NLP and Generative Pre-trained Transformers

A brief introduction.

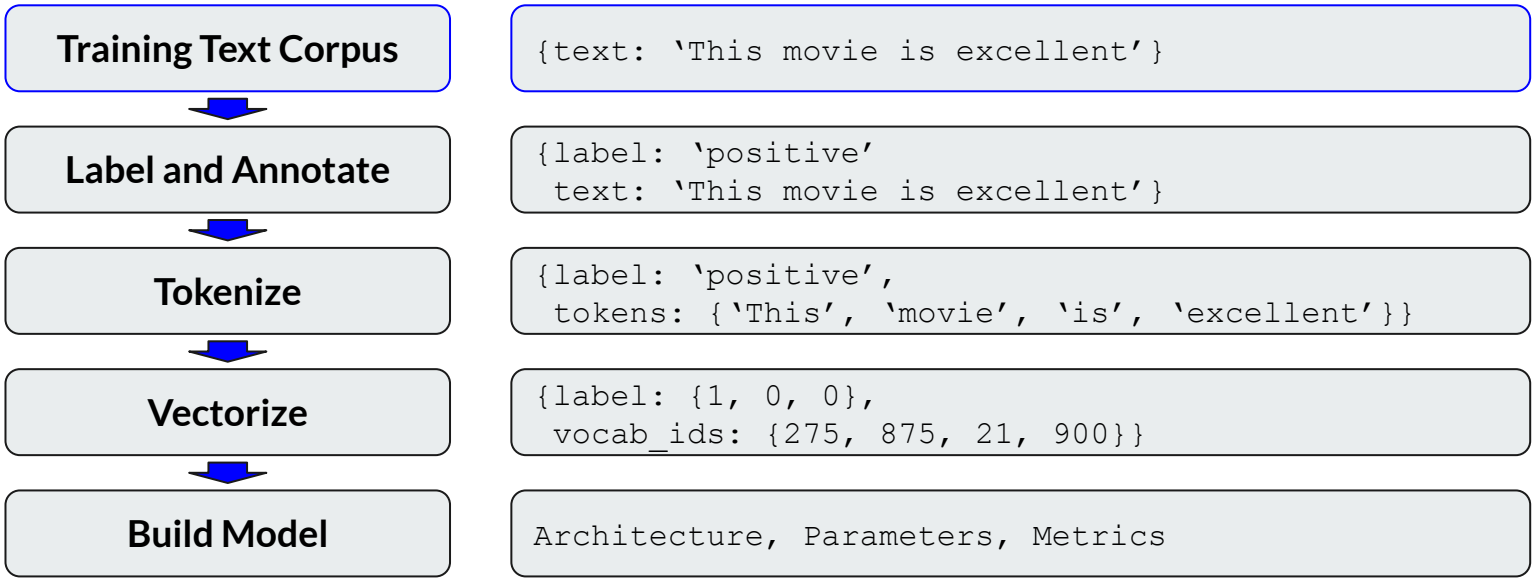
Notes from LinkedIn course by Kumaran Ponnambalam

Main Branches of NLP

NLU: Natural Language Understanding	To understand semantics, content and context. - Sentiment analysis - Summarization
Information Extraction	To extract structured information from text - Named entity recognition - Search
NLG: Natural Language Generation	To generate human-conversation-like text - Text to speech - Translation - Text Generation
ASR: Automatic Speech Recognition	To convert voice input into text - Speech to text - Trigger word detection

