \mid \text{a})$

$N(\text{a}) = 0 \longrightarrow$ try unigram

$P(\text{mat} \mid \text{a}) \approx P(\text{mat})$

More clever: Linear interpolation

A more clever solution is to mix all probabilities: unigram, bigram, trigram, etc. For this, we need scalar positive weights $\lambda_0, \lambda_1, \ldots, \lambda_{n-1}$ such that $\sum_i \lambda_i = 1$. Then the updated probability is:

$$\begin{aligned}\widehat{P}(\text{mat} \mid \text{cat on a}) \approx\ & \lambda_3 P(\text{mat} \mid \text{cat on a}) + \\ & \lambda_2 P(\text{mat} \mid \text{on a}) + \\ & \lambda_1 P(\text{mat} \mid \text{a}) + \\ & \lambda_0 P(\text{mat})\end{aligned}$$

Avoid zeros in the numerator

If the phrase **cat on a mat** never appeared in our corpus, the probability of the whole sentence will be zero - but this does not mean that the sentence is impossible! To avoid this, we also need a "plan B".

$$P(\text{mat} \mid \text{cat on a}) = \frac{\overbrace{N(\text{cat on a mat})}^{\text{zero}}}{N(\text{cat on a})} = ?$$

not good: zeros out probability of the whole sentence

Laplace smoothing (aka add-one smoothing)

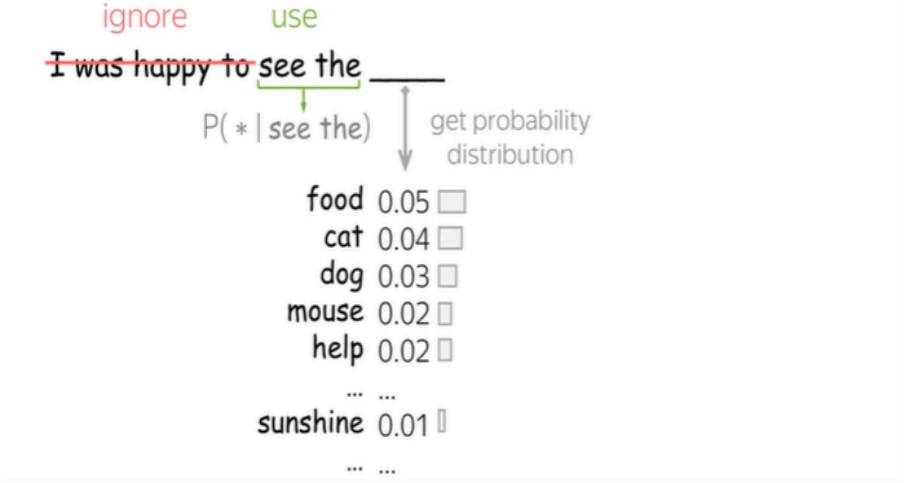
The simplest way to avoid this is just to pretend we saw all n-grams at least one time: just add 1 to all counts! Alternatively, instead of 1, you can add a small δ:

$$\hat{P}(\text{mat} \mid \text{cat on a}) = \frac{\delta + N(\text{cat on a mat})}{\delta \cdot |V| + N(\text{cat on a})}$$

- The most popular smoothing for n-gram LMs is Kneser-Ney smoothing

Generation (and Examples)

The generation procedure for a n-gram language model is the same as the general one: given current context (history), generate a probability distribution for the next token (over all tokens in the vocabulary), sample a token, add this token to the sequence, and repeat all steps again. The only part which is specific to n-gram models is the way we compute the probabilities. Look at the illustration.



• Project : Optical Character Recognition (OCR):

Use n-grams to validate or correct text extracted from scanned documents.

- OCR often introduces errors when processing scanned documents due to poor image quality, unusual fonts, or misaligned text. N-grams can help detect and correct these errors by validating text sequences against common language patterns

1. Problem Setup

- **Input:**
  Text extracted from an OCR system (may contain errors like "he11o" instead of "hello").
- **Goal:**
  Use n-grams to correct or suggest fixes for OCR errors by identifying unlikely or invalid word sequences.

- **Github Link** : https://github.com/malekzitouni/OCR-using-N_gram-approach

An **OCR (Optical Character Recognition) engine** is a software system that converts scanned images, photographs, or handwritten or printed text into machine-readable digital text. It enables computers to extract and process text from images, PDFs, or other visual formats. OCR engines are widely used in document digitization, automated data entry, and text extraction for analysis.

How OCR Engines Work

OCR engines use a combination of image processing, machine learning, and pattern recognition techniques to identify and digitize text. The process can be divided into the following steps:

1. Image Preprocessing

- Enhances the input image for better accuracy.
- Common techniques include:
  - Noise removal.

    - Binarization (converting to black-and-white images).
    - Deskewing (straightening tilted text).
    - Rescaling and cropping.

2. Text Detection

- Locates regions of the image likely to contain text.
- Approaches include:
    - Sliding window methods.
    - Convolutional Neural Networks (CNNs) for detecting bounding boxes.

3. Character Segmentation

- Breaks down text into individual characters or words.
- Involves identifying baselines, spacing, and letter boundaries.

4. Character Recognition

- Compares each character with a database of known shapes (glyphs) or uses a machine learning model to classify characters.
- Two main approaches:
    - **Template Matching:** Matches each character with stored templates.
    - **Feature Extraction:** Identifies unique features of each character (e.g., edges, strokes) for classification.

5. Post-Processing

- Enhances the accuracy of recognition by correcting errors using:
    - Language models (e.g., n-grams).
    - Dictionaries for valid words.
    - Context-aware corrections (e.g., "hte" → "the").

● Tuple :

A **tuple** is an ordered collection of elements in programming. It is similar to a list, but unlike lists, tuples are **immutable**. This means that once a tuple is created, its contents cannot be changed (i.e., no additions, deletions, or modifications of elements).

Characteristics of a Tuple:

1. **Ordered**: The elements in a tuple have a defined order, meaning that their position within the tuple matters.
2. **Immutable**: Once a tuple is created, you cannot modify its elements. This makes tuples safer in situations where you don't want the data to be changed.
3. **Can store mixed data types**: A tuple can hold elements of different types, such as integers, strings, lists, or even other tuples.
4. **Defined using parentheses**: A tuple is typically defined using parentheses `()`.

OCR engines

- tesseract - The definitive Open Source OCR engine `Apache 2.0`
- EasyOCR - OCR engine built on PyTorch by JaidedAI, `Apache 2.0`
- ocropus - OCR engine based on LSTM, `Apache 2.0`
- ocropus 0.4 - Older v0.4 state of Ocropus, with tesseract 2.04 and iulib, C++
- kraken - Ocropus fork with sane defaults
- gocr - OCR engine under the GNU Public License led by Joerg Schulenburg.
- Ocrad - The GNU OCR. `GPL`
- ocular - Machine-learning OCR for historic documents
- SwiftOCR - fast and simple OCR library written in Swift
- attention-ocr - OCR engine using visual attention mechanisms
- RWTH-OCR - The RWTH Aachen University Optical Character Recognition System
- simple-ocr-opencv and its fork - A simple pythonic OCR engine using opencv and numpy
- Calamari - OCR Engine based on OCRopy and Kraken
- doctr - A seamless & high-performing OCR library powered by Deep Learning

– The goal of using **n-grams** (bigrams, trigrams, etc.) instead of just using **tokens** directly in tasks like **OCR error correction**, **language modeling**, or **text analysis** is to capture **contextual relationships** between words and improve understanding or prediction based on word sequences. Here's why n-grams are valuable:

1. Capturing Context

- **Single tokens** (individual words) do not capture the relationship between consecutive words. For example, the word "jumped" might have different meanings or interpretations depending on the words before or after it. By using n-grams (like bigrams or trigrams), you capture the **context** in which a word appears, which helps in identifying errors and understanding word relationships more effectively.
  For example:
  - **Tokens**: ["the", "quick", "fox"]
  - **Bigrams**: [("the", "quick"), ("quick", "fox")]
- In the bigram ("quick", "fox"), we can understand that "quick" is more likely to be followed by "fox" than by something like "dog", and it helps to detect errors in OCR output (like mistaking "fox" for "box").

2. Handling OCR Errors and Unlikely Sequences

OCR systems may misinterpret individual words, especially in noisy or low-quality images. In many cases, the OCR engine might produce a sequence of words that are grammatically or semantically incorrect, but a direct token analysis would miss that. By using n-grams:

- You can detect **improbable word combinations**. For example, a bigram like ("br0wn", "b0x") would stand out as incorrect in a corpus, as “brown” would almost certainly be followed by "fox", not "b0x".

By analyzing n-grams instead of just individual tokens, you have a better chance of recognizing these **unlikely sequences** and flagging potential OCR errors.

3. Language Modeling

N-grams are commonly used in **language models** to predict the probability of the next word based on the current sequence of words. For instance, given the bigram ("quick", "brown"), a language model can use the probability distribution

learned from a corpus to predict that the next word is likely to be "fox", based on how common that sequence is.

This can be helpful for:

- **Spell checking**: Correcting OCR errors based on common sequences of words.
- **Text prediction**: If you want to predict the next word in a sequence, n-grams help capture this dependency.

4. Improved Error Detection

By generating n-grams, we can spot word sequences that don't fit well with typical patterns in the language. For example:

- If an OCR engine outputs a string like "jump3d over", the bigram ("jump3d", "over") might be flagged as an error since it's unlikely in normal language use. A better prediction would be ("jumped", "over").
- You can then use this information to correct OCR errors by suggesting better n-grams based on the probability of their occurrence in the language.

