Natural Language Processing with Language Models

Ruvan Weerasinghe

Informatics Institute of Technology

University of Colombo School of Computing

Overview

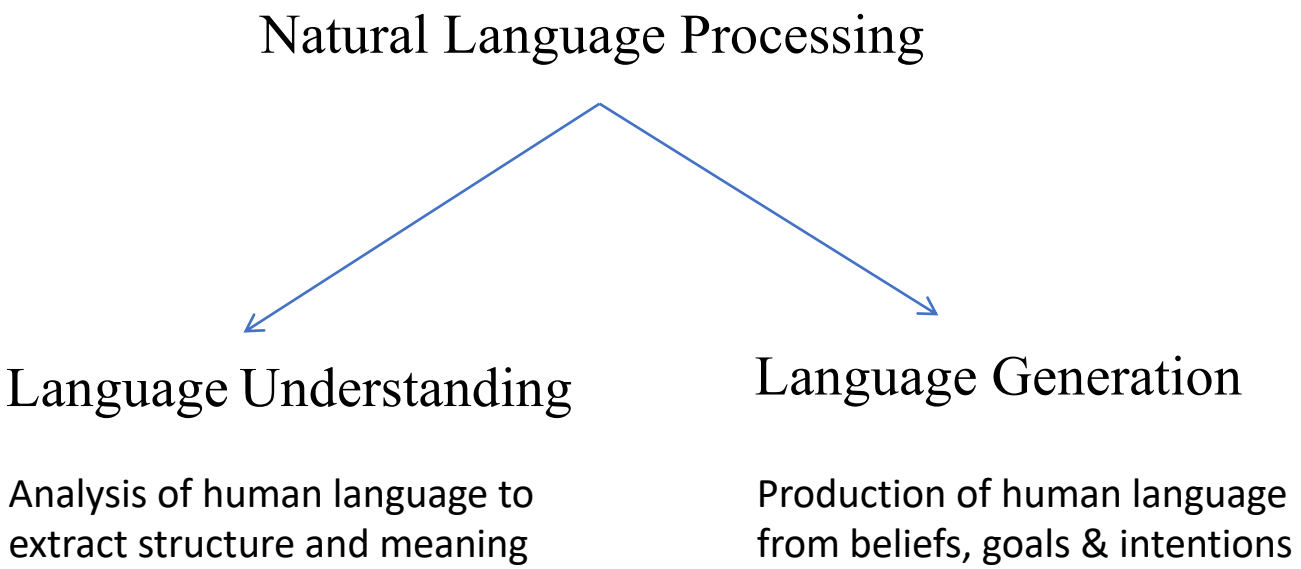
- NLP Now and Then
  - The Linguistic Stack
  - The Corpus-based model: n-grams
  - Dealing with longer contexts
  - The Transformer paradigm shift
- LLMs and Gen AI
  - LLMs under the hood
  - LLM apps
  - Customizing LLM instruct models

Overview

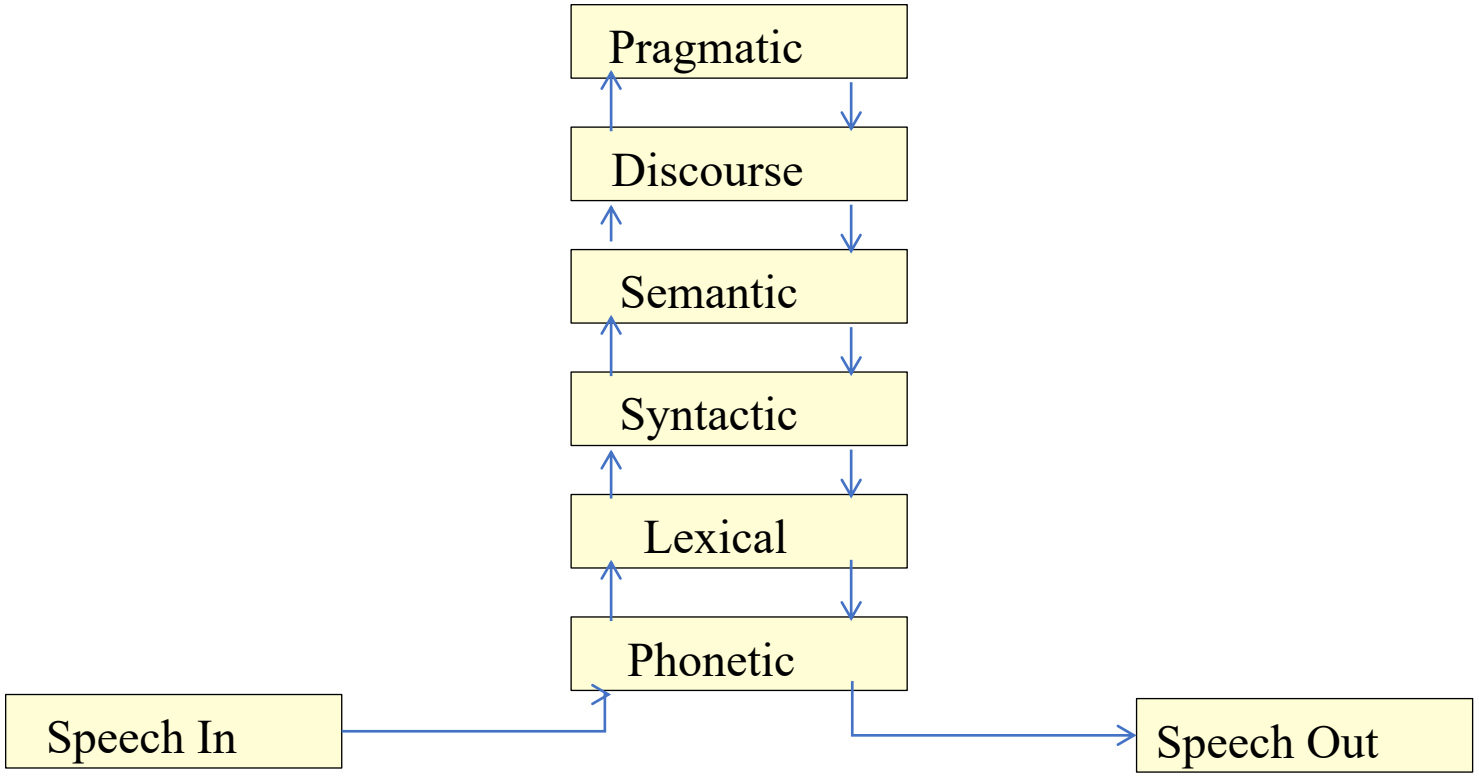
- NLP Now and Then
  - The Linguistic Stack
  - The Corpus-based model: n-grams
  - Dealing with longer contexts
  - The Transformer paradigm shift
- LLMs and Gen AI
  - LLMs under the hood
  - LLM apps
  - Customizing LLM instruct models

What is NLP?

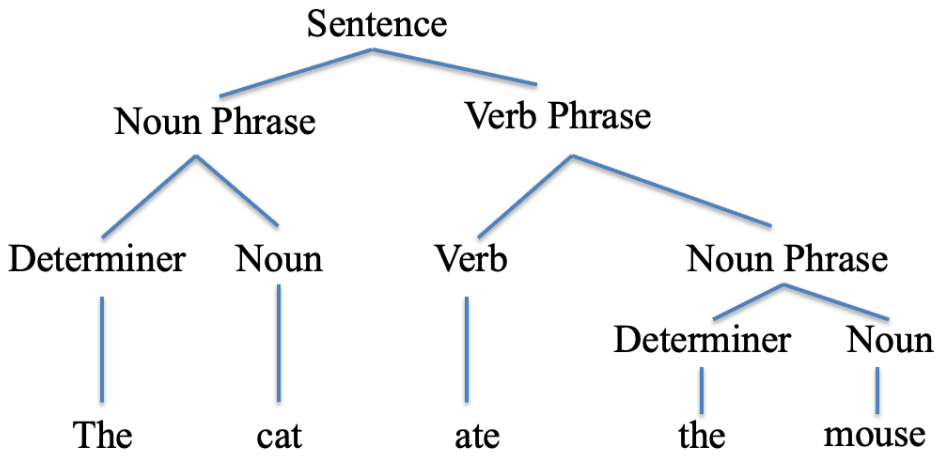
The two sides of NLP: *analysis* and *synthesis*



The Linguistic Stack



Typical Use Cases…



The phase structure rules underlying this analysis are as follows:

Sentence → Noun Phrase Verb Phrase

Noun Phrase → Determiner Noun

Verb Phrase → Verb Noun Phrase

Determiner = The

Noun = cat

Noun = mouse

Verb = ate

Parsing a sentence using simple phrase structure rules

But…

The cats ate the mouse X

The cats eats the mouse X

Can eat mouse X

The Corpus-based Model

- Ditching the grammar rules for *real* text!
- The Language Model idea
  - A model that can attach a probability for any given sentence
  - e.g. P(The cat chases the rat) > P(The cat chase the rat)
    also P(The cat chase the rat) >> P(Cat the rat chase the)
- How to do this?
  - Simple maximum likelihood estimates computed from actual text
- Use?
  - To predict the probability of a sentence
  - To predict the next word/phrase, given the previous word(s)

What is a Corpus?

- Corpus is a collection of written texts, especially the entire works of a particular author or a body of writing on a particular subject
- In linguistics, a corpus or text corpus is a large and structured set of texts
  - Google Books N-gram Corpus
  - American National Corpus - 22 million words of written and spoken data
  - COBUILD corpus - 4.5 billion words
  - British National Corpus - 100-million-word
  - Corpus of Contemporary American English (COCA) - 425 million words
  - UCSC 10M words Sinhala Corpus – 10 million words

Corpus-based Language Models

- **Goal:** compute the probability of a sentence or sequence of words:

  P(W) = P(w1,w2,w3,w4,w5…wn)

- Related task: probability of an upcoming word:
  - P(w5|w1, w2, w3, w4)
  - Conditional probability that w5 occurs, given that we know that w1, w2, w3, and w4 already occurred.
- A model that computes either of these is called a ***language model***
  - We might call this a grammar because it predicts the structure of the language, but the “language model” is the standard terminology.