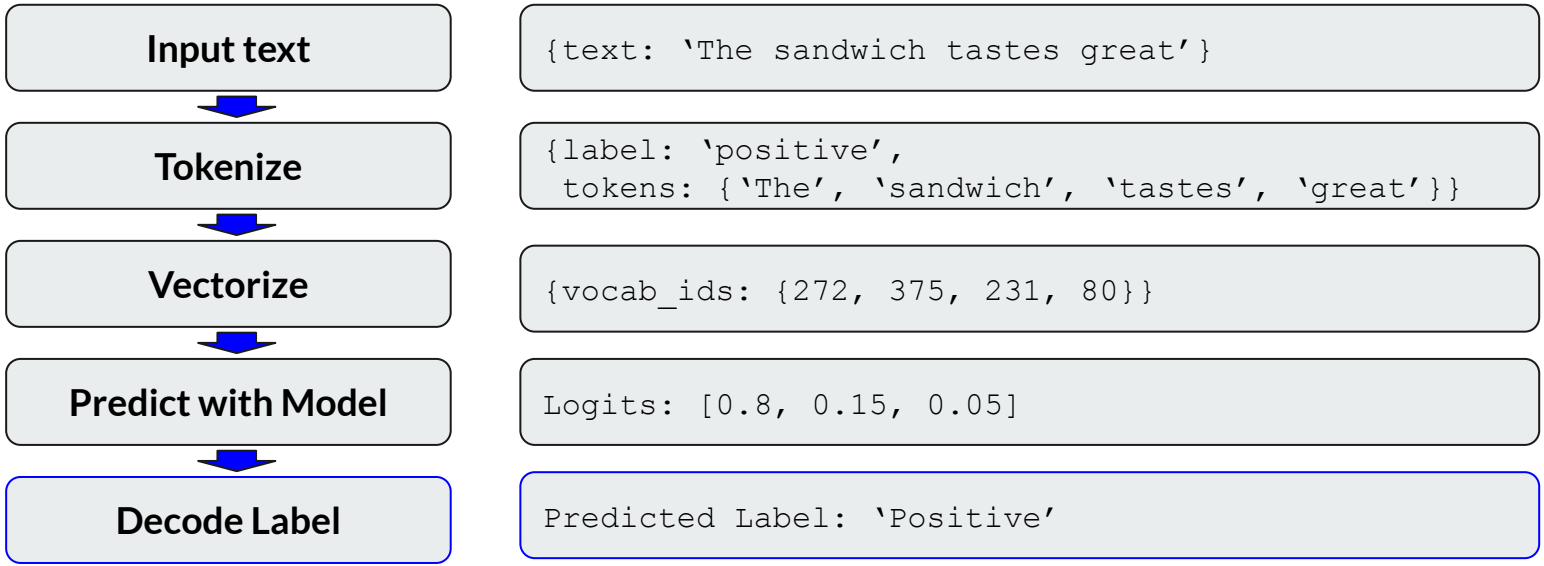
ML Process for NLP

1. Training for NLP



ML Process for NLP

1. Inference for NLP



Labeling for NLP

Data Labeling/Annotation is the process of adding tags/labels to training data that can then be used as targets for ML training

Ex: Review by John

“This is a great movie. The hero delivered his career’s best performance. A bit slow in the beginning, but great action in the second half. Would recommend to anyone who loves action”

Label Name	**Label Value**
Sentiment	Positive
Rating	4.5
Reviewer gender	Male
Movie type	Action

Labeling Resources

Expert Labeling	• High Accuracy levels • Limitations due to available time/experts
Crowd-sourcing	• Free labeling services from a large pool • Inaccuracies and inconsistencies
Third-party Annotators	• Professional and accurate • High costs
Programmatic Labeling	• Scaling and adaptability at low costs • Takes time to build an accurate program

Tokenization

Is the process of splitting a phrase, sentence or document into smaller units (word, subword, or character) called **tokens** and then using them as inputs for further text processing and analytics.

Tokenizers

- A program/algorithm/function that converts text to tokens
- Supports a vocabulary with each pre-defined token associated with an ID
- May be specific to a language or domain
- Pre-built tokenizers are available in popular framework and extensively used in NLP

Vectorization

Converting text data to equivalent numerical representation that can be consumed by ML algorithms

Available Techniques:

- **Bag of Words:** each unique tokens in a the vocabulary is a feature used for one-hot encoding
- **Term-Frequency Inverse Document Frequency (TDF-IDF):** each unique token is a feature providing weights values by frequency and uniqueness
- **Embedding Matrix**: uses words embedding to represent each token as an array of numeric values to capture relationships based on features

Pre built word embeddings are used in most cases.