5. Reducing Ambiguity

A single word or token can have multiple meanings or interpretations, but an n-gram can resolve ambiguity by considering the surrounding words:

- **Tokens**: "bank" (could mean a financial institution or the side of a river)
- **Bigrams**: ("side", "bank") → Refers to the side of a river, not a financial institution.

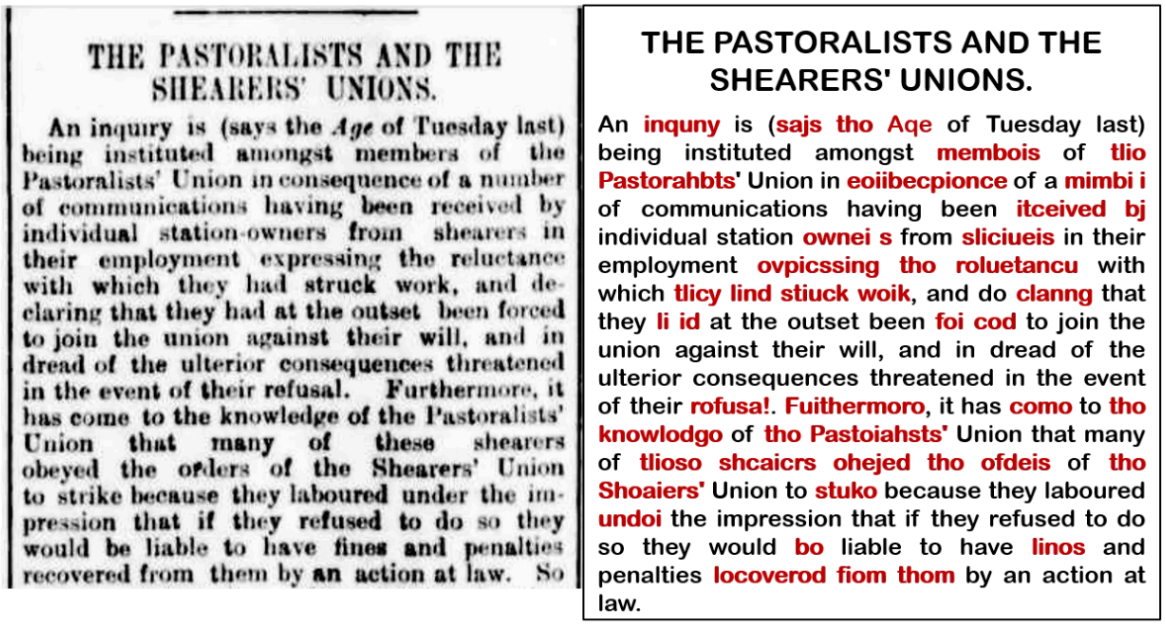
Thus, by using n-grams, you improve your ability to disambiguate words based on context, making the analysis more robust.

6. Handling Misspellings or OCR Mistakes

- N-grams allow you to identify incorrect words in the context of the whole sequence, not just in isolation. For example:
    - If "foox" appears instead of "fox", a bigram like ("quick", "foox") would seem unusual, and you could infer that "foox" is likely a misspelling or OCR error.

This helps to correct individual word errors by considering the word's surrounding context rather than focusing solely on isolated words.

-Some of the original texts in Trove are difficult to read even with the human eye, so it is no surprise that machines have struggled! The example below shows a scanned article next to the OCR-derived text, with the OCR errors shown in red.



THE PASTORALISTS AND THE SHEARERS' UNIONS.

An inquiry is (says the *Age* of Tuesday last) being instituted amongst members of the Pastoralists' Union in consequence of a number of communications having been received by individual station-owners from shearers in their employment expressing the reluctance with which they had struck work, and declaring that they had at the outset been forced to join the union against their will, and in dread of the ulterior consequences threatened in the event of their refusal. Furthermore, it has come to the knowledge of the Pastoralists' Union that many of these shearers obeyed the orders of the Shearers' Union to strike because they laboured under the impression that if they refused to do so they would be liable to have fines and penalties recovered from them by an action at law. So

THE PASTORALISTS AND THE SHEARERS' UNIONS.

An inquny is (sajs tho Aqe of Tuesday last) being instituted amongst membois of tlio Pastorahbts' Union in eoiibecpionce of a mimbi i of communications having been itceived bj individual station ownei s from sliciueis in their employment ovpicssing tho roluetancu with which tlicy lind stiuck woik, and do clanng that they li id at the outset been foi cod to join the union against their will, and in dread of the ulterior consequences threatened in the event of their rofusa!. Fuithermoro, it has como to tho knowlodgo of tho Pastoiahsts' Union that many of tlioso shcaicrs ohejed tho ofdeis of tho Shoaiers' Union to stuko because they laboured undoi the impression that if they refused to do so they would bo liable to have linos and penalties locoverod fiom thom by an action at law.

Book : A Novel Machine Learning Based Approach for Post-OCR Error Detection

- Post processing is the most conventional approach for correcting errors that are caused by Optical Character Recognition (OCR) systems. Traditionally, the task is divided into two subtasks: (1) Error detection to classify words1 as either erroneous or valid, and (2) Error correction to find suitable candidates to correct the erroneous words (A large body of work has proven the success of statistical and supervised machine learning methods for both subtasks
- Machine learning methods largely rely on feature engineering for their performances, particularly in supervised settings. Feature engineering involves exploring various features and feature combinations that best characterize the data. However, for post-OCR error detection, finding a

suitable set of features is challenging because of the diversity of OCR errors

- To address this challenge we propose a novel approach to the error detection task. Instead of examining more features we focus merely on a single feature, namely the n-gram counts of the candidate token. Our approach is inspired by dictionary lookup approaches, which are known for their simplicity and efficiency but are restricted to the dictionary size. Since building large-scale dictionaries is a challenging task in itself, we propose to generate n-grams of a given candidate token, and then use their counts as the only feature to train machine learning models

- Overview - ICDAR2019 Robust Reading Challenge on Arbitrary-Shaped Text

  This is a challenge of scene text understanding, which can be broken down into scene text detection, recognition, and spotting problems. The main novelty of this competition resides in the nature of the competition's dataset - the ArT dataset. Specifically, almost a quarter of the text instances in the dataset are arbitrary-shaped

Figure 1. *Example images of the ArT dataset. Red color binding lines are formed with polygon ground truth format.*

- This approach is very simple and is computationally less expensive as it does not require any other feature computation apart from the n-gram counts.

- Taking this approach, dictionaries and large word lists are usually compiled from corpora and other sources. Each token in the data is then compared with the word in the dictionary to determine whether it is an error word or not. This approach has been explored by the CSIITJ team who was among the six successful teams participating in the ICDAR 2019 competition . Dictionary lookup methods for post-OCR are challenging because they usually suffer from out-of-vocabulary problems. Another limitation is in

detecting real-word errors, i.e. the word appears in the dictionary but is wrong in its context. Therefore alternative methods to OCR error detection have been proposed

- The remaining teams in ICDAR 2019 applied techniques from:

  - Context-based character correction using BERT (CCC)

  - Character level attention approach using the open source system OpenNMT (CLAM)

  - Weighted finite-state transducers based on noisy channel model (REA1&2)

  -Character level seq2seq multi-layer LSTM (UVA)

- Other approaches to post-OCR error detection combined character : word-n-grams and context based features to train a machine learning model
- Khirbat in 2017 trained a support vector machine (SVM) model with 3 features: presence of non alphanumeric characters, bi-gram frequency of the word and context information that is if the word appears with its context in other places. s. Nguyen et al. (2019) trained a Gradient Tree Boosting classifier on a set of 13 character and word features on two datasets of English historical handwritten documents (monograph and periodical) taken from the ICDAR competition (Chiron et al., 2017). The features they experimented with include character and word n-gram frequencies, part-of-speech, and the frequency of the OCR token. Dannells and Persson ´ (2020) trained an SVM model on 6 statistical and word based features including the number of non alphanumeric characters, number of vowels, word length, tri-gram character frequencies, number of uppercase characters and the amount of numbers occurring in the word.
- The choice of the features is essential for the performance of the machine learning model. The advantage of these methods is that they are trained to detect both real-word and non_word errors. The drawback is they require laborious feature engineering , Laborious feature engineering is a bottleneck not only for machine learning but also for other statistical approaches that rely on pre-defined features extracted from data, such as noisy channel approaches

- **Method**

  - Datasets and Preprocessing We used the datasets from the ICDAR2019 competition on post OCR error detection and correction.3 The total size of the original data is 22 million characters and it contains varying numbers of characters for ten European languages (Bulgarian, Czech, German, English, Spanish, Finnish, French, Dutch, Polish, Slovenian) together with the corresponding ground truth data. The dataset comes with the OCRed and ground truth aligned at the character level. For our experiments, we needed to align it at the token (word) level. We did that by tokenizing the ground truth at space and for each token taking the same number of characters from the OCRed version. After we removed the special alignment symbols ('@' and '#') inserted by organizers for alignment. The resulting OCRed and ground truth tokens were compared to set the labels '0' if the token was erroneous or '1' if the token was

valid. These labels are the dependent variables that are to be learned and predicted by the machine learning models

English			Danish			Finnish		
Word	GT	Label	Word	GT	Label	Word	GT	Label
matter	matter	1	Bezirke	Bezirke	1	jolloin	jolloin	1
the	the	1	.Fiili	@F@li	0	lainasimat	lainasiwat	0
king@	king	0	welche	welche	1	saimat	saiwat	0
very	very	1	niedergese@ht	niedergesetzt	0	takaisin	takaisin	1
glad	glad	1	Bericht	Bericht	1	jtt	jtt	1
hereof,@	hereof,	0	Wesentlichen	Wesentlichen	1	kaupungissa	kaupungissa	1
@Hkewise	likewise	0	hkle	hatte	0	Maan	Waan	0

Table 1: A sample from the English, Danish, and Finnish datasets after the preprocessing step (GT = Ground Truth).