Chain rule applied…

- Compute the probability of a sentence by computing the joint probability of all the words conditioned by the previous words

$$P(w_1 w_2 \cdots w_n) = \prod_i P(w_i \mid w_1 w_2 \cdots w_{i-1})$$

P(“the dog chased the cat”) =

P(the) x P(dog|the) x P(chased|the dog) x P(the|the dog chased)

x P(cat|the dog chased the)

Computing Probabilities

- Normally, we just compute the probability that something occurred by counting its occurrences and dividing by the total number

$$P(\text{the} \mid \text{its water is so transparent that}) = \frac{Count(\text{its water is so transparent that the})}{Count(\text{its water is so transparent that})}$$

- But there are way too many English sentences in any realistic corpus for this to work!
  - We'll never see enough data

Markov Assumption

- Instead we make the simplifying Markov assumption that we can predict the next word based on only one word previous:

$$P(\text{the} \mid \text{its water is so transparent that}) \approx P(\text{the} \mid \text{that})$$

- or perhaps two words previous:

$$P(\text{the} \mid \text{its water is so transparent that}) \approx P(\text{the} \mid \text{transparent that})$$

Andrei Markov

N-gram Models

- **Unigram Model:** The simplest case is that we predict a sentence probability just base on the probabilities of the words with no preceding words

$$P(w_1 w_2 \cdots w_n) \approx \prod_i P(w_i)$$

- **Bigram Model:** Prediction based on one previous word:

$$P(w_i \mid w_1 w_2 \cdots w_{i-1}) \approx P(w_i \mid w_{i-1})$$

Bigrams

- Examples of bigrams are any two words that occur together
  - In the text: "*two great and powerful groups of nations*", the bigrams are;
    "*two great*", "*great and*", "*and powerful*", etc.

- The ***frequency*** of an n-gram is the percentage of times the n-gram occurs in all the n-grams of the corpus and could be useful in corpus statistics
  - For bigram xy:
    - Count of bigram xy / Count of all bigrams in corpus

- But in bigram language models, we use the *bigram probability* to predict how likely it is that the second word follows the first

Example of Bigram probabilities

- Example mini-corpus of three sentences, where we have sentence detection and we include the sentence tags in order to represent the beginning and end of the sentence.

  <S> *I am Sam* </S>

  <S> *Sam I am* </S>

  <S> *I do not like green eggs and ham* </S>

- Bigram probabilities:

  P (I | <S>) (*probability that I follows* <S>) =

  P (</S> | Sam) =

  P (Sam | <S>) =

  P (Sam | am) =

  P (am | I) =

Example of Bigram probabilities

- Example mini-corpus of three sentences, where we have sentence detection and we include the sentence tags in order to represent the beginning and end of the sentence.

  <S> *I am Sam* </S>

  <S> *Sam I am* </S>

  <S> *I do not like green eggs and ham* </S>

- Bigram probabilities:

  P (I | <S>) = 2/3 = .67

  P (</S> | Sam) = ½ = .5

  P (Sam | <S>) = 1/3 = .33

  P (Sam | am) = ½ = .5

  P (am | I) = 2/3 = .67

Using N-Grams for sentences

- Bigram probabilities:

  P (I | <S>) = 2/3 = .67 (*probability that I follows* <S>)

  P (</S> | Sam) = ½ = .5

  P (Sam | <S>) = 1/3 = .33

  P (Sam | am) = ½ = .5

  P (I | Sam) = 1/4 = .25

  P (am | I) = 2/3 = .67

  P (</S> | am) = 1/8 = .125

<S> *I am Sam* </S> =

<S> *Sam I am* </S> =

Using N-Grams for sentences

- Bigram probabilities:

  P (I | <S>) = 2/3 = .67 (*probability that I follows* <S>)

  P (</S> | Sam) = ½ = .5

  P (Sam | <S>) = 1/3 = .33

  P (Sam | am) = ½ = .5

  P (I | Sam) = 1/4 = .25

  P (am | I) = 2/3 = .67

  P (</S> | am) = 1/8 = .125

<S> *I am Sam* </S> = .67 x .67 x .5 x .5 = .1122

<S> *Sam I am* </S> = .33 x .25 x .67 x .125 = .0069

Limitations of N-gram Models

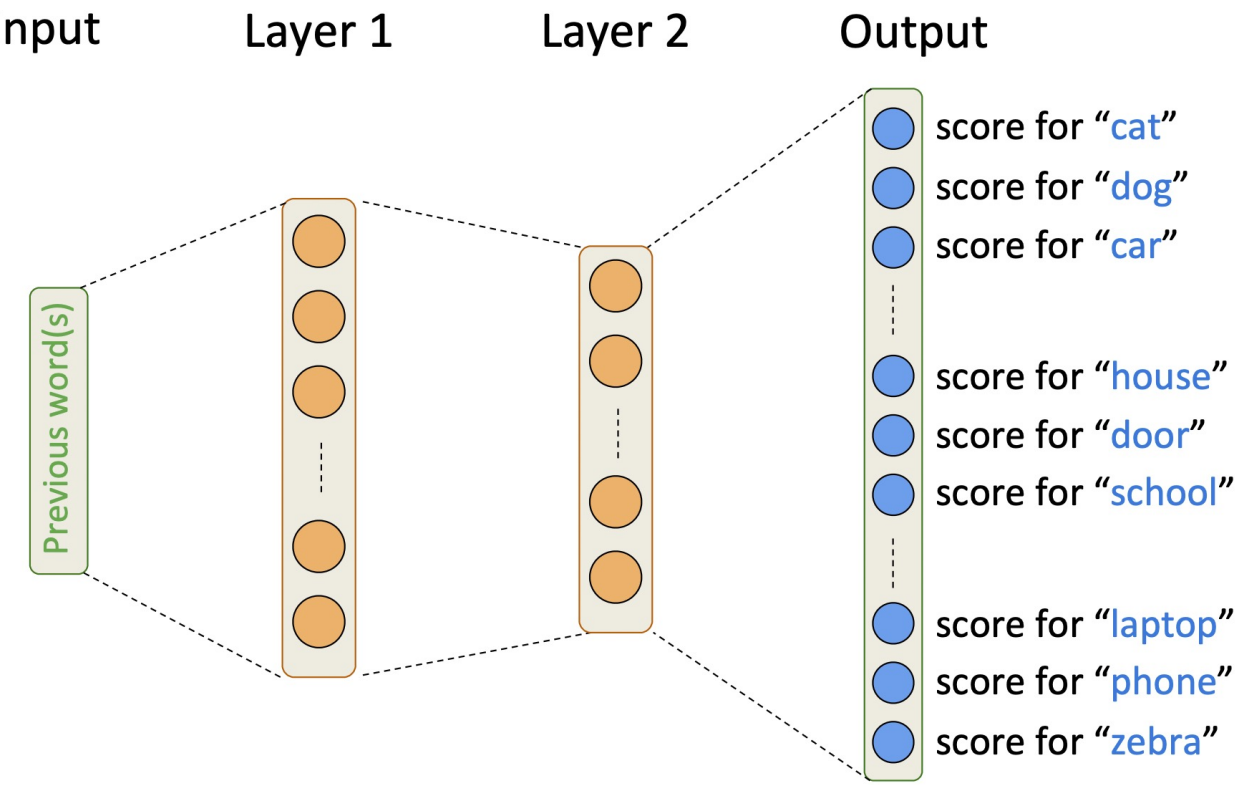
- We can extend to trigrams, 4-grams, 5-grams
  - Each higher number will get a more accurate model, but will be harder to find examples of the longer word sequences in the corpus
- In general this is an insufficient model of language
  - because language has long-distance dependencies:

"The computer which I had just put into the machine room on the fifth floor crashed."

  - the last word *crashed* is not very likely to follow the word *floor*, but it is likely to be the main verb of the word *computer*

Dealing with longer contexts

- The Neural Language Model



Representing words…

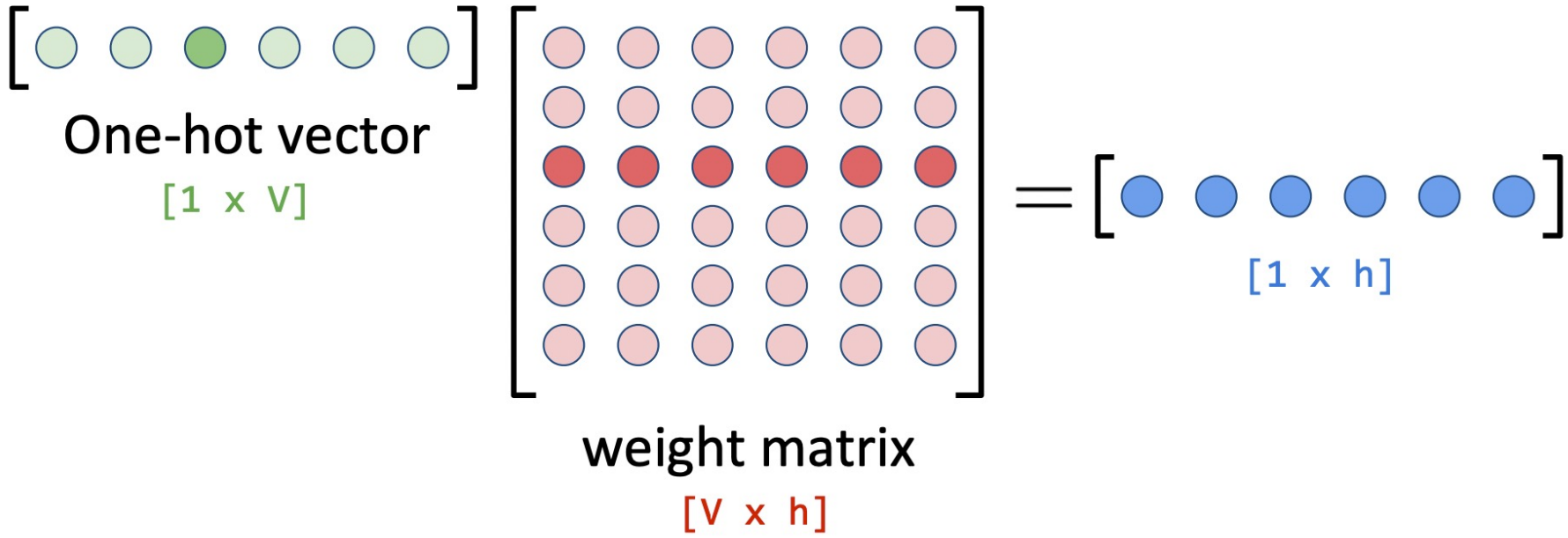
- The one-hot representation
  - Every word can be represented as a ***one hot vector***
  - Suppose the total number of unique words in the corpus is 10,000
  - Assign each word a unique index:

Vector size will be the size of the vocabulary, i.e. 10,000 in this case

$$cat = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \qquad car = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

One-hot representation

Representing words… (contd.)



