Machine Translation

- Rule-based Machine Translation (RBMT): 1970s-1990s
- Statistical Machine Translation (SMT): 1990s-2010s
- Neural Machine Translation (NMT): 2014-

- **Rule-based Machine Translation**
- A rule-based system requires experts' knowledge about the source and the target language to develop syntactic, semantic and morphological rules to achieve the translation.
- An RBMT system contains a pipeline of Natural Language Processing (NLP) tasks including Tokenisation, Part-of-Speech tagging and so on. Most of these jobs have to be done in both source and target language.

- Apterium is open-source RBMT software released under the terms of GNU General Public License. It is available in 35 languages and it is still under development. It was originally designed for languages closely related to Spanish.
- SYSTRAN is one of the oldest Machine Translation company. It translates from and to around 20 languages.

Statistical Machine Translation or SMT

Statistical Machine Translation (SMT) learns how to translate by analyzing existing human translations (known as bilingual text corpora)

It expects to decide the correspondence between a word from the source language and a word from the objective language.

A genuine illustration of this is Google Translate.

Idea…

- *Given a sentence T in the target language, we seek the sentence S from which the translator produced T. We know that our chance of error is minimized by choosing that sentence S that is most probable given T. Thus, we wish to choose S so as to maximize Pr(S|T).*

Example of SMT

- Google Translate (between 2006 and 2016, when they announced to change to NMT)
- Microsoft Translator (in 2016 changed to NMT)
- Moses: Open source toolkit for statistical machine translation.

Approaches to MT

SMT

Source (S) ⟶ Target (T)

$$\hat{T} = \underset{T}{argmax}\, P(T|S)$$

$$\hat{T} = \underset{T}{argmax}\, \underbrace{P(S|T)}_{\text{Translation model}}\, \underbrace{P(T)}_{\text{Language model}}$$

SMT

Statistical Machine Translation

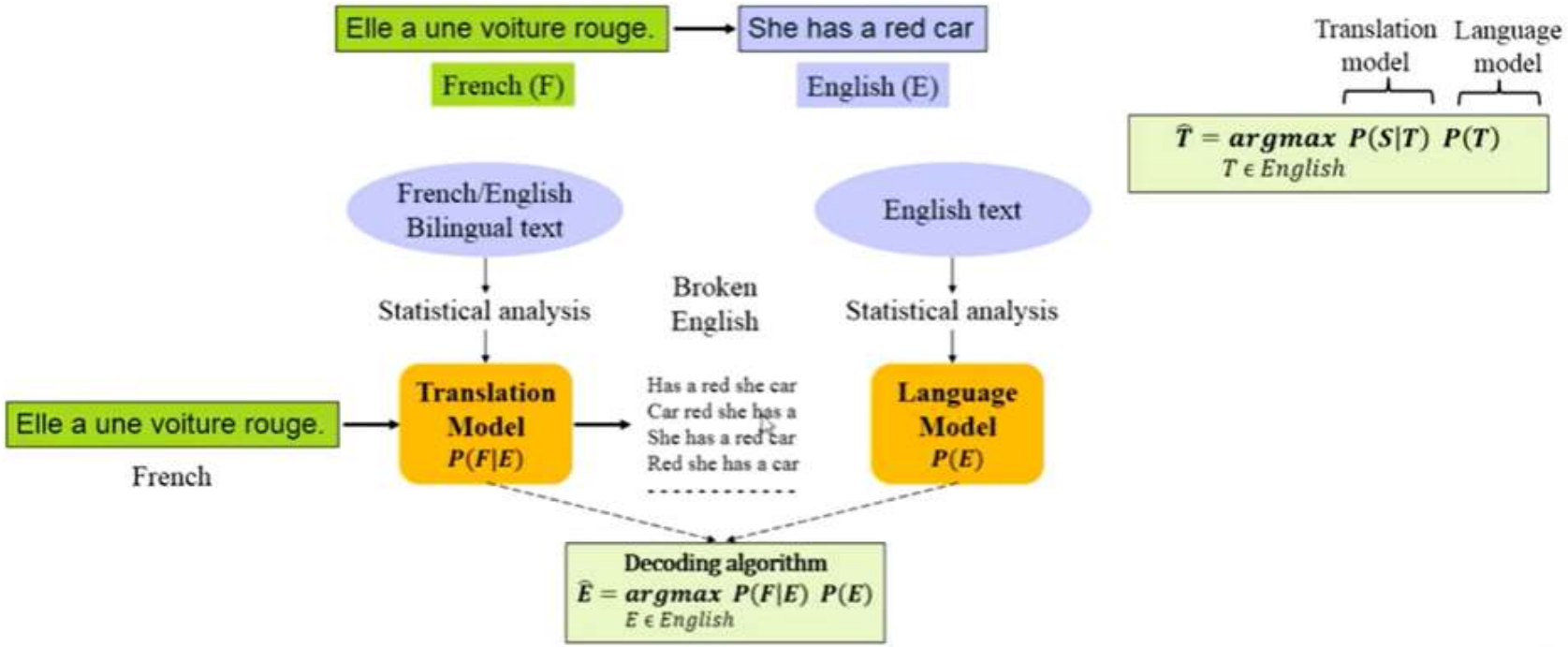
- Word based (WBMT)
- Syntax based (SBMT)
- Phrase based (PBMT)