Language	Training Set				Test Set			
	0		1		0		1	
	#words	%	#words	%	#words	%	#words	%
Bulgarian (BG)	17844	0.37	29635	0.63	9750	0.40	14404	0.60
Czech (CZ)	7227	0.19	31230	0.81	3335	0.27	9081	0.73
Danish (DE)	18216	0.29	45325	0.71	3573	0.25	10784	0.75
English (EN)	9844	0.33	20559	0.67	3802	0.33	7609	0.67
Spanish (ES)	33164	0.51	31698	0.49	8996	0.59	6287	0.41
Finnish (FI)	48644	0.22	165940	0.78	11996	0.23	39781	0.77
French (FR)	85678	0.20	343858	0.80	21855	0.20	85535	0.80
Dutch (NL)	45593	0.51	47875	0.49	15619	0.47	17712	0.53
Polish (PL)	21436	0.70	8912	0.30	6730	0.57	5008	0.43
Slovenian (SL)	6098	0.16	31865	0.84	5128	0.31	11501	0.69

Table 2: Training and test dataset statistics

- Machine learning classifiers are known to have pros and cons depending on the task at hand. Dannells ´ and Virk (2020) compared 5 state-of-the art machine learning classifiers including Logistic Regression, Decision Tree, Bernoulli Naive Bayes, Naive Bayes and SVM. They found that SVM is the best choice for post-OCR detection. Others have also shown the performance of SVM is equivalent to the performance of artificial neural networks (Arora et al., 2010; Hamid and Sjarif, 2017; Amrhein and Clematide, 2018). In response to these previous experiments, in this study we have chosen to experiment with SVM models

- Because the model requires the data to be in numeric form we used one-hot encoding for data transformations . The major idea behind one-hot encoding is to add an extra dimension in the feature vector for each unique feature value. This produces an N dimensional feature vector (the learned encoding), where N is the total number of unique values of all features. In our case we have words as the training data. Suppose [w1,w2,w3.....wn] is the set of unique words, and [0,1,1.....0] is the set of corresponding labels representing whether the word is erroneous or not. The learned one-hot encoding will be a (n+2) dimensional vector, where n is the unique number of words in the training data and there are 2 unique label values. Each word is then encoded by setting the corresponding word and label dimensions of the vector to 1 while the remaining dimensions are set to 0.

Training Data:

plaintext Copier le code

```
Vocabulary: [w1, w2, w3]
Labels: 0 (correct), 1 (erroneous)
```

Encoding:

- $n = 3$ (three unique words: w_1, w_2, w_3).
- $+2$ (two additional dimensions for labels).

Representation:

Each word-label pair is represented as an $(n + 2)$-dimensional vector:

- Word w_1, Label `0`:

  plaintext Copier le code

  ```
  [1, 0, 0, 1, 0]
  ```

  Explanation:

  - The first three positions correspond to the word encoding: `[1, 0, 0]` (indicating w_1).
  - The last two positions correspond to the label encoding: `[1, 0]` (indicating "correct").
- Word w_2, Label `1`:

  plaintext Copier le code

  ```
  [0, 1, 0, 0, 1]
  ```

- **Generating the Machine Learning Features :**

-Instead of building a dictionary separately from different external resources, we let our SVM model build it from the training data. This was achieved by learning one-hot encoding from the training data (i.e. words), encoding the training data and then using the resulting vectors as the only feature to train the SVM model.
-This type of approach has a major restriction that it is not scalable and is bound to feature values seen in the training data. In our case this means if a word has not been seen in the training data the system will simply fail to predict whether it is erroneous or not. Another downfall is that depending on the size of training data it may take days to train such models. To overcome these limitations we experimented further with the n-gram approach . Instead of using the complete word, for each candidate token, we generated character uni-grams, bi-grams, and tri-grams from it. These n-grams together with their counts within the token were used as feature values to train and test the model. To take an example, suppose our candidate word is 'passenger'. The computed uni-, bi-, and tri-gram counts will be as follows

- uni-gram {'a':1, 'e':2, 'g':1, 'n':1, 'p':1, 'r':1, 's':2}
- bi-gram {' p':1, 'as':1, 'en':1, 'er':1, 'ge':1, 'ng':1, 'pa':1, 'r ':1, 'se':1, 'ss':1}
- tri-gram {' pa':1, 'ass':1, 'eng':1, 'er ':1, 'ger':1, 'nge':1, 'pas':1, 'sen':1, 'sse':1}

-It is worth mentioning that the scope of n-gram counts is limited to the word itself, rather than the entire training data i.e. these counts represent the occurrence of a particular uni-, bi-, or tri-gram within the word rather than the total count of the n-gram in the training data.
-The intuition for using n-grams instead of complete word to overcome the above mentioned limitation is simple: For a given word, it is more probable that the uni-, bi-, and tri-grams generated from the word have been seen in the training data as opposed to the complete word, This can remove the previously mentioned limitation of using the words as feature based approach and make the system more scalable and computationally less expensive

	Word			Unigram						Bigram			Trigram		
							class_weight=balanced								
	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
BG	0.68	0.93	**0.79**	0.83	0.80	**0.82**	0.90	0.69	**0.79**	0.89	0.84	**0.86**	0.86	0.84	**0.85**
CZ	0.93	0.12	0.21	0.95	0.57	0.70	0.83	0.73	**0.78**	0.96	0.63	**0.74**	0.96	0.51	0.67
DE	0.95	0.14	0.25	1.0	0.55	0.71	1.0	0.54	0.70	0.93	0.72	0.81	0.81	0.76	0.78
EN	0.63	0.91	**0.75**	0.89	0.71	**0.79**	0.96	0.59	**0.73**	0.89	0.78	**0.83**	0.84	0.77	**0.81**
ES	0.83	0.90	**0.86**	0.88	0.88	**0.88**	0.99	0.61	**0.75**	0.90	0.89	**0.90**	0.87	0.91	**0.89**
FI	x	x	x	0.79	0.64	0.71	0.68	0.77	0.72	0.90	0.77	0.83	0.90	0.77	0.83
FR	0.94	0.23	0.36	0.86	0.75	**0.80**	1.0	0.63	**0.77**	0.82	0.82	**0.82**	0.81	0.84	**0.83**
NL	0.80	0.95	**0.87**	0.71	1.00	**0.83**	0.98	0.45	0.62	0.71	1.00	**0.83**	0.72	0.99	**0.84**
PL	0.75	0.97	**0.85**	0.70	1.00	**0.83**	0.98	0.57	0.72	0.73	0.93	**0.84**	0.73	0.98	**0.83**
SL	0.92	0.14	0.24	1.00	0.42	0.59	0.98	0.60	**0.74**	0.97	0.44	0.60	0.93	0.32	0.48

Table 4: Evaluation results of post-error detection using the proposed methodology.

- Training supervised machine learning models with a large number of features is a computationally expensive task. This has been demonstrated in previous work where handcrafted features were considered at the expense of high computational costs. In this study we have taken a different approach and have proposed to use n-gram counts as the only feature to train SVM models. N-gram counts have previously been used for post-OCR detection, but not in the sense that we have proposed in this study. Instead of computing the n-gram counts over the entire training data we have proposed to compute them within a given token and use them as the only feature to train and test our models.
- The proposed approach is interesting because it eliminates the need for feature engineering; a task which is laborious and computationally expensive. The results show simple n-gram counts, which are fairly easy to compute, are enough for the task at hand. The approach is also gainful because it does not require large amounts of data. Given the relatively small datasets we experimented with we were able to show our method is performing better for the majority of languages compared to deep learning systems such as the ones explored by the CCC and UVA teams

- Digital libraries, like the Internet Archive, offer a vast collection of historical and culturally important books in image formats, including works written in low-resource and endangered languages. However, their image-only format limits content accessibility, hindering the use of these essential resources. Therefore, Optical Character Recognition (OCR) technologies are evidently useful in this context. However, OCR outputs frequently contain errors, particularly when working with texts featuring complex styles, archaic fonts, or unconventional layouts. These errors may include character recognition mistakes, formatting issues, and hyphenation problems, which are particularly prominent when dealing with low-resource languages Ignat et al. (2022). Poor quality OCR can reduce the usefulness of these digital texts, adversely affecting downstream tasks

Book : CharBERT: Character-aware Pre-trained Language Model

Most pre-trained language models (PLMs) construct word representations at subword level with Byte-Pair Encoding (BPE) or its variations, by which OOV (out-of-vocab) words are almost avoidable. However, those methods split a word into subword units and make the representation incomplete and fragile.

CharBERT is a model architecture designed to enhance traditional text embeddings by incorporating character-level information into word-level embeddings, improving performance on various natural language processing (NLP) tasks. It extends the **BERT (Bidirectional Encoder Representations from Transformers)** model by integrating character-level features alongside token embeddings, making it particularly effective for tasks involving morphologically rich languages or noisy text (e.g., typos, social media text).