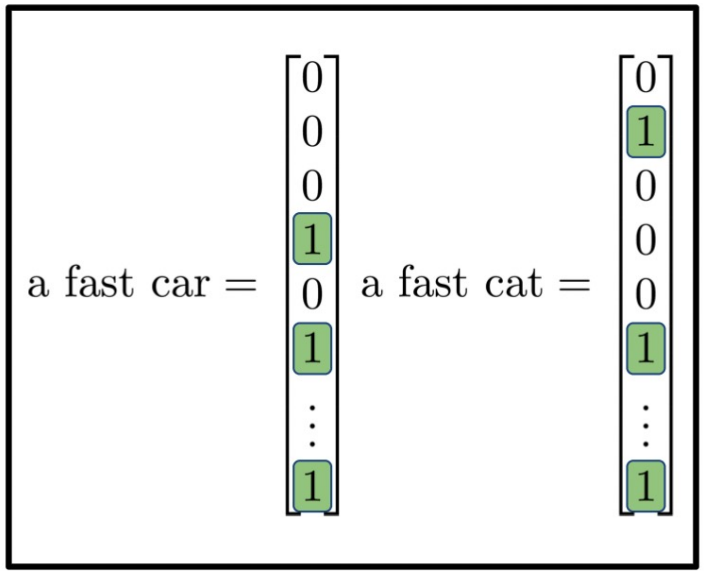
One-hot vector will “turn on” one row of weights

Representing words… (contd.)

What about representing multiple words?

Bag of words approach

Trigram: indices of the *three previous words* are 1 in the vector

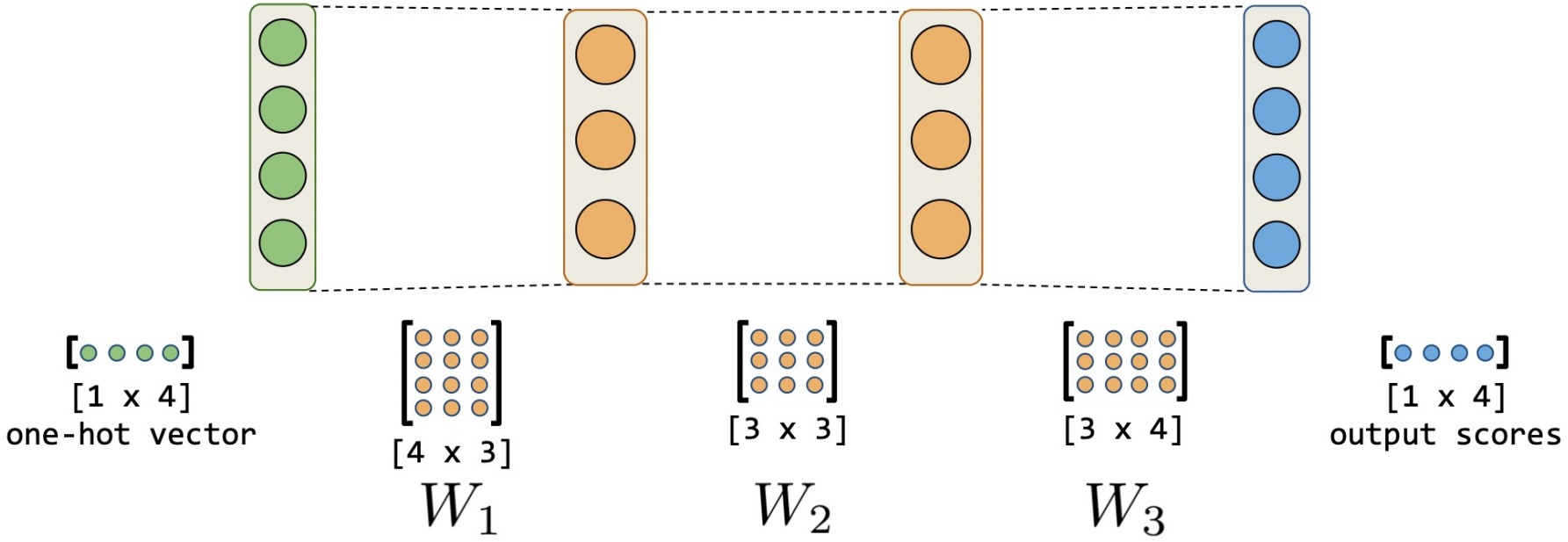


Representing words… (contd.)

Let us look at a complete example:

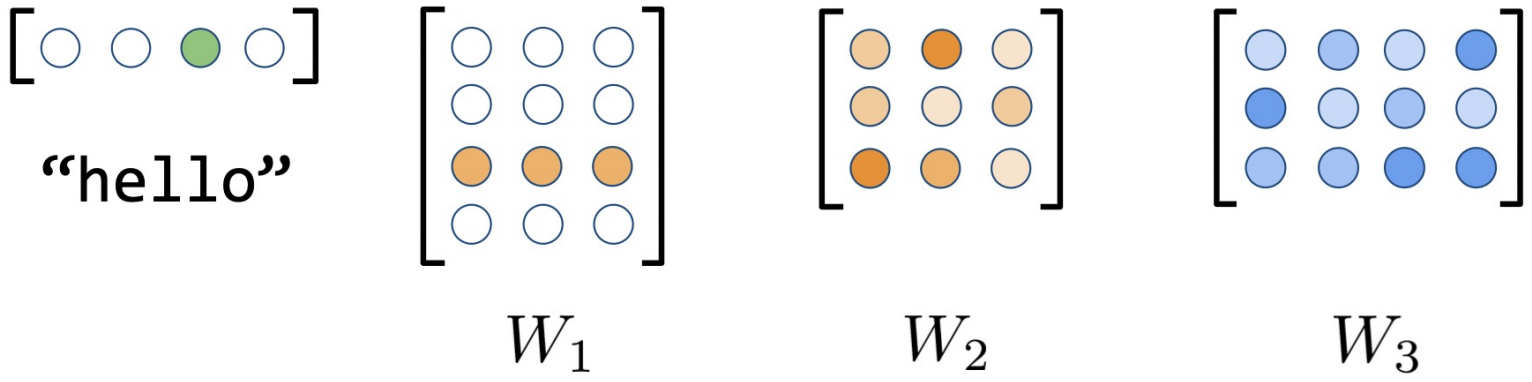
Vocabulary: {“how”, “you”, “hello”, “are”}

Network Architecture: 2 hidden layers of size 3 each

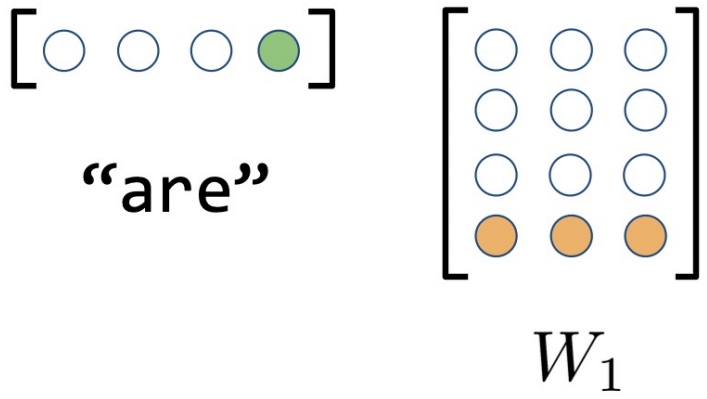


Representing words… (contd.)

Vocabulary: {“how”, “you”, “hello”, “are”}



Representing words… (contd.)



Each one-hot vector turns on one row in the weight matrix and results in `[1 x 3]` vector

Can we say that the `[1 x 3]` *vector represents the input word?*

Representing words… (contd.)

- In one-hot vector representation, a word is represented as one large ***sparse*** vector
- Instead, **word embeddings** are ***dense*** vectors in some vector space

word vectors are ***continuous*** representations of words

vectors of different words give us information about the potential relations between the words - words closer together in meaning have vectors closer to each other

Representing words… (contd.)

$$\text{cat} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \longrightarrow \text{cat} = \begin{bmatrix} 0.523 \\ -0.813 \\ 0.417 \\ 1.738 \\ -0.145 \\ -3.451 \\ \vdots \\ 0.451 \end{bmatrix}$$

one-hot vector

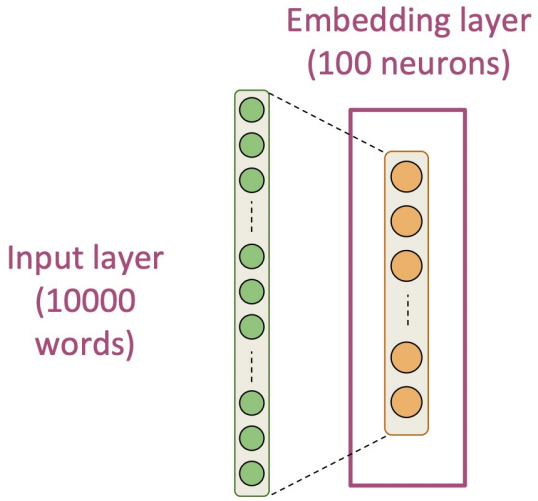
word embedding

Inherit benefits

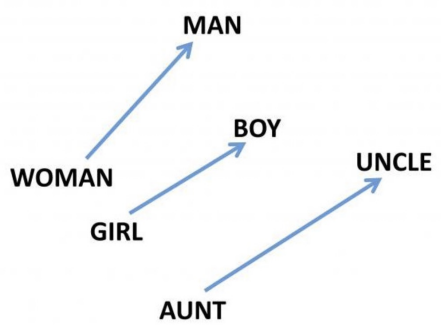
- Reduce dimensionality
- Semantic relatedness
- Increase expressiveness
  - one word is represented in the form of several features (numbers)

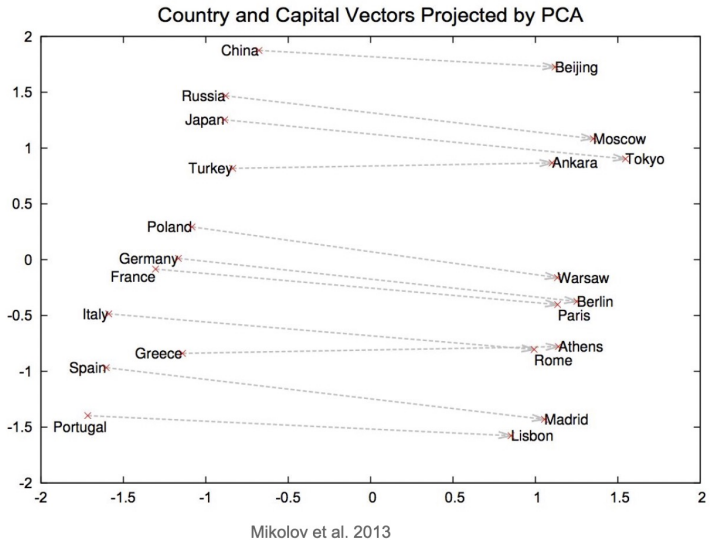
Representing words… (contd.)