Bayes' rule $P(A|B) = \dfrac{P(B|A)\,P(A)}{P(B)}$

- **Translation model**: How probable T is as a translation of sentence S. What is the probability that S comes from T?
- **Language model**: How probable the sentence T is in target language. (helps in choosing the grammatically correct sentence. i.e. it will prefer 'I go' over 'go I')
- **Decoder**: Given S, the translation and language model, produces the most probable T.

Approaches to MT

SMT



Advantages

- Less manual work from linguistic experts
- One SMT suitable for more language pairs
- Less out-of-dictionary translation: with the right language model, the translation is more fluent

Disadvantages

- Requires bilingual corpus
- Specific errors are hard to fix
- Less suitable for language pairs with big differences in word order

Neural Machine Translation

The neural approach uses neural networks to achieve machine translation. Compared to the previous models, NMTs can be built with one network instead of a pipeline of separate tasks.

NMT examples

- Google Translate (from 2016) link to language team at Google AI
- Microsoft Translate (from 2016) link to MT research at Microsoft
- Translation on Facebook: link to NLP at Facebook AI
- OpenNMT: An open-source neural machine translation system. [16]

- A problem with neural networks occurs if the training data is unbalanced, the model cannot learn from the rare samples as well as frequent ones.

MUSE

- MUSE (Multilingual Unsupervised and Supervised Embeddings) is a Python library that enables faster and easier development and evaluation of cross-lingual word embeddings and natural language processing. This library enables researchers and developers to ship their AI technologies to new languages faster.

- MUSE is a Python library that is meant for multilingual word embeddings, and whose goal is to provide the community with:

- state-of-the-art multilingual word embeddings that are based on fastText

- large-scale high-quality bilingual dictionaries for the purpose of training and evaluation

BERT

- At the end of 2018 researchers at Google AI Language open-sourced a new technique for Natural Language Processing (NLP) called **BERT** (Bidirectional Encoder Representations from Transformers)

Why was BERT needed??

- One of the biggest challenges in NLP is the lack of enough training data.
- in order to perform well, deep learning-based NLP models require much larger amounts of data.
- To help bridge this gap in data, researchers have developed various techniques for training general purpose language representation models using the enormous piles of unannotated text on the web (this is known as ***pre-training***).
- These general purpose pre-trained models can then be ***fine-tuned*** on smaller task-specific datasets

- we can either use the BERT models to extract high quality language features from our text data.
- we can fine-tune these models on a specific task, like sentiment analysis and question answering, with our own data to produce state-of-the-art predictions.

Example

- “The woman went to the store and bought a _____ of shoes.”

- In the pre-BERT world, a language model would have looked at this text sequence during training from either left-to-right or combined left-to-right and right-to-left.
- This one-directional approach works well for generating sentences
- we can predict the next word, append that to the sequence, then predict the next to next word until we have a complete sentence.