Two popular open-source methods are **GloVe** and **Word2Vec**

Domain/Use-case-specific word embeddings are developed in special cases

Transformer

An advanced neural network that is capable of modeling complex content and meaning in text and hence can be used in a variety of NLP use cases and applications.

Advantages

- Built for parallel processing (can overcome sequential processing constraints of RNNs)
- **Capture relationships and context between tokens**
- **Pay attention to all tokens in a sequence**
- **Help build general-purpose foundational models/language models**
- **Adapt pre-trained transformers for a variety of use cases**

Transformers' architecture

1. Positional Encoding

The process of deriving a vector for each token in a sentence to represent its position in a sentence

- Recurrent neural networks (RNNs) capture positional information by processing one token at the time, in sequence, to build a hidden state
- Transformers process all tokens in the input in parallel - hence, additional positional context is needed
- The positional encoding vector is of the same dimension as the embedding vector

Attention

- Tokens in a sequence have semantic relationships
- The attention mechanism models relationships with parallel execution capabilities
- Is an alternative to the hidden states in RNN

Encoder-Decoder

The Encoder

The encoder converts a given set of tokens in a sentence into equivalent vectors, called hidden states or context. These vectors capture the semantics and relationships between the tokens in the sentence.

It makes use of multiple techniques (positional encoding, embedding matrix, attention)

The Dercoder

The decoder uses the encoder's hidden states to iteratively generate a sequence of output tokens, one at the time. The form of output depends on the use case (ex: classification, translation, etc).

Tansformer training

- Create a transformer model architecture
- Initialize weights and other parameters
- Pass training data to the encoder-decoder and predict the output
- Compute loss (prediction - true label) and cost
- Update weights and continue until reaching the expected accuracy levels
- Save model

Transformer predictions

- Load the saved model
- Pass the input (for inference) to the encoder-decoder pipeline
- Predict hidden states, softmax, and tokens based on the use case

Language models

A **language model** is an ML model that represents language in a general or specific context. It can understand words, meanings, and flow and can predict the next word based on a previous sequence.

Features

- Trained on a large corpus of data for a generic or specific domain
- Predict next words based on previous words
- Relate words with similar meaning
- Understand sentence context based on words and their sequence
- Can be used as foundational models for a large variety of use case (sentiment analysis, question answering, translation)

Pre-trained Transformer Models

Given that most language models are very large and resource-intensive, building tools for specific use cases can be more easily achieved through the use of pre-trained models.

Pre-trained Transformer Models are general purpose transformer architecture-based models, built using large corpora of data. They can then be reused to solve a variety of NLP tasks.

What are the features of a pre-trained model?

- Has a specific architecture with trained parameters
- Uses tasks like masked language modeling (MSM) to create the model, eliminating the need for labels
- Creates checkpoints: specific versions of a model, trained on specific datasets

Transfer Learning

In transfer learning general models serves as foundational model for

- Uses an existing model built for a given task as a starting point for a new task or use case
- Starts with a predefined architecture with pre-trained weights
- Can freezes or updates existing layers during training
- Can add new layers that are specific to the task
- Uses a small corpus of context-specific training data for additional learning