- Advantages of embeddings
  - i.e. dense representations
- Reduction of dimensionality



- Induction of semantic relatedness

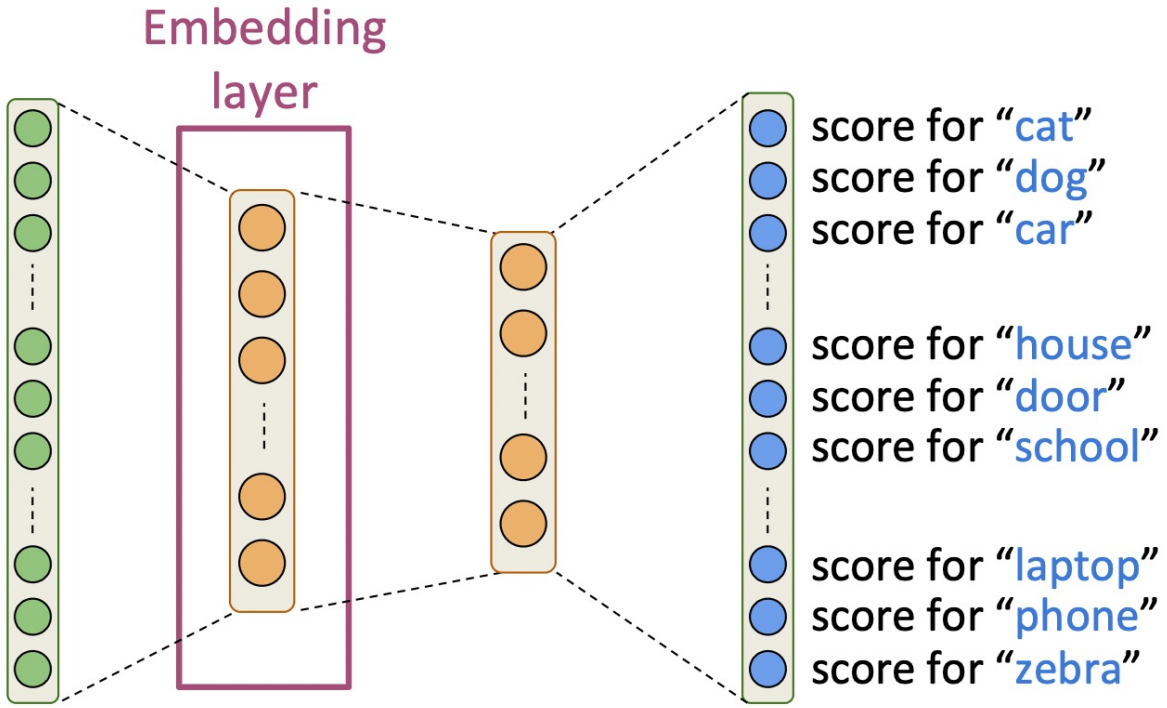




Mikolov et al. 2013

Representing words… (contd.)

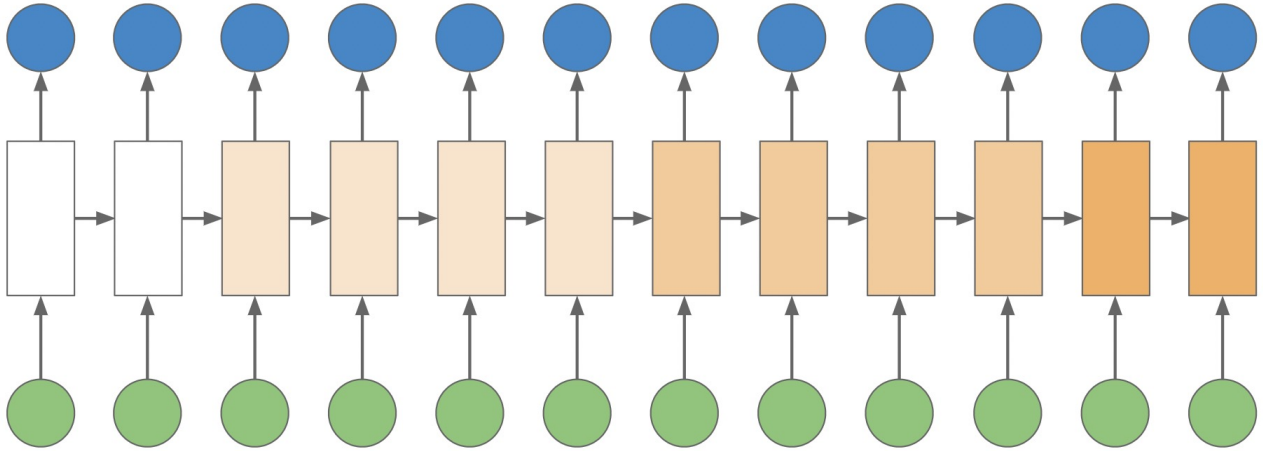
- Inducing vectors from text
  - Using Neural Networks



- Word2Vec (from Google)
- FastText (from Facebook)
- GloVe (from Stanford)

Tracking longer contexts with RNNs

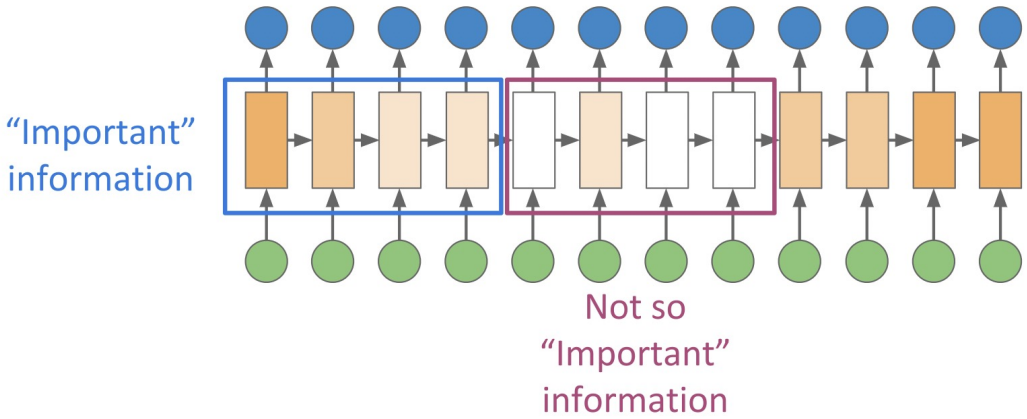
- The unreasonable effectiveness of Recurrent Neural Networks!



- The remaining problems
  - Information decay
  - Vanishing gradients

Tracking longer contexts with RNNs (contd.)

- Using LSTM units in the RNN

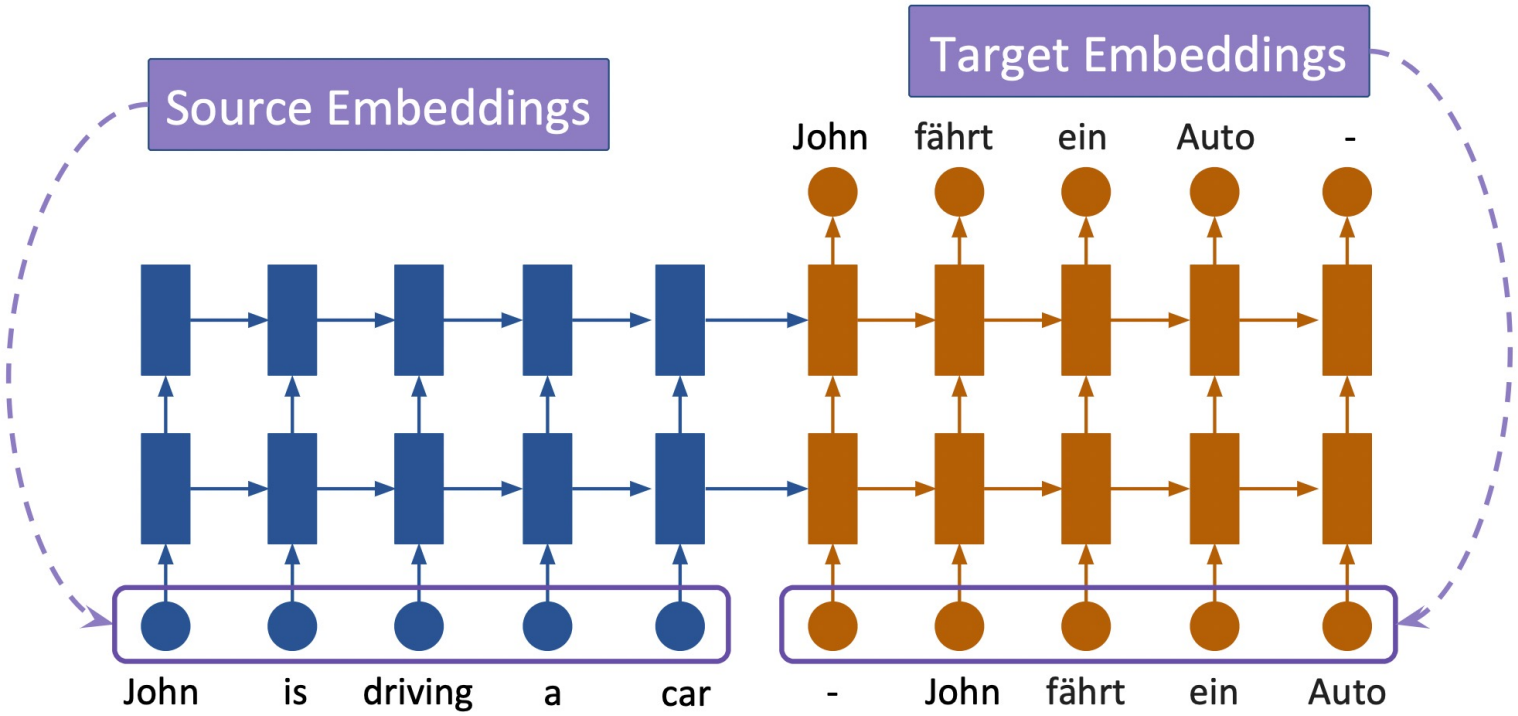


Alice studies computational linguistics. **She** is currently learning about LSTM's. Bob on the other hand studies about cyber security. He is completely confused right now!

Alice studies computational linguistics. She is currently learning about LSTM's. **Bob** on the other hand studies about cyber security. **He** is completely confused right now!

Tracking longer contexts with RNNs (contd.)