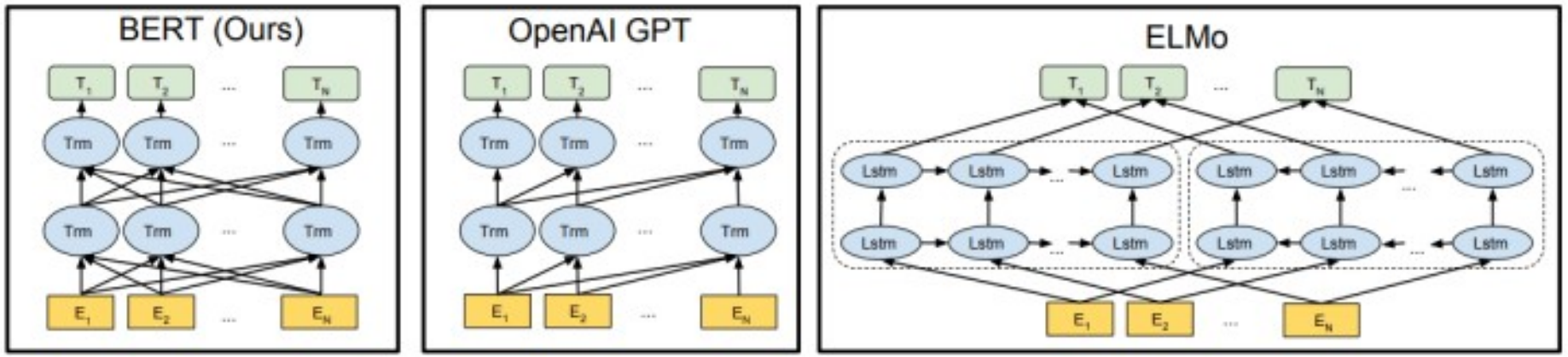
- Now enters BERT, a language model which is **bidirectionally trained** (this is also its key technical innovation). This means we can now have a deeper sense of language context and flow compared to the single-direction language models.

- Instead of predicting the next word in a sequence, BERT makes use of a novel technique called **Masked LM** (MLM)
- it randomly masks words in the sentence and then it tries to predict them.
- Masking means that the model looks in both directions and it uses the full context of the sentence, both left and right surroundings, in order to predict the masked word.
- Unlike the previous language models, it takes both the previous and next tokens into account at the **same time.**
- The existing combined left-to-right and right-to-left LSTM based models were missing this “same-time part”. (It might be more accurate to say that BERT is non-directional though.)

- Pre-trained language representations can either be ***context-free*** or ***context-based***. *Context-based* representations can then be ***unidirectional*** or ***bidirectional***. Context-free models like word2vec generate a single word embedding representation (a vector of numbers) for each word in the vocabulary.

- For example, the word “*bank*” would have the same context-free representation in “*bank account*” and “*bank of the river.*” On the other hand, context-based models generate a representation of each word that is based on the other words in the sentence.
- For example, in the sentence “*I accessed the bank account*,” a unidirectional contextual model would represent “*bank*” based on “*I accessed the*” but not “*account*.” However, BERT represents “*bank*” using both its previous and next context — “*I accessed the … account*” — starting from the very bottom of a deep neural network, making it deeply bidirectional.

BERT is based on the Transformer model architecture



- A Transformer works by performing a small, constant number of steps. In each step, it applies an attention mechanism to understand relationships between all words in a sentence, regardless of their respective position.
- For example, given the sentence, “I arrived at the bank after crossing the river”, to determine that the word “bank” refers to the shore of a river and not a financial institution, the Transformer can learn to immediately pay attention to the word “river” and make this decision in just one step.

SURPRISE QUIZ

What Next???

- 1D-CNN for text classification
- Sub-Word model
- Open AI's GPT
- Google's ALBERT
- ULMFiT
- Facebook's ROBERTa
- Text Summarization
- Transformer models for text summarization.

Sentiment Analysis using CNN

- https://www.kaggle.com/c/tweet-sentiment-extraction/overview
- https://github.com/rsreetech/TextClassificationTensorFlowCNN

- Load labelled dataset
- Apply pre-processing
- Convert into sequence
- Train the data
- Test and validate

Transformation of word

- Count the number of occurrences of every word in each sentence and provide those counts to the entire set of words in the dataset (called corpus in NLP) .
- Make a vocabulary where every word has its special index number. More formally, we can say classifying every word into its tied index.

Feature vector

example = ['analytics india magazine is good magazine ', 'analytics india magazine provides good information']

Next, we can vectorize the sentence using Countvectorizer.

Input:

```
from sklearn.feature_extraction.text import CountVectorizer
examplevectorizer = CountVectorizer()
examplevectorizer.fit(example)
examplevectorizer.vocabulary_
```

Output:

```
{'analytics': 0, 'good': 1,  'india': 2,  'information': 3,  'is': 4,  'magazine': 5, 'provides': 6}
```

Split the data

```
from sklearn.feature_extraction.text import CountVectorizer
 review_vectorizer = CountVectorizer()
 review_vectorizer.fit(review_train)
 Xlr_train = review_vectorizer.transform(review_train)
 Xlr_test  = review_vectorizer.transform(review_test)
 Xlr_train
```

- from sklearn.linear_model import LogisticRegression
- LRmodel = LogisticRegression()
- LRmodel.fit(Xlr_train, label_train)
- score = LRmodel.score(Xlr_test, label_test)
- print("Accuracy:", score)

Output

Accuracy: 0.8195050946142649

Word Embedding?