Key Features of CharBERT

1. **Character-Level Embeddings**:

    - CharBERT adds a character-level encoder to capture fine-grained subword information, improving the handling of misspellings, rare words, and unseen vocabulary.
    - This character-level encoder often uses a convolutional neural network (CNN) or recurrent neural network (RNN) to process sequences of characters.
2. **Fusion of Word and Character Representations**:
    - CharBERT fuses character-level and word-level embeddings into a unified representation.
    - This fusion ensures the model retains the high-level semantic context from BERT while benefiting from the detailed morphological information of characters.
3. **Transformer Backbone**:
    - The core of CharBERT is a standard BERT-like Transformer architecture.
    - The model integrates character embeddings at the input or intermediate levels to enhance the word embeddings used by the Transformer layers.

Architecture of CharBERT

1. **Character Encoder**:
    - Converts characters within a word into embeddings.
    - Uses CNNs (for local feature extraction) or LSTMs (for sequential character information).
2. **Word Embedding Fusion**:
    - Combines character-level representations with pre-trained word embeddings (e.g., BERT embeddings).
    - Techniques include concatenation, gating mechanisms, or attention layers.
3. **Transformer Layers**:
    - Applies BERT's multi-head self-attention and feed-forward layers on the fused embeddings.
4. **Output Layers**:
    - Provides outputs for downstream tasks such as text classification, named entity recognition (NER), question answering, or text generation.

Benefits of CharBERT

1. **Robustness to Noisy Text**:
    - Handles typographical errors, abbreviations, and rare words better than standard BERT.
2. **Improved Generalization**:
    - Incorporates morphological details, making it effective for languages with complex word structures.
3. **Seamless Integration**:
    - Builds upon pre-trained BERT models, leveraging their semantic understanding while adding morphological insights.

Applications

1. **Text Classification**: Improved accuracy in classifying text with noisy or domain-specific language.
2. **Named Entity Recognition (NER)**: Better recognition of rare or misspelled named entities.
3. **Question Answering**: Enhanced understanding of out-of-vocabulary terms in context.
4. **Social Media Analysis**: Effective in handling informal language and non-standard spelling.

- Unsupervised pre-trained language models like BERT (Devlin et al., 2019) and RoBERTa (Liu et al., 2019) have achieved surprising results on multiple NLP benchmarks. These models are pre-trained over large-scale open-domain corpora to obtain general language representations and then fine-tuned for specific downstream tasks. To deal with the large vocabulary, these models use Byte-Pair Encoding (BPE) (Sennrich et al., 2016) or its variations as the encoding method. Instead of whole words, BPE performs statistical analysis of the training corpus and split the words into subword units, a hybrid between character- and word-level representation. Even though BPE can encode almost all the words in the vocabulary into WordPiece tokens without OOV words, it has two problems: 1) incomplete modeling: the subword representations may not incorporate the fine-grained character information and the representation of the whole word; 2) fragile representation: minor typos can drastically change the BPE tokens, leading to inaccurate or incomplete representations. This lack of robustness severely hinders its applicability in real-world applications. We illustrate the two problems by the example in Figure 1. For a word like backhand, we can decompose its representation at different levels by a tree with a depth of 3: the complete

word at the first layer, the subwords at the second layer, and the last characters. BPE only considers representations of subwords on the second layer and misses the potentially useful information at the first and last layer. Furthermore, if there is noise or typo in the characters (e.g., missing the letter ‘k’), the subwords and its number at the second layer will be changed at the same time. Models relying purely on these subword representations thus suffer from this lack of robustness

- We propose a new pre-training method CharBERT (BERT can also be replaced by other pre-trained models like RoBERTa) to solve these problems. Instead of the traditional CNN layer for modeling the character information, we use the context string embedding (Akbik et al., 2018) to model the word’s fine-grained representation. We use a dual-channel architecture for characters and original subwords and fuse them after each transformer block. Furthermore, we propose an unsupervised character learning task, which injects noises into characters and trains the model to denoise and restores the original word. The main advantages of our methods are: 1) character-aware: we construct word representations from characters based on the original subwords, which greatly complements the subword-based modeling. 2) robustness: we improve not only the performance but also the robustness of the pre-trained model; 3) model-agnostic: our method is agnostic to the backbone PLM like BERT and RoBERTa, so that we can adapt it to any transformer-based PLM. In summary, our contributions in this paper are: • We propose a character-aware pre-training method CharBERT, which can enrich the word representation in PLMs by incorporating features at different levels of a word • We evaluate our method on 8 benchmarks, and the results show that our method can significantly improve the performance compared to the strong BERT and RoBERTa baselines; • We construct three character attack test sets on three types of tasks. The experimental results indicate that our method can improve the robustness by a large margin

- **Model Architecture :**

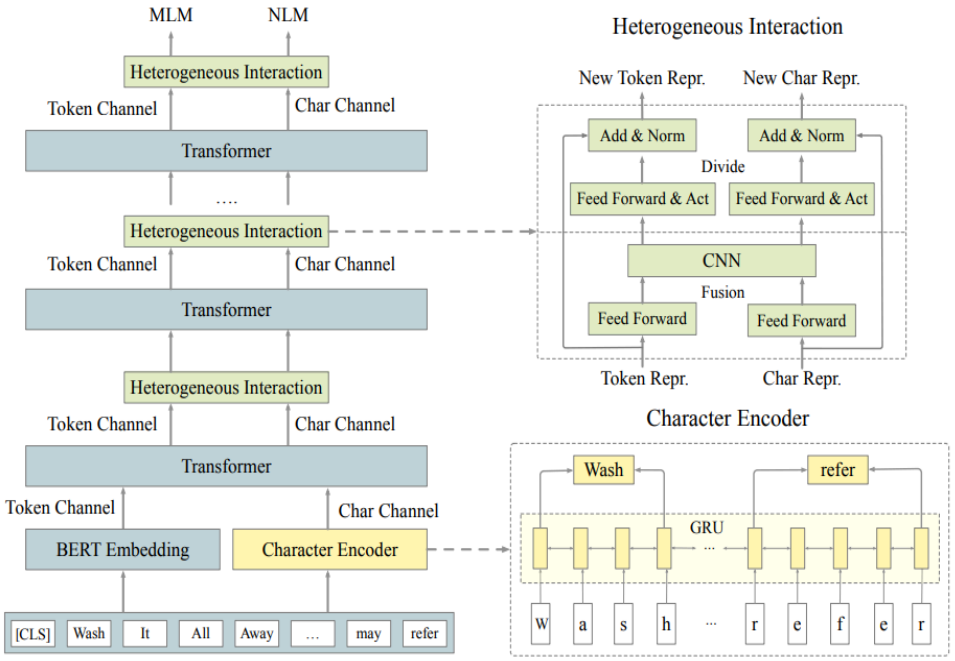


Figure 2: The neural architecture of CharBERT. The left part is the main structure of CharBERT based on the original pre-trained models like BERT. The modules in the right part are the heart of CharBERT: character encoder and heterogeneous interaction. *(best viewed in color)*

-Character Encoder :

we need to form token-level embeddings with the input sentences as sequences of characters. We first convert the sequences of tokens into characters and embed them into fixed-size vectors then apply a bidirectional GRU layer (Cho et al., 2014) to construct the contextual character embeddings, which can be formulated by

$$e_j^i = W_c \cdot c_j^i \;\; ; \;\; h_j^i(x) = \text{Bi-GRU}(e_j^i);$$

-where Wc is the character embedding matrix , h i j is the representation for jth character in the ith token. We apply the bi-GRU on the characters with a length of N for the whole input sequence instead of a single token, building the representations from the characters within and among the subwords. To construct token-level embeddings, we concatenate the hidden of the first and last character of the token.

$$h_i(x) = [h_1^i(x); h_{n_i}^i(x)]$$

where ni is the length of ith token and hi(x) is the token-level embedding from characters. The contextual character embeddings are derived by characters and can also catch the full word information by bi-GRU layers.

- **GRU :**

The GRU (Gated Recurrent Unit) is a type of recurrent neural network (RNN) architecture used in natural language processing (NLP) tasks. In the context of CharBERT, GRUs are particularly employed to process character-level embeddings and learn sequential dependencies within textual data at the character level.

. Role of the GRU in CharBERT

- **Character-level Embeddings**:
  - For each word (or subword), CharBERT generates embeddings from its characters.
  - These character embeddings are passed through a GRU layer (or sometimes a BiGRU, the bidirectional version of GRU) to capture sequential dependencies among characters.
- **Why GRU?**:
  - GRUs are computationally efficient compared to other RNNs like LSTMs while still capturing long-term dependencies.

    - GRUs are well-suited for processing sequences where memory of previous steps is crucial, as in understanding how characters contribute to forming a word or subword.

3. Output of the GRU:

- The GRU outputs a **character-level representation** of each word, which encodes rich information about its spelling, morphology, and potentially typographical errors or rare spellings.
- These representations are then combined with BERT's word-level embeddings to enhance its performance on downstream tasks such as sentiment analysis, named entity recognition, or other NLP tasks.

4. Advantages of Using GRU in CharBERT:

- **Better handling of noisy or rare data**: Character-level information can help in cases where words are rare or contain typos.
- **Morphological understanding**: GRUs allow the model to capture patterns in how prefixes, suffixes, and stems combine in words.
- **Efficiency**: Compared to LSTMs, GRUs reduce the computational load while maintaining comparable performance.

● Transformer

The core Transformer in CharBERT remains similar to BERT but operates on the enriched embeddings.

Key Components of Each Transformer Layer:

1. **Multi-Head Self-Attention**:

- Enables tokens to attend to other tokens in the sequence, capturing contextual relationships bidirectionally.
- In CharBERT, the character-level enhancements in embeddings provide additional granularity, allowing the self-attention mechanism to capture morphological and subword nuances.