- Sequence-to-sequence models

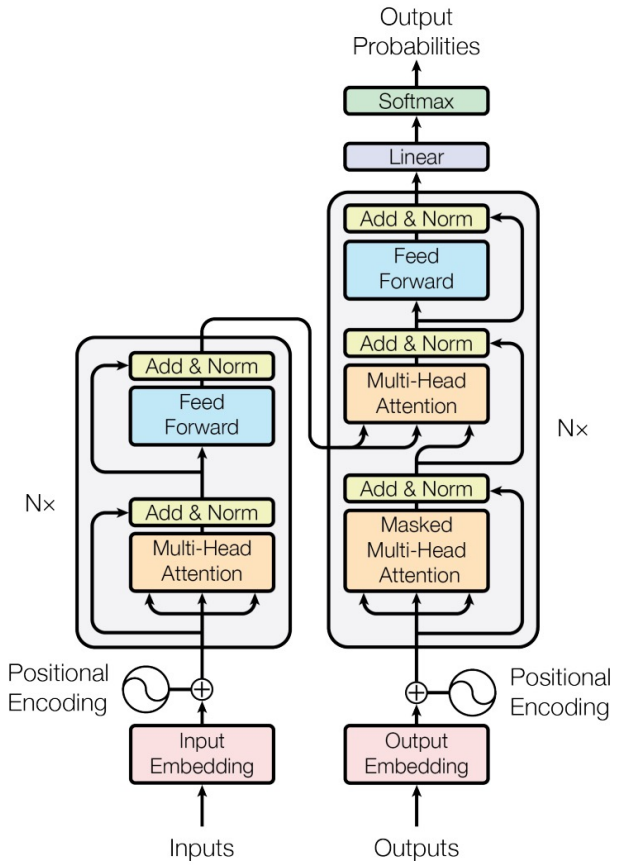


The Transformer – Attention is all you need (2017)

Implemented by Google in 2018

BERT

Encoder

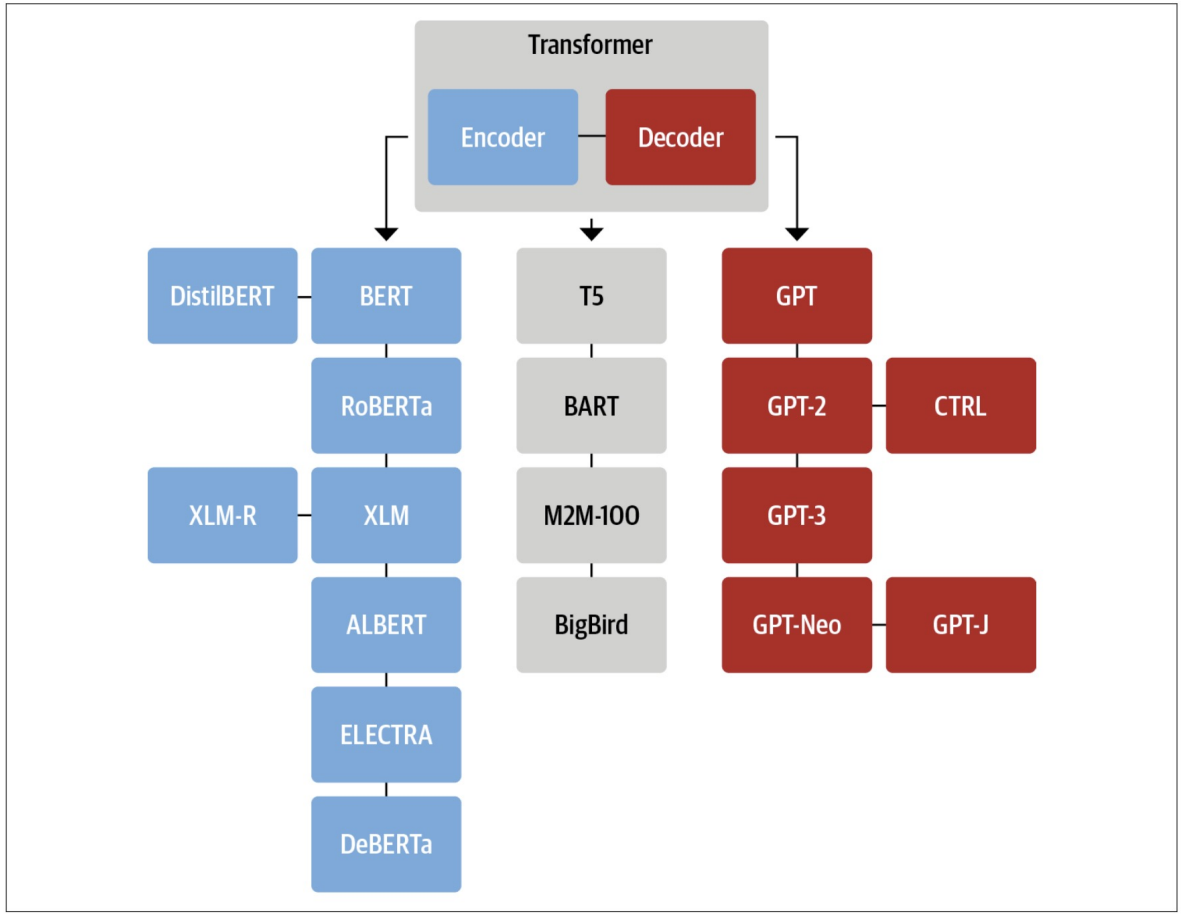


GPT

Decoder

Implemented by OpenAI in 2018

Transformer Families

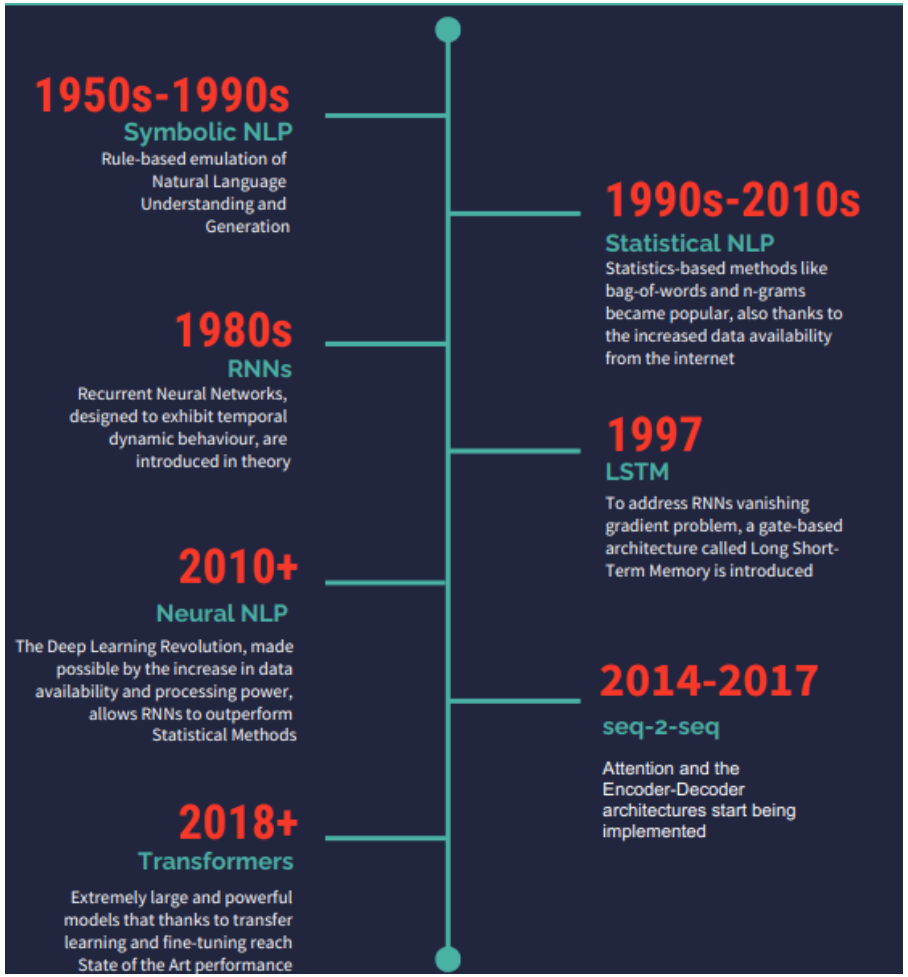


Article on BERT variants - https://360digitmg.com/blog/bert-variants-and-their-differences

Article on GPT models - https://www.kdnuggets.com/2023/05/deep-dive-gpt-models.html

tl;dr – Language Modelling

- Simulating the Linguists' account
  - Thousands of rules and exceptions!
- Observing from real text
  - Statistical models with n-grams
- Accounting for longer sequences
  - Deep Neural Networks models
- Basic task of a language model
  - Assign a probability to a sentence (from data)
  - To be able to predict the next word, phrase, sentence…

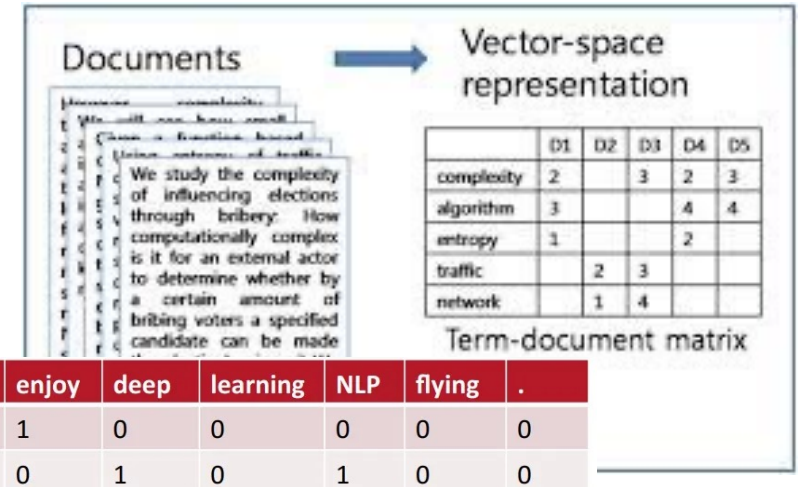


tl;dr – Language Modelling Breakthrough

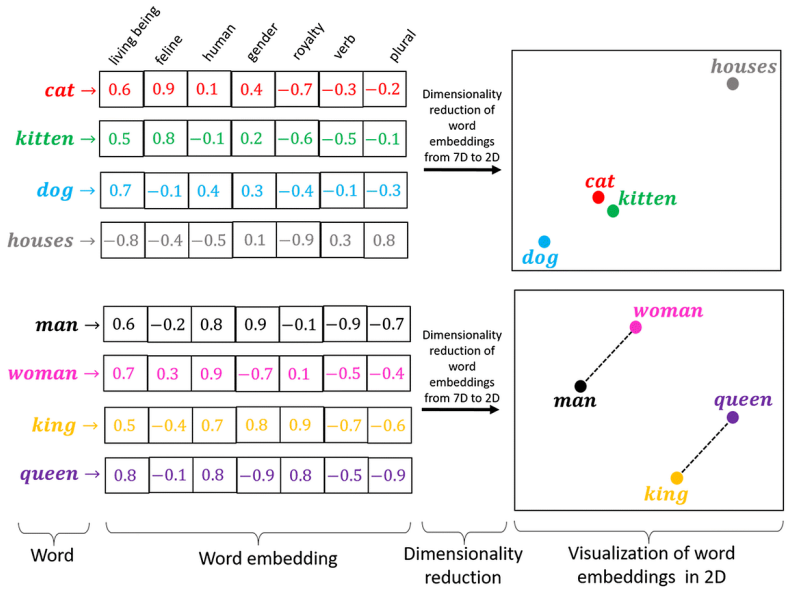
- A way of **representing words**, documents… text in general
  - From characters and words to numeric vectors (2013)
  - Leading to *contextual embedding* (2018)
- A way of **remembering enough** to predict what could come next
  - Deep learning with attention (LSTM)
- A way of **computing efficiently**!
  - *Attention is all you need* (2017) – the parallelized encoder-decoder *Transformer* paper
  - Realized in BERT (2018) – the *encoder* part
  - Realized in GPT (2018) – the *decoder* part

tl;dr – Language Modelling (representation)