- Unlike what we have done with the Countvectorizer. It is a different way to preprocess the data. This embedding can map semantically similar words. It does not consider the text as a human language but maps the structure of sets of words used in the corpus.
- They aim to map words into a geometric space which is called an embedding space.
- If embedding finds a good relationship between works like for an example
- King – man + women = queen

- Keras provides a couple of methods for text preprocessing and sequence preprocessing. We can use them to make our data a better fit for the TextCNN model.

input:

```
from keras.preprocessing.text import Tokenizer
tokenizer = Tokenizer(num_words=5000)
tokenizer.fit_on_texts(review_train)
Xcnn_train = tokenizer.texts_to_sequences(review_train)
Xcnn_test = tokenizer.texts_to_sequences(review_test)
vocab_size = len(tokenizer.word_index) + 1
print(review_train[1])
print(Xcnn_train[1])
```

```
There was a warm feeling with the service and I felt like their guest for a special treat.
[43, 10, 4, 607, 323, 15, 1, 47, 2, 3, 350, 37, 109, 1908, 12, 4, 279, 1236]
```

Problem

- One problem is that in each sequence is the different length of words, and to specify the length of word sequence, we need to provide a mexlen parameter and to solve this, we need to use pad_sequence(), which simply pads the sequence of words with zeros

- from keras.preprocessing.sequence import pad_sequences
- maxlen = 100
- Xcnn_train = pad_sequences(Xcnn_train, padding='post', maxlen=maxlen)
- Xcnn_test = pad_sequences(Xcnn_test, padding='post', maxlen=maxlen)
- print(Xcnn_train[0, :40])

```
[ 278  295  212 1907   39  349    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0]
```

Text Summarization in NLP

- In Natural Language Processing, or NLP, Text Summarization refers to the process of using Deep Learning and Machine Learning models to synthesize large bodies of texts into their most important parts.
- Text Summarization can be applied to static, pre-existing texts, like research papers or news stories, or to audio or video streams, like a podcast or YouTube video, with the help of Speech-to-Text APIs.

Two forms of summary

- Some Text Summarization APIs provide a single summary for a text, regardless of length, while others break the summary down into shorter time stamps.

Text Summarization Methods

- Extractive and Abstractive.
- Extractive Text Summarization, where the model “extracts” the most important sentences from the original text, is the more traditional method.
- Extractive Text Summarization does not alter the original language used in the text.
- In contrast, Abstractive Text Summarization requires the model itself to generate the summaries, which may or may not include words and/or sentences from the original text.

With Extractive perspective

- we might consider the TextRank algorithm. Google uses an algorithm called PageRank in order to rank web pages in their search engine results. TextRank implements PageRank in a specialized way for Text Summarization, where the highest “ranking” sentences in the text are the ones that describe it the best.
- As before, we can extract the highest ranking sentences to yield a summary of the text.

- Beyond pure transformer models, there are even GAN-based methods to Text Summarization, which train a generator to create summaries and a discriminator to differentiate between real and generated summaries.
- Some methods even combine both extractive and abstractive elements into one framework

Types of API

1. AssemblyAI's Auto Chapters API

- AssemblyAI offers highly-accurate Speech-to-Text APIs and Audio Intelligence APIs. Its Auto Chapters API, part of its Audio Intelligence suite of APIs, applies Text Summarization on top of the data from an audio or video stream, and supplies both a one paragraph summary and single sentence headline for each chapter. This process is a unique adaptation of Text Summarization to AssemblyAI.
- The API is used by top product teams in podcasts, telephony, virtual meeting platforms, conversation intelligence platforms, and more

2. plnia's Text Summarization API

The plnia Text Summarization API generates summaries of static documents or other pre-existing bodies of text. In addition to Text Summarization, plnia also offers Sentiment Analysis, Keyword Extractor, Abusive Language Check, and more.