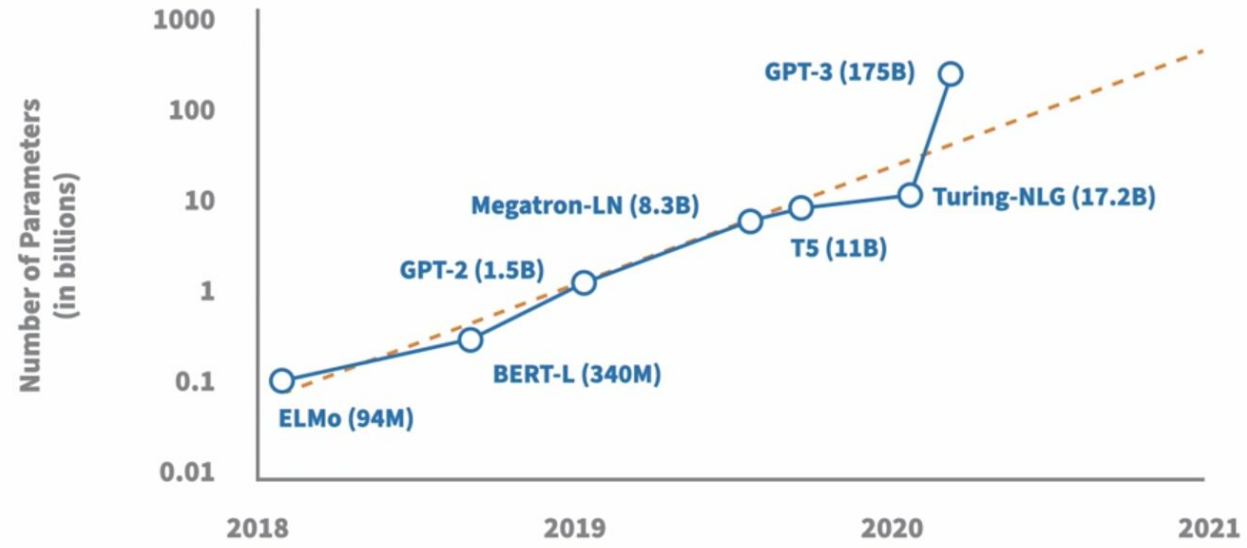
Generally a transformer model is used as foundational model for a specific NLP application.

Popular Models for Transfer Learning

Some of the most widely used pretrained models for transfer learning include:

- **BERT** (Bidirectional Encoder Representations from TRansformers) is a encoder-only transformer created by Google. It uses more than 345 million parameters. Many variants have been created, including **BERT Large, DistilBERT, RoBERTa, ALBERT, DeBERTa**
- **GPT** (Generative Pre-trained Transformer), a decoder-only transformer created by Open AI. It can generate new text based on an initial prompt. It is popular for natural language tasks like question answering. Early versions GPT and GPT2 are open source. GPT-3 (with more than 175 billion parameters) and GPT-4 (1. 76 trillion parameters) are not open source, available via a paid API.
- **T5** (Text-to-Text Tranfer Transformer), an encoder-decoder model created by Google. It is a single pretrained model that can be trained for multiple tasks like summarization, question answering and classification. The input prefix provides the hint for the type of task and no transfer learning is needed in most cases.

Transformers history and evolution



Large Language Models Comparison

Date	Model	What we learned	Param.	Training tokens	Creator
May 20	**GPT-3**	Few-shot learning with large model	175B	300B	OpenAI
Dec 21	**GLam**	Reduce trainning/inference costs using a sparse mixture-of-experts model	1.2T	280B+	Google
Jan 22	**MT-NLG**	Large model with parallelism across compute and memory	530B	270B	Microsoft/ Nvidia
Jan 22	**Gopher**	Model performance across a range of model sizes and tasks. In general, larger models perform better on logical and mathematical reasoning tasks	280B	300B	DeepMind
Apr 22	**Chinchilla**	Current large language models are significantly undertrained. Given a flop budget, increasing model size is less efficient than increasing the training data	70B	1.4T	DeepMind
Apr 22	**PaLM**	Increased model flops utilization	540B	780B	Google

Hugging Face

An open-source community for sharing model artifacts. It features:

- Pre-trained and packaged models for popular ML tasks
- Preformatted and labeled datasets
- A library and an API for easy training andinference
- Additional building blocks like tokenizers, metrics, etc.
- The public can contribute new/refined artifacts
- API usage is charged but artifacts and Python packages are free

https://huggingface.co/docs

Pre-trained models in Hugging Face

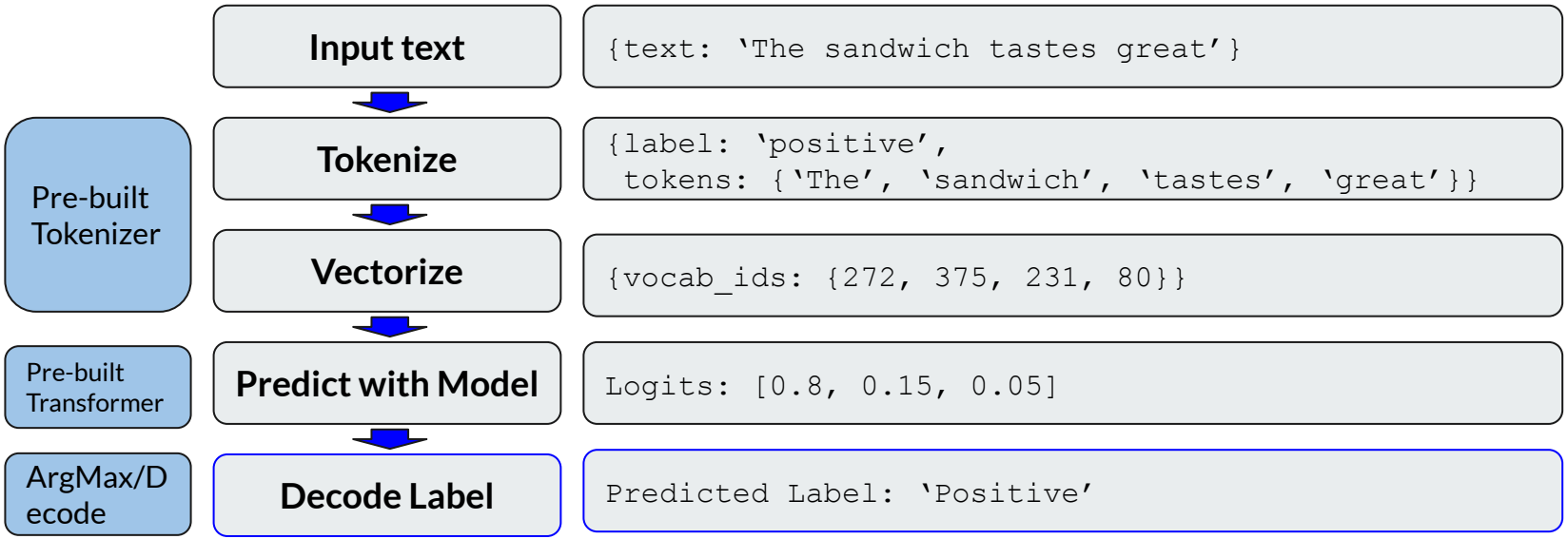
Hugging Face hosts a large selection of pre-trained models called **checkpoints.**

- Searchable by task, ML framework and architecture
- Each model features a model card providing background and documentation
- Mostly focused on Pythorch, but models are available also for TensorFlow and other libraries
- An hosted inference API is available to run a quick example
- Datasets are available (often in Apache Arrow format) for download, caching and training

https://huggingface.co/models

Inference with Hugging Face

Example of inference model:



Hugging Face Pipeline API

It combines all inference operations under one method call

- Tokenization
- Prediction
- Decoding

Predictions can be achieved with a single line of code

Custom tokeinzers and model checkpoints are provided

Training with Hugging Face

- Use transfer learning to customize a model for specific use cases
- A small use-case-specific dataset can be used for training
- It is possible to freeze the encoder-decoder weights of the original checkpoint and only train the classifier
- Artifacts like tokenizers, models and datasets from HF can be reused

Sentiment Analysis with Hugging Face

- **Predict the sentiment of a body of text**
  - Positive, neutral, negative
  - Emotions (advanced)
- **Popular use case in NLP**
- **Multiple pretrained model checkpoints available for use**
- **Predefined Hugging Face pipeline task**

Named Entity Recognition (NER)