- Representing words…
  - Document – Term Matrix
  - Term – Term Matrix (sparse vectors)
  - Word embeddings (dense vectors)



	D1	D2	D3	D4	D5
complexity	2		3	2	3
algorithm	3			4	4
entropy	1			2	
traffic		2	3		
network		1	4		



counts	I	like	enjoy	deep	learning	NLP	flying	.
I	0	2	1	0	0	0	0	0
like	2	0	0	1	0	1	0	0
enjoy	1	0	0	0	0	0	1	0
deep	0	1	0	0	1	0	0	0
learning	0	0	0	1	0	0	0	1
NLP	0	1	0	0	0	0	0	1
flying	0	0	1	0	0	0	0	1
.	0	0	0	0	1	1	1	0

40k numbers for each word?

300 - 900 numbers for each word?

Tools for Natural Language Processing

- Mostly python based
- NLTK, Spacy
- Scikit-Learn, Gensim
- Keras/Tensorflow, Pytorch
- Hugging Face Transformers

Overview

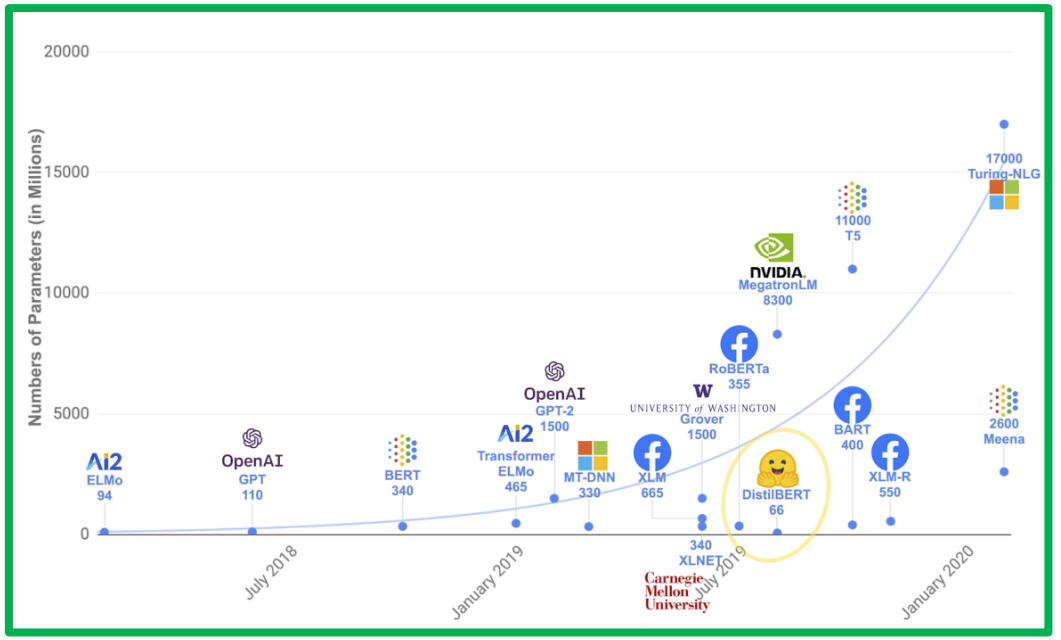
- NLP Now and Then
  - The Linguistic Stack
  - The Corpus-based model: n-grams
  - Dealing with longer contexts
  - The Transformer paradigm shift
- LLMs and Gen AI
  - LLMs under the hood
  - LLM apps
  - Customizing LLM instruct models

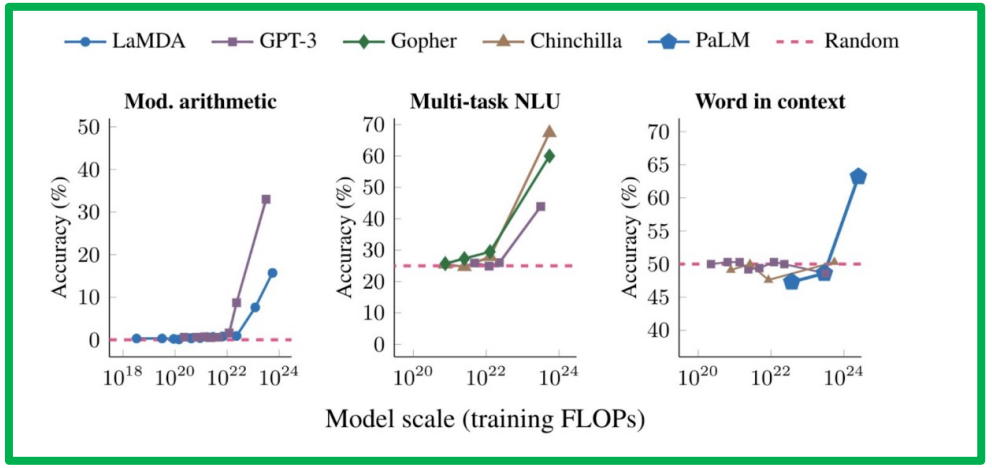
Overview

- NLP Now and Then
  - The Linguistic Stack
  - The Corpus-based model: n-grams
  - Dealing with longer contexts
  - The Transformer paradigm shift
- LLMs and Gen AI
  - LLMs under the hood
  - LLM apps
  - Customizing LLM instruct models

From LMs to LLMs…

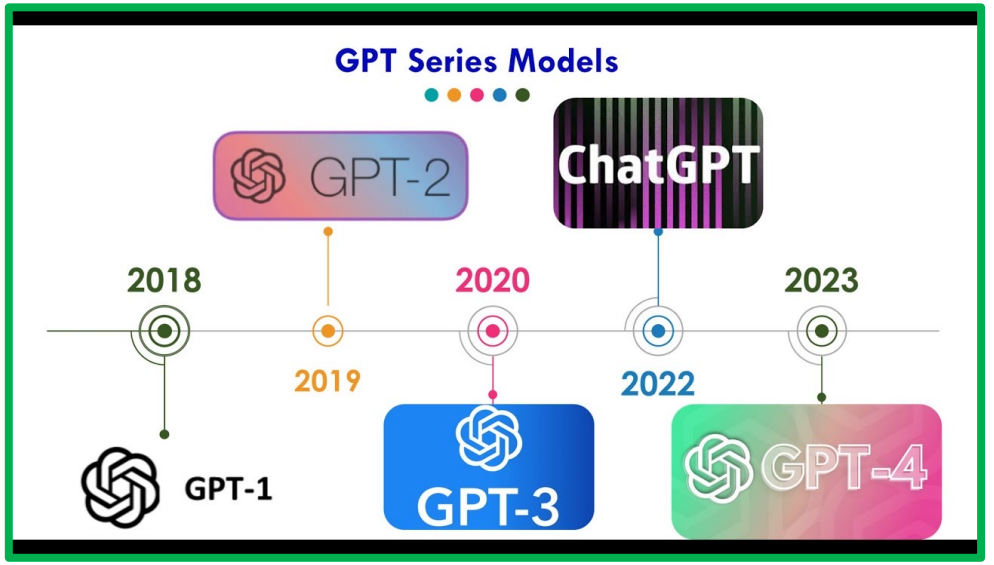
- Rule-based (Grammars)
  - A 1000 interacting rules!
- Statistical LMs
  - Thousands to Tens of thousands of parameters
- Large LMs (LLMs)
  - Millions to hundreds of millions of parameters
  - e.g. ELMo (94m), GPT (117m); GPT3 (175b)…
- *Emergent behaviour* of LLMs
  - Seen in GPT series from ~3.5