3. Microsoft Azure Text Summarization

As part of its Text Analytics suite, Azure's Text Summarization API offers extractive summarization for articles, papers, or documents. Requirements to get started include an Azure subscription and the Visual Studio IDE. Pricing to use the API is pay-as-you-go, though prices vary depending on usage and other desired features.

4. MeaningCloud's Automatic Summarization

MeaningCloud's Automatic Summarization API lets users summarize the meaning of any document by extracting the most relevant sentences and using these to build a synopsis. The API is multilingual, so users can use the API regardless of the language the text is in.

- **5. NLP Cloud Summarization API**

NLP Cloud offers several text understanding and NLP APIs, including Text Summarization, in addition to supporting fine-tuning and deploying of community AI models to boost accuracy further. Developers can also build their own custom models and train and deploy them into production

Generative Pre-trained Transformer

- *Generative Pre-trained Transformer 3* (**GPT-3**) is a language model that leverages deep learning to generate human-like text (output). Not only can it produce text, but it can also generate code, stories, poems.

- GPT-3 was introduced by Open AI earlier in May 2020 as a successor to their previous language model (LM) GPT-2

Language Models (LMs)?

- Simply put, language models are statistical tools to predict the next word(s) in a sequence. In other words, language models are probability distribution over a sequence of words.

Application

- Part of Speech (PoS) Tagging
- Machine Translation
- Text Classification
- Speech Recognition
- Information Retrieval
- News Article Generation
- Question Answering

OpenAI GPT-3 Architecture

Model Name	n_{params}	n_{layers}	d_{model}	n_{heads}	d_{head}	Batch Size	Learning Rate
GPT-3 Small	125M	12	768	12	64	0.5M	6.0×10^{-4}
GPT-3 Medium	350M	24	1024	16	64	0.5M	3.0×10^{-4}
GPT-3 Large	760M	24	1536	16	96	0.5M	2.5×10^{-4}
GPT-3 XL	1.3B	24	2048	24	128	1M	2.0×10^{-4}
GPT-3 2.7B	2.7B	32	2560	32	80	1M	1.6×10^{-4}
GPT-3 6.7B	6.7B	32	4096	32	128	2M	1.2×10^{-4}
GPT-3 13B	13.0B	40	5140	40	128	2M	1.0×10^{-4}
GPT-3 175B or "GPT-3"	175.0B	96	12288	96	128	3.2M	0.6×10^{-4}

Sizes, architectures, and learning hyper-parameters of the GPT-3 models

OpenAI GPT-3 family of models is based on the same transformer-based architecture of the GPT-2 model including the modified initialisation, pre-normalisation, reverse tokenisation, with the exception that it uses alternating dense and sparse attention patterns

Google ALBERT

- Google AI has open-sourced A Lite Bert (ALBERT), **a deep-learning natural language processing (NLP) model**, which uses 89% fewer parameters than the state-of-the-art BERT model, with little loss of accuracy. The model can also be scaled-up to achieve new state-of-the-art performance on NLP benchmarks

ULMFiT

- **Universal Language Model Fine-tuning**, or **ULMFiT**, is an architecture and transfer learning method that can be applied to NLP tasks. It involves a 3-layer AWD-LSTM architecture for its representations. The training consists of three steps: 1) general language model pre-training on a Wikipedia-based text, 2) fine-tuning the language model on a target task, and 3) fine-tuning the classifier on the target task.

Facebook RoBERTa

- Facebook AI open-sourced a new deep-learning natural-language processing (NLP) model, robustly-optimized BERT approach (RoBERTa). Based on Google's BERT pre-training model, RoBERTa includes additional pre-training improvements that achieve state-of-the-art results on several benchmarks, using only unlabeled text from the world-wide web, with minimal fine-tuning and no data augmentation.

Surprise Quiz 2

Subword

- Subword is **in between word and character**. It is not too fine-grained while able to handle unseen word and rare word. For example, we can split “subword” to “sub” and “word”. In other word we use two vector (i.e. “sub” and “word”) to represent “subword”.

Purely character-level models

- Why we need character-level models?

Because some languages don't have word segmentations, such as Chinese. Even for languages that have word segments, they segment words in different ways, such as German and Dutch contain many compounds. Besides, we need to handle the large, open vocabulary, such as online comments include lots of common informal spelling.

Advantage

- **In character-level models, word embeddings can be composed of character embeddings.** There are at least 3 benefits for character-level model**: 1). it can generate embeddings for unknown words; 2). similar spellings share similar embeddings; 3). it solves out of vocabulary (OOV) problem**