2. **Feed-Forward Networks (FFN)**:
   - Processes the output of the self-attention mechanism.
   - The FFN helps refine token representations by applying two dense layers with a non-linear activation (e.g., ReLU).
3. **Layer Normalization and Residual Connections**:
   - Residual connections help preserve information across layers.
   - Layer normalization stabilizes the network, making training more robust.

• Heterogeneous Interaction

The embeddings from characters and the original token-channel are fed into the same transformer layers in pre-trained models. The token and char representations are fused and split by the heterogeneous interaction module after each transformer layer. In the fusion step, the two representations are transformed by different fully-connected layers. Then they are concatenated and fused by a CNN layer, which can be formulated by

$$t'_i(x) = W_1 * t_i(x) + b_1 \;\; ; \;\; h'_i(x) = W_2 * h_i(x) + b_2$$
$$w_i(x) = [t'_i(x); h'_i(x)] \;\; ; \;\; m_{j,t} = tanh(W_3^j * w_{t:t+s_j-1} + b_3^j)$$

where ti(x) is the token representations, W, b are parameters, wt:t+sj−1 refers to the concatenation of the embedding of (wt ,...,wt+sj−1), sj is the window size of jth filter, and m is the fusion representation with the dimension same with the number of filters. In the divide step, we transform the fusion representations by another fully connected layer with GELU activation layer (Hendrycks and Gimpel, 2016). We then use the residual connection to retain the respective information from the two channels.

The left part shows the overall structure of **CharBERT**, which is based on BERT and enhanced by incorporating a **character encoder** and **heterogeneous interaction layers**.

A. Token Channel

- This is the standard BERT processing pipeline.
- Tokens are embedded using BERT's **WordPiece embeddings**, along with segment embeddings and positional embeddings.
- The token embeddings are fed into a stack of **Transformer layers** for contextualization, just like in vanilla BERT.

B. Character Channel

- Each token (or subword) is further broken down into its individual **characters**.
- These characters are processed by the **Character Encoder** to generate a **character-level representation** for each token.
- The character representations are then fed into the **heterogeneous interaction** layers, where they interact with token-level features.

C. Heterogeneous Interaction

- After each Transformer layer, the token and character representations are updated through the **heterogeneous interaction mechanism**.
- This mechanism allows the model to fuse character-level and token-level information iteratively.
- The updated representations are passed back to the Transformer layers for further processing.

Final Outputs:

- The final token and character representations are used for downstream tasks like Masked Language Modeling (MLM) or Next Sentence Prediction (NSP), or fine-tuning for specific NLP tasks.

Final Outputs of the Token Channel

- The token channel goes through multiple layers of Transformers, where token representations are continuously refined by integrating context from surrounding tokens (via self-attention) and from character-level representations (via heterogeneous interaction).
- After the last Transformer layer, the token channel produces **contextualized token embeddings**:

    - These embeddings capture the meaning of each token (subword) in the context of the entire sentence.
    - They are the core output used for downstream NLP tasks.

---

2. Final Outputs of the Character Channel

- The character channel processes the token's character-level representations through GRU layers and heterogeneous interaction mechanisms.
- The character embeddings are updated at every layer and fused with token embeddings in the heterogeneous interaction modules.
- By the end, the character channel outputs **contextualized character representations**:
    - These embeddings capture subword-level morphological features and are particularly helpful for handling typos, rare words, or morphologically complex languages.

---

3. Combined Outputs: Task-Specific Representations

The final outputs depend on the specific task for which CharBERT is being used. The token and character representations can either be used individually or fused further for the task. Below are common use cases:

A. Masked Language Modeling (MLM)

- MLM is a pre training task where random tokens in the input sequence are masked (e.g., replaced with `[MASK]`), and the model predicts the original tokens.
- Final **token embeddings** from the last Transformer layer are passed through a softmax classifier to predict the masked tokens.
- Character-level features help make better predictions for rare or misspelled tokens.

B. Next Sentence Prediction (NSP)

- NSP is another pretraining task where the model predicts whether one sentence follows another in the corpus.
- The final embedding of the `[CLS]` token (a special token added to the input) is used as a **sentence-level representation**.

- This `[CLS]` embedding is passed through a classifier to predict whether the second sentence is the next sentence.

C. Fine-Tuning for Downstream Tasks

When CharBERT is fine-tuned for specific NLP tasks, the final outputs are adapted as follows:

1. **Sequence Classification (e.g., Sentiment Analysis)**
    - The final embedding of the `[CLS]` token is passed to a classifier.
    - The `[CLS]` token summarizes the information from both token and character channels, making it suitable for classification tasks.
2. **Named Entity Recognition (NER)**
    - The final embeddings of each token (from the token channel) are used to classify each token as belonging to an entity type (e.g., person, organization, location) or not.
    - Character-level information is particularly useful for identifying named entities with uncommon spellings.
3. **Question Answering (QA)**
    - For tasks like SQuAD, the final embeddings of all tokens are used to predict:
        - The **start position** and **end position** of the answer span within the input text.
    - Character-level features help improve predictions for text with typos or uncommon words.
4. **Part-of-Speech Tagging (POS)**
    - The final embeddings of each token are classified into their respective part-of-speech categories.
    - Morphological patterns captured by the character channel are particularly beneficial.

---

4. Fusion of Token and Character Representations

The heterogeneous interaction ensures that the final outputs of the token and character channels are **highly synergistic**:

- **Token embeddings** carry semantic and contextual information.
- **Character embeddings** carry subword-level morphological and spelling-related information.
- Together, they provide a rich representation for handling a wide range of NLP tasks.

Summary of Outputs

- **Token Outputs**: Contextualized token representations for all tokens in the sequence, derived from BERT-style processing.
- **Character Outputs**: Contextualized character-level representations, enriched by GRU and CNN processing.
- **Task-Specific Outputs**:
    - `[CLS]` embedding: Used for classification tasks (e.g., sentiment analysis, NSP).
    - Token embeddings: Used for token-level tasks (e.g., NER, POS tagging, QA).

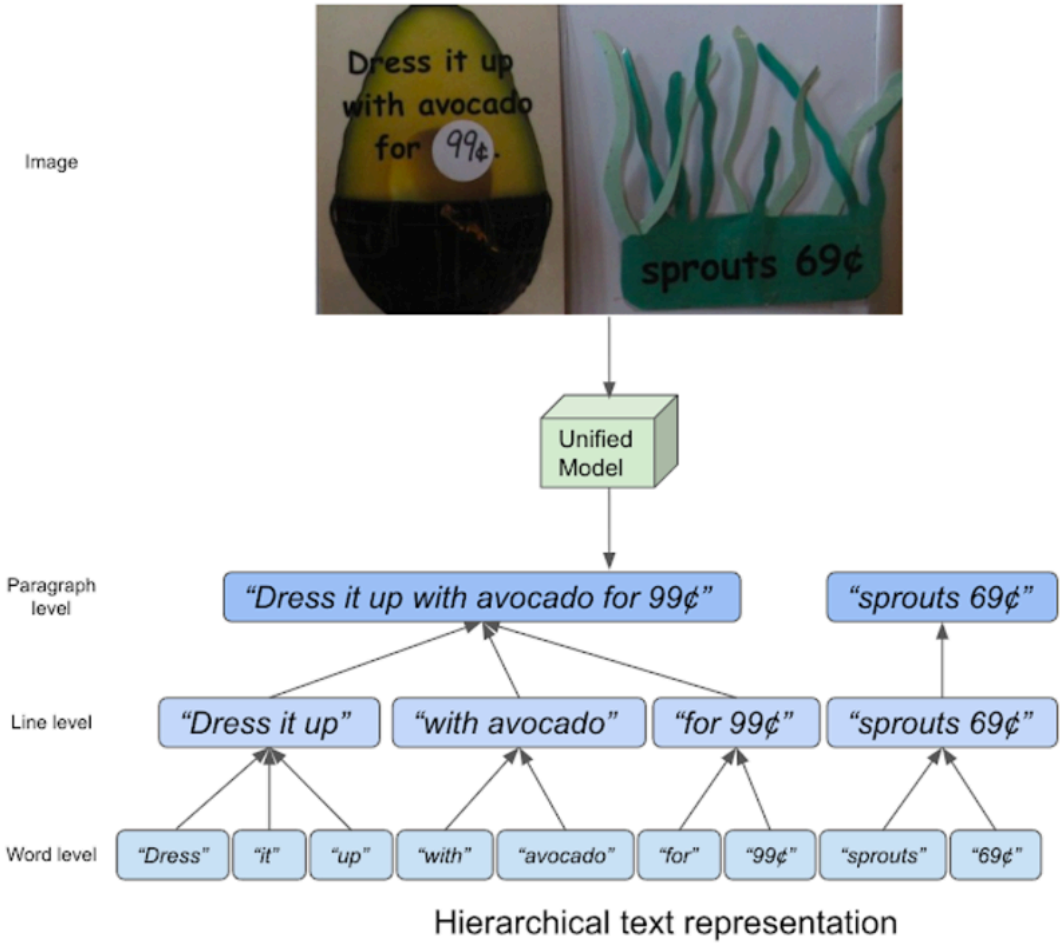
Role of the `[CLS]` Token in CharBERT

- The `[CLS]` token is inserted as the very first token in the input sequence during training or inference.
- Its primary purpose is to aggregate information about the entire input sequence, including all tokens and their relationships.
- As the sequence passes through the layers of the Transformer, the `[CLS]` token's embedding is continuously updated to encode contextualized information from the entire sequence.