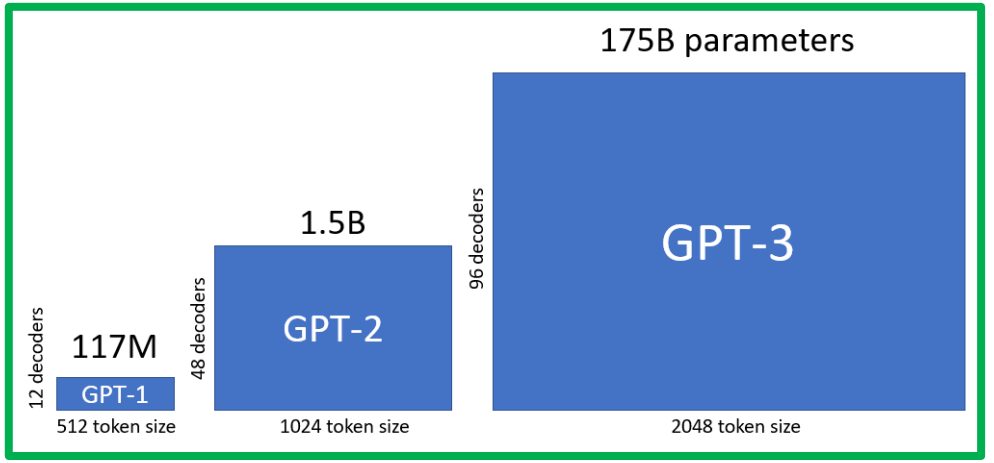




The GPT phenomenon…

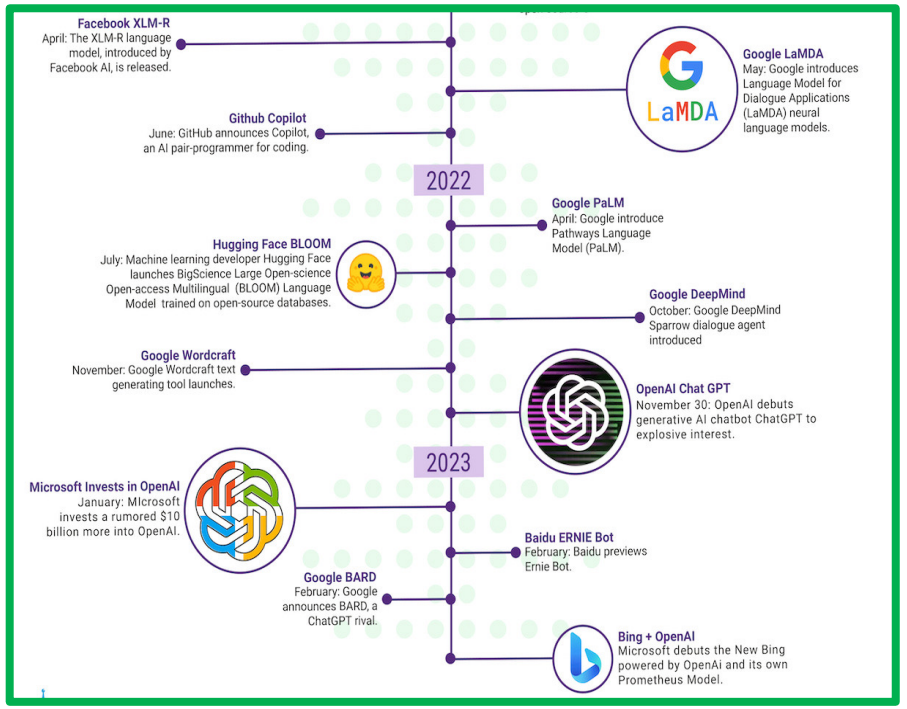
- OpenAI setup to democratize AI
  - Has since changed its stance
  - Owing to danger or commercial interest (esp. after Microsoft funding)
- GPT-3 available via API for 2 years
  - For largely text completion
- GPT on steroids – ChatGPT
  - Unsupervised LLM
  - Supervised using instructions and responses
  - Reinforced using RLHF
  - No details provided, ostensibly owing to danger!
- GPT-4 can do image and audio too!

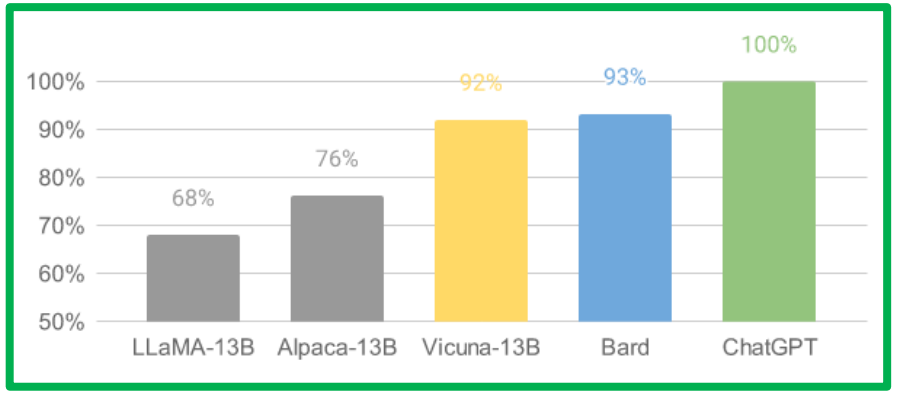




The explosion of LLMs in 2023

- The AI race among the big players
  - From 94m parameter Elmo to possibly 1+t parameters of GPT-4
- Meta's LLaMA 'leak'
  - A 65b parameter LLM
- The rise of the Open Source LLM
  - Stanford's Alpaca based on LLaMA
  - Vicuna, Falcon and Mistral…
- How do these stack up to the 'big boys'?
  - In short, pretty well!





Under the hood of LLM-based Chatbots

- So, how does an LLM become at conversational agent?
  - After all it should be only able to '*auto complete*'
- Supervised fine-tuning was the trick that changed the game!
  - *Instruct model* training – aka supervised fine tuning (SFT)
  - *Unclear* how much RLHF helped
- Still requires 'stopping' continuous output!
  - By detecting the *follow up question*!!
- What about math, coding etc.?
  - Requires parsing user input and determining *which API to use*!
- What about inappropriate prompts?
  - Requires 'ad hoc' guardrails!

All need to be attended to for DIY chatbots a la *Langchain*

Getting the most out of Gen AI

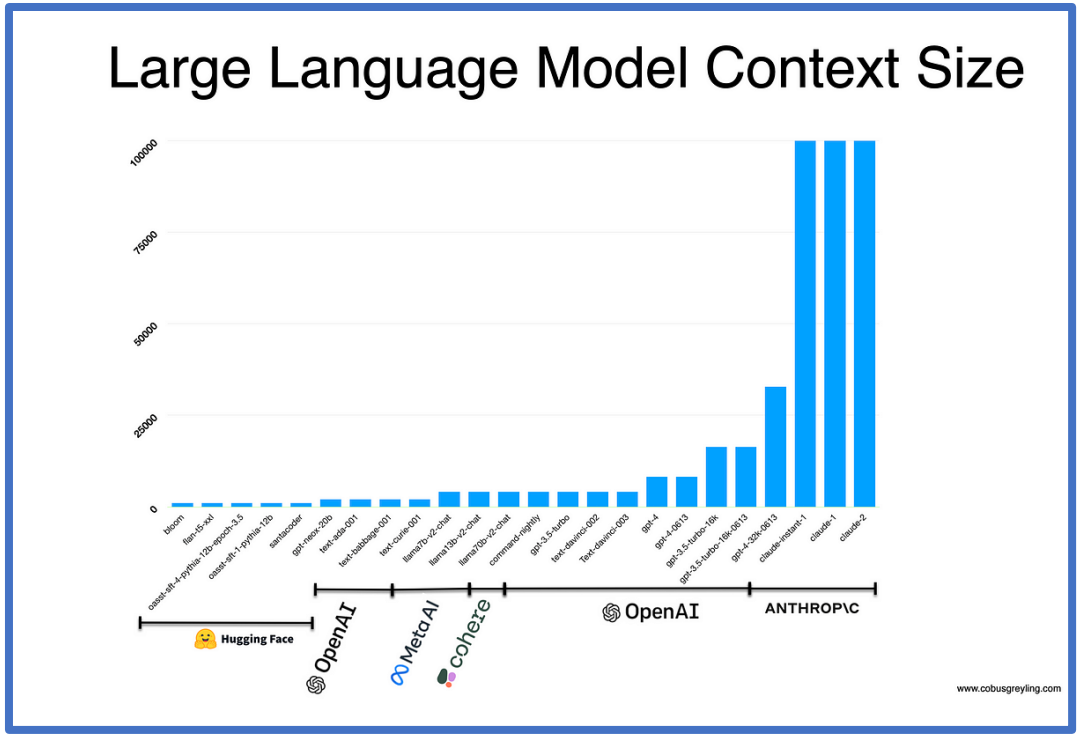
- Engineer your prompts – e.g.

  Act as <persona> to perform <task> for <target audience> in <output format>
  <persona> = historian, movie critic, Einstein, English teacher, legal expert
  <task> = write article, blog post, tweet, summarize, rewrite, code, visualize
  <target audience> = beginner, college student, 5 year old, grandmother
  <output format> = HTML, python, markdown, table, graph, excel, jason

- A list of lists for prompt engineering:
  https://medium.com/mlearning-ai/the-chatgpt-list-of-lists-a-collection-of-1500-useful-mind-blowing-and-strange-use-cases-8b14c35eb

Getting the most out of Gen AI

- How to get even more out of it?
- Remembering conversation thread
- Using examples to guide the response (few-shot learning)
- Answering questions not in model (RAG)
- Fitting all this in the LLM's context window
- Requires a framework!



LLM app Frameworks

Langchain
Llamaindex
AgentGPT
BabyAGI
Auto-GPT
AutoGen

Langchain for LLM Apps

- Components of the Langchain framework
  - Large language models (including open-source models from Hugging Face)
  - Prompt templates
  - Chains to combine the components
  - Indexes to access external data
  - Memory to remember previous conversations
  - Agents for accessing other tools via APIs
- Langchain has a rich collection of templates for common use cases
  - Different from ‘prompt templates’

Langchain Models

- Large language models (LLMs)
  - Proprietary models such as from OpenAI or Google
  - Or open-source ones from Hugging Face
  - Input is usually encoded in embeddings (proprietary or open-source)

```python
# Proprietary LLM from e.g. OpenAI
# pip install openai
from langchain.llms import OpenAI
llm = OpenAI(model_name="text-davinci-003")

# Alternatively, open-source LLM hosted on Hugging Face
# pip install huggingface_hub
from langchain import HuggingFaceHub
llm = HuggingFaceHub(repo_id = "google/flan-t5-xl")

# The LLM takes a prompt as an input and outputs a completion
prompt = "Alice has a parrot. What animal is Alice's pet?"
completion = llm(prompt)
```

```python
# Proprietary text embedding model from e.g. OpenAI
# pip install tiktoken
from langchain.embeddings import OpenAIEmbeddings
embeddings = OpenAIEmbeddings()

# Alternatively, open-source text embedding model hosted on Hugging Face
# pip install sentence_transformers
from langchain.embeddings import HuggingFaceEmbeddings
embeddings = HuggingFaceEmbeddings(model_name = "sentence-transformers/all-Min

# The embeddings model takes a text as an input and outputs a list of floats
text = "Alice has a parrot. What animal is Alice's pet?"
text_embedding = embeddings.embed_query(text)
```

Langchain Prompts

- Beyond simple queries to engineered prompts
  - Used to create prompt templates
  - More than just a single component
  - Provides for few-shot prompts too

```python
from langchain import PromptTemplate

template = "What is a good name for a company that makes {product}?"

prompt = PromptTemplate(
    input_variables=["product"],
    template=template,
)

prompt.format(product="colorful socks")
```

```python
from langchain import PromptTemplate, FewShotPromptTemplate

examples = [
    {"word": "happy", "antonym": "sad"},
    {"word": "tall", "antonym": "short"},
]

example_template = """
Word: {word}
Antonym: {antonym}\n
"""

example_prompt = PromptTemplate(
    input_variables=["word", "antonym"],
    template=example_template,
)

few_shot_prompt = FewShotPromptTemplate(
    examples=examples,
    example_prompt=example_prompt,
    prefix="Give the antonym of every input",
    suffix="Word: {input}\nAntonym:",
    input_variables=["input"],
    example_separator="\n",
)

few_shot_prompt.format(input="big")
```

Langchain Chains

- Combining LLMs with other components
  - With prompt templates
  - With other LLMs
  - With external data
  - With long term memory (chat history)