[CLS] Sentence A [SEP] Sentence B [SEP]

`[CLS]`: Marks the start of the sequence.
`[SEP]`: Separates Sentence A from Sentence B.

Book : Enhancing OCR Performance through Post-OCR Models: Adopting Glyph Embedding for Improved Correction



The concept of hierarchical text representation.

- The last few decades have witnessed the rapid development of Optical Character Recognition (OCR) technology, which has evolved from an academic benchmark task used in early breakthroughs of deep learning research to tangible products available in consumer devices and to third party developers for daily use. These OCR products digitize and democratize the valuable information that is stored in paper or image-based sources (e.g., books, magazines, newspapers, forms, street signs, restaurant menus) so that they can be indexed, searched, translated, and further processed by state-of-the-art natural language processing techniques.

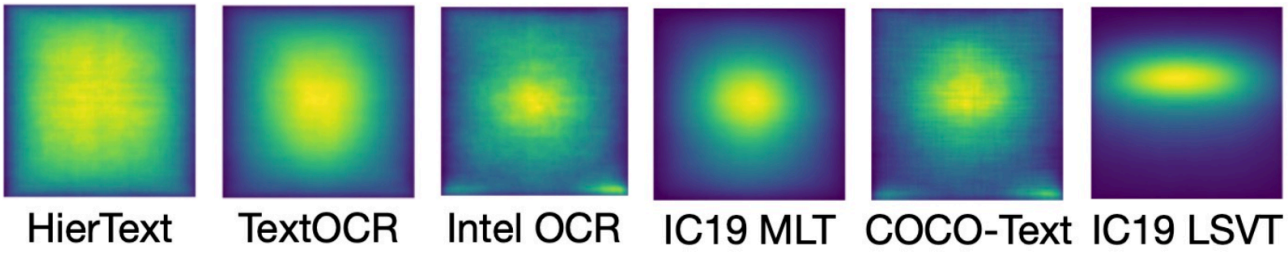
- Research in scene text detection and recognition (or scene text spotting) has been the major driver of this rapid development through adapting OCR to natural images that have more complex backgrounds than document images. These research efforts, however, focus on the detection and recognition of each individual word in images, without understanding how these words compose sentences and articles.
- Layout analysis is another relevant line of research that takes a document image and extracts its structure, i.e., title, paragraphs, headings, figures, tables and captions. These layout analysis efforts are parallel to OCR and have been largely developed as independent techniques that are typically evaluated only on document images. As such, the synergy between OCR and layout analysis remains largely under-explored. We believe that OCR and layout analysis are mutually complementary tasks that enable machine learning to interpret text in images and, when combined, could improve the accuracy and efficiency of both tasks.
- Google announce the *Competition on Hierarchical Text Detection and Recognition* (the HierText Challenge), hosted as part of the 17th annual International Conference on Document Analysis and Recognition (ICDAR 2023). The competition is hosted on the Robust Reading Competition website, and represents the first major effort to unify OCR and layout analysis. In this competition, we invite researchers from around the world to build systems that can produce hierarchical annotations of text in images using words clustered into lines and paragraphs
- In this competition, we use the HierText dataset that we published at CVPR 2022 with our paper "Towards End-to-End Unified Scene Text Detection and Layout Analysis". It's the first real-image dataset that provides hierarchical annotations of text, containing *word*, *line*, and *paragraph* level annotations. Here, "*words*" are defined as sequences of textual characters not interrupted by *spaces*. "*Lines*" are then interpreted as "*space*"-separated clusters of "*words*" that are logically connected in one direction, and aligned in spatial proximity. Finally, "*paragraphs*" are composed of "*lines*" that share the same semantic topic and are geometrically coherent.
- To build this dataset, we first annotated images from the Open Images dataset using the Google Cloud Platform (GCP) Text Detection API. We filtered through these annotated images, keeping only images rich in text content and layout structure. Then, we worked with our third-party partners to manually

correct all transcriptions and to label words, lines and paragraph composition. As a result, we obtained 11,639 transcribed images, split into three subsets: (1) a train set with 8,281 images, (2) a validation set with 1,724 images, and (3) a test set with 1,634 images. As detailed in the paper, we also checked the overlap between our dataset, TextOCR, and Intel OCR (both of which also extracted annotated images from Open Images), making sure that the test images in the HierText dataset were not also included in the TextOCR or Intel OCR training and validation splits and vice versa. Below, we visualize examples using the HierText dataset and demonstrate the concept of hierarchical text by shading each text entity with different colors

- HierText is currently the most dense publicly available OCR dataset. Below we summarize the characteristics of HierText in comparison with other OCR datasets. HierText identifies 103.8 words per image on average, which is more than 3x the density of TextOCR and 25x more dense than ICDAR-2015. This high density poses unique challenges for detection and recognition, and as a consequence HierText is used as one of the primary datasets for OCR research at Google.

Dataset	Training split	Validation split	Testing split	Words per image
ICDAR-2015	1,000	0	500	4.4
TextOCR	21,778	3,124	3,232	32.1
Intel OCR	19,1059	16,731	0	10.0
HierText	8,281	1,724	1,634	103.8

- We also find that text in the HierText dataset has a much more even spatial distribution than other OCR datasets, including TextOCR, Intel OCR, IC19 MLT, COCO-Text and IC19 LSVT. These previous datasets tend to have well-composed images, where text is placed in the middle of the images, and are thus easier to identify. On the contrary, text entities in HierText are broadly distributed across the images. It's proof that our images are from more diverse domains. This characteristic makes HierText uniquely challenging among public OCR datasets.



Spatial distribution of text instances in different datasets.

- **Glyph embedding :**

Glyph embedding is a concept in natural language processing (NLP) and computer vision that integrates the visual or graphical features of characters (glyphs) into machine learning models. This approach is especially relevant in languages where visual representation plays a crucial role, such as Chinese, Japanese, or Korean, where characters carry semantic or phonetic information visually.

What Are Glyphs?

Glyphs are the visual symbols or shapes representing a character in a specific typeface or script. In the context of text processing, glyph embeddings capture these visual structures, often incorporating them into models alongside traditional textual features.

Why Use Glyph Embeddings?

1. **Rich Semantic Information**: Languages like Chinese have ideographic characters where the structure of the glyph carries meaning.
2. **Handle Rare Characters**: Glyph embeddings can help deal with rare or unseen characters by using their visual similarity to known characters.
3. **Improved Multimodal Models**: They bridge the gap between vision and text models by incorporating visual information.

How Glyph Embeddings Are Created

1. **Feature Extraction from Glyphs**:
    - **Bitmap Representation**: Convert glyphs into pixelated bitmaps and use convolutional neural networks (CNNs) to extract visual features.
    - **Vector Representation**: Use scalable vector graphics (SVG) to represent glyphs, focusing on their geometric properties.
2. **Fusion with Textual Embeddings**:
    - Combine glyph features with textual embeddings (e.g., Word2Vec, GloVe, or Transformer embeddings).
    - Use concatenation, attention mechanisms, or specialized layers to integrate the information.

Applications of Glyph Embedding

1. **Language Modeling**: Improved understanding of logographic languages in NLP tasks like machine translation and sentiment analysis.
2. **Handwriting Recognition**: Extracting features from glyphs enhances the performance of models trained to recognize handwriting or printed text.
3. **OCR (Optical Character Recognition)**: Glyph embeddings improve recognition accuracy for languages with complex scripts.
4. **Cross-Language NLP**: Helps bridge languages that share script features, such as Kanji in Japanese and Chinese characters.

Examples of Models Using Glyph Embedding

- **Glyph-aware Transformers**: Extensions of Transformer architectures that incorporate glyph embeddings into their layers for improved understanding of visually complex scripts.
- **Hybrid CNN-LSTM Models**: Convolutional layers extract glyph features, and LSTM layers process sequential text information.

-Converting images into digital formats has become increasingly practical for preserving ancient documents (Avadesh and Goyal, 2018; Narang et al., 2020), understanding real-time road signs (Wu et al., 2005; Kim, 2020), multimodal information extraction (Liu et al., 2019; Sun et al., 2021) and evaluating text-guided image generation (Petsiuk et al., 2022; Zhang et al., 2023; Rodríguez et al., 2023), just to name a few. These digital representations can be further processed using language models to extract underlying features. A high-quality OCR model can benefit various domains. However, the performance of optical character recognition (OCR) is often limited due to factors such as image quality or inherent model limitations.
- Glyph embedding captures the visual characteristics of the characters, rather than solely focusing on semantic aspects like most conventional embeddings
-In the initial phase of our study, we employ the extensively researched and widely employed ICDAR 2013 dataset as our primary data source for OCR tasks. The results obtained from this phase will subsequently serve as inputs for the next step, where we evaluate the performance of our postOCR correction model. Additionally, to facilitate training of the glyph embedding in our postOCR correction model, we rely on the Chars74K dataset .

- ## Chars74k dataset :

Dataset Structure

The Chars74K dataset is organized into three primary components:

1. English Characters

- **Total Images**: Approximately 64,000.
- **Categories**: 62 classes (26 uppercase letters, 26 lowercase letters, and 10 digits).
- **Sources**:
    - **Handwritten**: Characters written by multiple individuals.
    - **Printed**: Characters rendered in a variety of computer fonts.
    - **Natural Scenes**: Photographs of characters captured in real-world scenarios (e.g., signboards, license plates).

2. Kannada Characters

- **Total Images**: Around 7,000.
- **Categories**: 657 classes representing the Kannada script.
- **Sources**:
    - Focuses on handwritten characters written by multiple contributors.

3. Tamil Characters

- A smaller subset focusing on Tamil script characters, also handwritten.

Challenges in the Dataset

1. **Variations in Appearance**:
    - Different fonts, colors, and sizes of characters.
    - Handwritten characters vary significantly based on the writer's style.
2. **Natural Scene Challenges**:
    - Characters in natural scenes are often distorted, rotated, or affected by lighting conditions, occlusion, and noise.

3. **Multilingual Complexity**:
    - Recognizing characters in different scripts like Kannada and Tamil requires specialized approaches.

Dataset Organization

The dataset is organized into folders, typically grouped by class (e.g., folders for 'A', 'B', '1', '2', etc.), and each folder contains images corresponding to that character.

File Details:

- **Format**: Images are provided in various formats, including `.jpg`.
- **Image Size**: The images vary in size and resolution.
- **Labels**: Each image is labeled with its corresponding character for supervised learning tasks.

Applications

The Chars74K dataset is commonly used for:

1. **Character Recognition**:
    - Developing models for OCR, particularly for recognizing handwritten and scene-text characters.
2. **Scene Text Recognition**:
    - Training models to detect and classify text in natural images, such as street signs or advertisements.
3. **Font and Style Analysis**:
    - Analyzing variations in character shapes across different fonts and writing styles.
4. **Multilingual OCR**:
    - Recognizing characters from Kannada and Tamil scripts.

Machine Learning Approaches

To use Chars74K for character recognition tasks, various deep learning and traditional approaches can be applied:

1. **Traditional Methods**:
    - Feature extraction (e.g., SIFT, HOG) followed by classification with SVMs or Random Forests.
2. **Deep Learning Models**:
    - **Convolutional Neural Networks (CNNs)**: Effective for image-based character classification.
    - **Recurrent Neural Networks (RNNs)**: Useful for sequence modeling, especially for handwritten data.
    - **End-to-End Scene Text Recognition Models**: Combine object detection and classification for natural scene data.

Limitations

1. **Imbalanced Classes**:
    - Some characters or styles might have fewer samples than others, making the dataset slightly unbalanced.
2. **Dataset Size**:
    - While large, it is smaller compared to modern large-scale datasets used for deep learning tasks, such as ImageNet or COCO.
3. **Scene Text Variety**:
    - Scene text samples are limited and may not cover all real-world complexities.

- The ICDAR 2013 dataset and the recently introduced ICDAR 2023 dataset serve as prominent benchmarks for evaluating Optical Character Recognition (OCR) models. The ICDAR 2013 dataset encompasses a diverse collection of document images, encompassing both printed and handwritten text captured under a variety of conditions, including different languages, font styles, sizes, and distortions. It serves as a robust benchmark for assessing the accuracy and robustness of OCR systems in a wide array of real world scenarios. In contrast, the ICDAR 2023 dataset expands upon the achievements of its predecessor by introducing additional challenges that push the boundaries of OCR technology. It incorporates more intricate document layouts, degraded text, low resolution images, and challenging background clutter, with the goal of evaluating OCR model performance in increasingly demanding scenarios
- Both datasets will undergo evaluations using three models: **EasyOCR2 , PaddleOCR3 , and TrOCR** (Transformer-based OCR) .The EasyOCR model will be utilized specifically for single word detection boxes, while the other two models will provide single line detection box functionality. Consequently, the outputs from EasyOCR will consist solely of single words, whereas PaddleOCR will generate output in the form of sentences.
- The Chars74K dataset is employed for training the glyph embedding. This dataset encompasses scene images of English and Kannada characters, although for this particular experiment, only English alphabets are utilized. In addition to the Chars74K dataset, we capture screenshots of Korean and Hebrew characters to serve as samples for the garbage class in open-set classification. For training the post-OCR correction model, the data consists of the outputs obtained from EasyOCR, PaddleOCR, and TrOCR in the initial phase

• SOTA(STATE OF THE ART MODELS) OCR models :

---

1. EasyOCR

EasyOCR is an open-source OCR library that combines efficiency and accuracy, widely used for extracting text from images. It relies on two key components:

1. **Text Detection**: Uses the **CRAFT algorithm (Character Region Awareness for Text detection)** by Baek et al., 2019.
2. **Text Recognition**: Based on the **CRNN architecture (Convolutional Recurrent Neural Network)** proposed by Shi et al., 2016.

1.1 Detection: CRAFT Algorithm

- CRAFT is a **character-level text detector**.
- It focuses on detecting individual characters instead of entire words or lines, which allows it to better handle irregularly shaped text.
- **Steps**:
    1. **Region Score Map**: Predicts the likelihood of text regions in the image.
    2. **Affinity Score Map**: Links detected characters to form words.
    3. **Text Region Grouping**: Groups characters into coherent text regions or words.

1.2 Recognition: CRNN

The **CRNN architecture** combines convolutional neural networks (CNNs) for feature extraction and recurrent neural networks (RNNs) for sequence modeling. Here's how it works:

1. **Feature Extraction**:
    - A **ResNet** or **VGG** backbone extracts robust visual features from the image.
    - This reduces the input to a sequence of feature vectors.
2. **Sequence Labeling**:
    - An **LSTM (Long Short-Term Memory)** processes the sequential data and models relationships between characters.
3. **Decoding**:

- Uses **CTC (Connectionist Temporal Classification)** loss for decoding the sequence into readable text without requiring explicit character segmentation.

---

2. PaddleOCR

PaddleOCR is an OCR framework built on the **PaddlePaddle deep learning platform**. It offers robust text detection and recognition, particularly for multilingual and scene text.

2.1 Detection: PP-OCRv3

- **PP-OCRv3** is the latest iteration of PaddleOCR's text detection model.
- It combines improvements in:
    1. **Backbone**: Uses an efficient lightweight backbone for better performance and lower computational cost.
    2. **Detection Head**: Implements a text detection algorithm similar to EAST (Efficient and Accurate Scene Text Detector) or DBNet, focusing on text regions.
    3. **Optimization**: Includes post-processing optimizations to refine detected bounding boxes.

2.2 Recognition

PaddleOCR's text recognition pipeline includes:

1. **Lightweight CRNN**:
    - Extracts features using a convolutional backbone (e.g., MobileNet) and processes them using a recurrent network (GRU or LSTM).
    - Uses **CTC loss** for sequence-to-sequence prediction.
2. **Transformer-Based Models** (Advanced Option):
    - Offers transformer-based models for recognition tasks requiring high accuracy and multilingual capabilities.

2.3 Features

- **Multilingual Support**: Supports over 80 languages.
- **Scene Text Recognition**: Handles curved, rotated, or noisy text with ease.
- **End-to-End OCR**: Integrates detection and recognition in a streamlined workflow.

---

3. TrOCR

TrOCR (Transformer-based OCR) is a cutting-edge OCR model developed by Microsoft. It is entirely based on transformers, following the success of models like BERT and GPT.

3.1 Transformer Architecture

1. **Encoder**:
    - Extracts visual features from the input image using a **Vision Transformer (ViT)** or CNN-based backbone.
    - Encodes the image into a sequence of embeddings.
2. **Decoder**:
    - A **transformer decoder** predicts text in an auto-regressive fashion, similar to how language models like GPT generate text.
    - Each output token is conditioned on the previous ones, enabling TrOCR to capture contextual relationships between characters.

3.2 Key Advantages

- **Contextual Understanding**:
    - Unlike traditional CNN-based OCR models, transformers excel at modeling global context, which is useful for recognizing text in noisy or complex environments.
- **Pretrained Language Models**:
    - TrOCR can be fine-tuned using pre trained language models, enhancing its ability to understand linguistic patterns and semantics.
- **End-to-End Training**:
    - Combines image-to-text prediction in a unified framework, simplifying the training process.

3.3 Applications

- **Scene Text Recognition**: Recognizes text in real-world images (e.g., street signs, billboards).
- **Handwritten Text Recognition**: Performs exceptionally well on handwritten datasets, such as IAM or CVL.

What is a Glyph?

- **Glyphs** are the visual representations of characters. In OCR, a glyph corresponds to the shape or image of a letter, number, or symbol in a document. OCR systems typically extract these glyphs from images of text.

Glyph Embedding

- **Glyph Embedding** involves converting these visual representations of characters (glyphs) into numerical vectors (embeddings) in a continuous vector space. The idea is to represent the shapes of characters in a way that preserves their semantic and visual properties. These embeddings can be used to refine the OCR system's predictions, making post-processing corrections more effective.

- Post-OCR Correction by Language Models Post-OCR correction using Language Models (LMs) has emerged as a robust methodology for enhancing the accuracy and quality of Optical Character Recognition (OCR) outputs. LMs, such as transformer-based architectures like BERT (Devlin et al., 2018) or GPT (Brown et al., 2020), have revolutionized the field of natural language processing. Within the realm of post-OCR correction, LMs leverage their contextual understanding and language modeling capabilities to detect and rectify errors present in the recognized text. By analyzing the neighboring words, sentence structure, and semantic context, LMs can effectively identify and rectify OCR mistakes, encompassing misspellings, punctuation errors, and grammatical inconsistencies. This approach has demonstrated its efficacy in significantly enhancing OCR accuracy, particularly in scenarios where OCR models face challenges associated with complex layouts, degraded text, or intricate font styles. The post-OCR correction model is composed of two parallel encoders and one decoder. The encoders receive inputs in the form of sentences that have been embedded using CharBERT and the glyph embedding that we developed through training. CharBERT and Glyph Embedding CharBERT is a pre-trained language model that takes a string as input and produces its corresponding embedding. The model combines BERT embedding with character-level embedding. It undergoes two pre-training tasks: masked word prediction4 and noise reduction 5 . We chose this embedding for our postOCR correction model because the

denoising pretraining task is expected to aid in the correction process. Similar to BERT's pre-training task, CharBERT randomly masks words for prediction. 5CharBERT introduces random noise, such as random characters, into the corpus and trains the model to denoise the text.