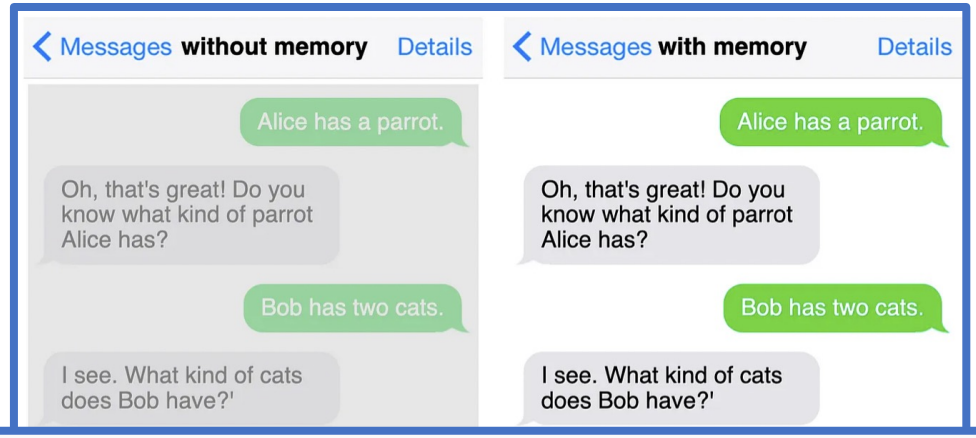
Langchain Indexes

- Indexing external data for retrieval augmented generation (RAG)
  - Access the external data
  - Convert using text embedding
  - Insert the data in a vector database
  - Retrieve the data based on prompt

```
from langchain.chains import RetrievalQA

retriever = db.as_retriever()

qa = RetrievalQA.from_chain_type(
    llm=llm,
    chain_type="stuff",
    retriever=retriever,
    return_source_documents=True)

query = "What am I never going to do?"
result = qa({"query": query})

print(result['result'])
```

Langchain Memory

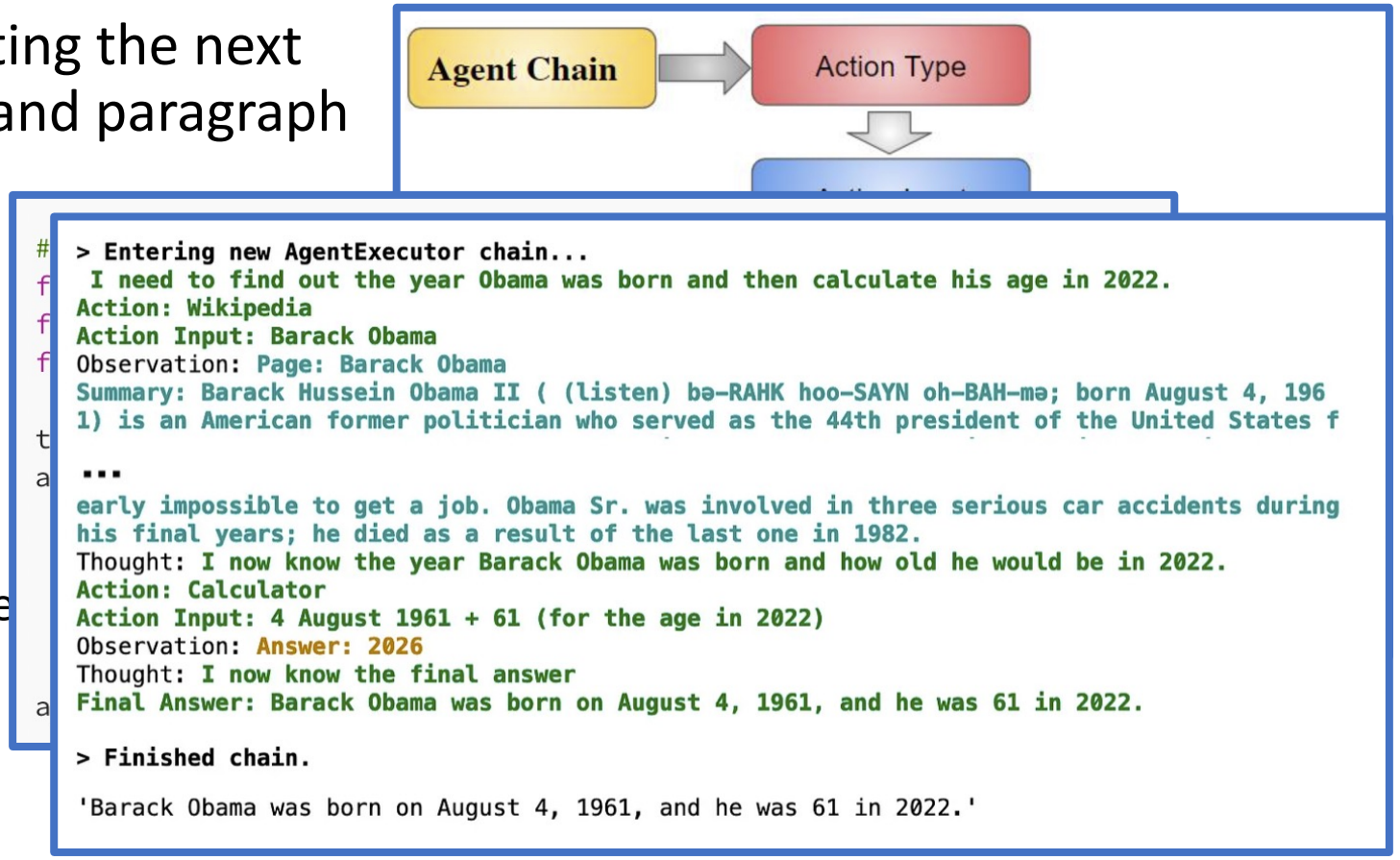
- Chat interactions are stateless – so you need to persist them!
  - Remembering previous interactions
  - Using a conversation chain
    - To keep all, last k or a summary
  - Is an essential part of the context
  - For interpreting follow up prompts



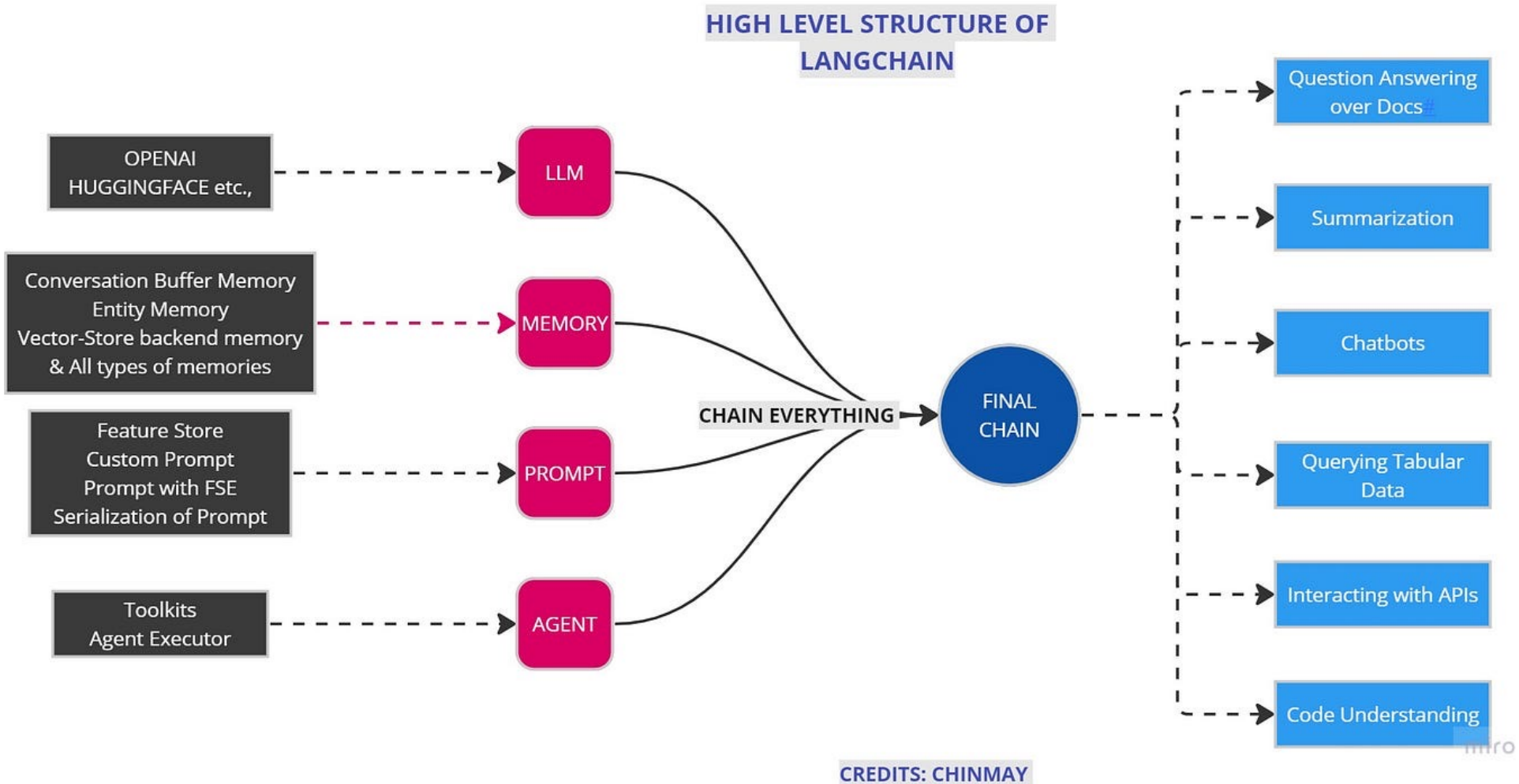

```
from langchain import ConversationChain

conversation = ConversationChain(llm=llm, verbose=True)

conversation.predict(input="Alice has a parrot.")

conversation.predict(input="Bob has two cats.")

conversation.predict(input="How many pets do Alice and Bob have?")
```

Langchain Agents

- LLMs are good at predicting the next word, phrase, sentence and paragraph
- A chatbot needs to use *tools* to do other things
  - Answer math problems
  - Execute arbitrary code
    - REPL, Wolfram Alpha
  - Lookup databases
  - Decide which tools to use



Langchain in a slide!



The use cases of LLM apps

- Using prompt templates
  - Simplest customization – little effort
- Using Retrieval Augmented Generation (RAG)
  - Effective for specialized narrow domains – possible on normal laptop
- Using supervised fine-tuning
  - Useful for domain specific LLMs and instruct models
- Full agent functionality
  - Accessing multiple knowledge sources for achieving tasks
- Towards training from scratch…!
  - Only for large corporates and countries!