- On the other hand, the glyph embedding is trained using open-set classification with a garbage class6 and adapted softmax. ResNet18 and ResNet50 models are utilized, with the last fully connected layer (fc) replaced and an additional fc layer appended for the purpose of fine-tuning. During inference, this additional fc layer is removed to generate an embedding of size 768 (Table 1). In fine-tuning, we experiment with unfreezing the last two fc layers as well as the last four layers. Unlike charBERT, which captures the semantic information of words, glyph embedding represents the visual glyph of characters. After fine-tuning, images of the same character are passed through the model to generate embeddings. The averaged embedding is considered as the glyph embedding for that specific character. Post-OCR Correction Model Once the glyph embedding training is complete, it becomes one of the inputs for the post-OCR correction model. This model is composed of two CNN encoders and one transformer decoder (Figure 1). CNN is a commonly employed architecture for processing character embeddings (Kim et al., 2016), while the transformer decoder offers a significant advantage in handling long-distance dependencies (Vaswani et al., 2017). Furthermore, the positional encoding within the transformer decoder aids in preserving the spatial information of the input sentence.

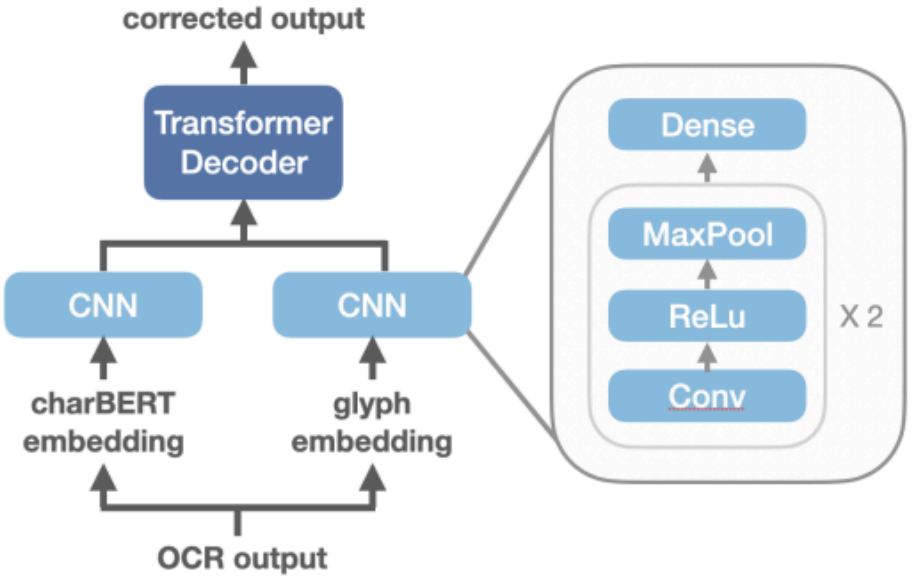


Figure 1: The post-OCR correction model consists of two parallel CNN encoders and a transformer decoder with CharBERT and glyph embedding as inputs.

Evaluation Criteria To assess the recognition performance of the OCR model, we employ a word level evaluation using full matching. In addition, for line-level evaluation, two metrics are utilized: Character Error Rate (CER) and Word Error Rate (WER). These metrics can be calculated using the equations 1, where S represents the number of substitutions, D represents the number of deletions, I represents the number of insertions, and C represents the total number of characters in the reference text. Similarly, WER can be calculated using equation 2, where N represents the total number of correct words. A lower value indicates better performance.

$$CER = \frac{S + D + I}{S + D + I + C}$$

$$WER = \frac{S_w + D_w + I_W}{N_w}$$

-Evaluation on Glyph Embedding Models Table 4 illustrates the performance of various models used for training glyph embedding. Unfreezing more than the two modified fc layers can lead to disturbances in the pre-trained ResNet model, resulting in decreased accuracy. Furthermore, in our specific case, the garbage class approach yields superior results compared to the adapted softmax approach.

Model	**Acc (%)**
RN18, garbage class, unfreeze L2	85.08
RN50, garbage class, unfreeze L2	**87.37**
RN50, garbage class, unfreeze L4	83.90
RN50, adapted softmax, unfreeze L4	83.19

Table 4: This table shows models employed for training glyph embedding and provides a comprehensive comparison of their performance. (RN refers to ResNet; L refers to number of layers)

- Evaluation on OCR Models with GPT Table 2 presents the performance of various OCR models on the word-level correctness of the ICDAR2013 dataset (about more than 800 images), considering both the original ground truth and the fixed ground truth. Our analysis focuses on EasyOCR, a slightly inferior OCR model, as well as two superior OCR models, PaddleOCR and TrOCR. Utilizing gpt-3.5- turbo-0301 for post-OCR correction demonstrates a noticeable improvement in the OCR results, indicating that the language model can enhance and compensate for the deficiencies to some extent. In Table 3, We demonstrate the post-OCR correction process based on ICDAR 2023 at line-level (we have selected about 50 images with hierarchical text). Similarly, the language model proves beneficial in rectifying errors by leveraging contextual information.

Model	WER	CER
EasyOCR	0.67	0.58
PaddleOCR	0.45	**0.11**
TrOCR	**0.22**	0.12
EasyOCR+GPT*	0.66	0.55
PaddleOCR+GPT*	**0.11**	**0.08**
TrOCR+GPT*	**0.11**	0.09

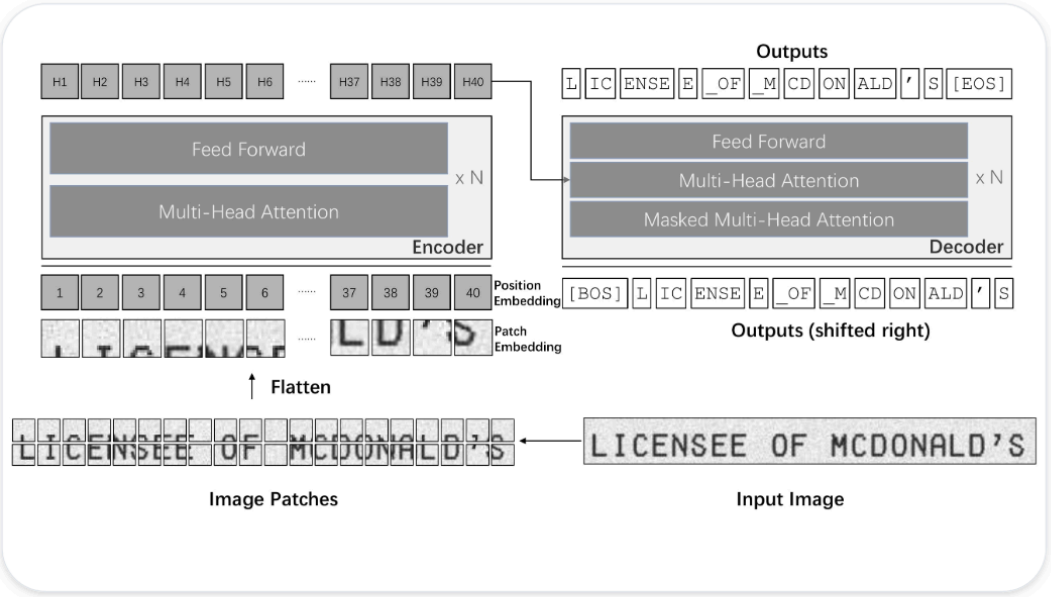
Table 3: The evaluation of line-level text was conducted using diverse models, with accuracy measured through the utilization of Word Error Rate (WER) and Character Error Rate (CER). In this context, "*GPT" specifically denotes the GPT-3.5-turbo-0301 model.

- Evaluation on Post-OCR Correction Models Table 5 demonstrates that the inclusion of glyph embedding leads to a significant enhancement in the WER and CER scores at the sentence level. However, the impact on the performance at the single-word level is minimal. Notably, the model exhibits proficient post-correction capabilities at the single-word level, as evidenced by the low CER and WER scores

OCR Output	Glyph	WER	CER
EasyOCR	False	-	0.0959
EasyOCR	True	-	**0.0292**
PaddleOCR	False	0.2642	0.1771
PaddleOCR	True	**0.0630**	**0.0283**
TrOCR	False	0.0859	0.0478
TrOCR	True	**0.0067**	**0.0013**

Table 5: The table provides a comprehensive comparison of post-OCR correction models that integrate glyph embedding with those that exclusively employ CharBERT embedding. The comparison encompasses various OCR model outputs.

- **TrOCR :**



TrOCR architecture. Taken from the original paper.

Links

https://www.youtube.com/watch?v=TQQlZhbC5ps
https://web.stanford.edu/class/cs25/
https://web.stanford.edu/class/cs25/
https://divyanshgarg.com/
https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1244/project.html
https://www.youtube.com/playlist?list=PLoROMvodv4rOSH4v6133s9LFPRHjEmbmJ

https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1234/project.ht
https://www.tensorflow.org/text/tutorials/word2vec

https://platform.openai.com/docs/guides/images

https://gemini.google.com/app/994b180b86adaa73

https://lena-voita.github.io/nlp_course/language_modeling.html#intro

https://arxiv.org/html/2408.02253v1

https://www.jaided.ai/easyocr/