- **Used to extract relevant entities from the body of a text, such as names or dates**
- **Uses semantics to identify entities**
- **Helps build automated language understanding systems**

  (ex. customer support chatbots/voice bots, automated email understanding and response, text analytics)

Entity Types

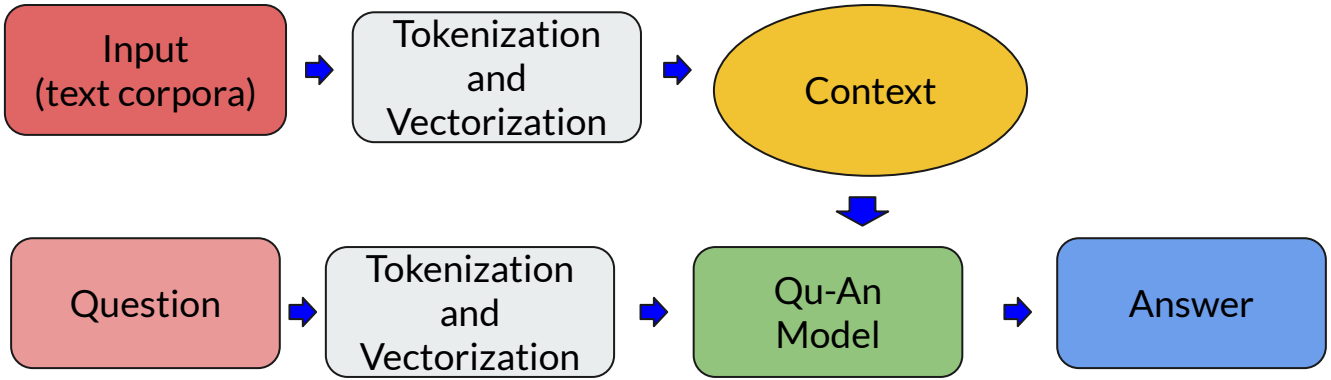
Common entity types include:

- PER - person
- LOC - location
- ORG - organization
- DATE - date/time
- MISC - miscellaneous

Transfer learning NER models allow you to create custom entity types for specific use cases

Question Answering (Qu-An) in NLP

- Question Answering is a popular NLP task that has many self-service applications
- It works based on a “context” (a body of text to search for answers)
- It extracts a subset of text/sentence from the context that answers the question



Open vs Close Domain QA

Open Domain (QuAn (ODQA):

- Answer questions regarding any general topic: for example. Used in search engines
- Built on a large text corpus, ex: Wikipedia
- Takes enormous resources and time to build corpus and train model
- Large model sizes
- Accuracy level may be low

Close Domain Qu-An (CDQA)

- Trained on a limited set of text, specific to a domain: for example, drug facts
- Smaller models and lower resources requirements
- Higher levels of accuracy easier as the domain is restricted
- Multiple enterprise use cases: FAQ-based answering, product brochures, queries, etc

SQuAD

The **Stanford Question Answering Data Set (SQuAD)** is a popular dataset for question answering

- It uses a scoring function to measure accuracy of answers (a collection of performance metrics)
- For each question, a ground truth or correct answer is needed for evaluation
- The algorithm compares the correct answer to the the predicted one to measure performance

SQuAD Metrics in Hugging Face

- The Metrics package in Hugging Face provides implementation of many evaluation metrics, including SQuAD
- It helps evaluate the performance of a pretrained model in a specific use case
- It requires an evaluation dataset (Context, Question, Correct Answer)

Text Summarization in NLP

Text Summarization creates a summary text out of larger input text. There are two main types:

- Extractive summarization
  - It extracts a subset of sentences form the input text
  - The resulting sentence/text in the summary are verbatim

- Abstractive summarization
  - It creates a human-like summary based on the input text
  - The summary text is not verbatim, but can capture the gist of the input text

The BART Model

- BART stands for Bidirectional Auto-Regressive Transformer; it was created by Facebook
- Is a transformer architecture that uses an encoder-decoder structure
- The encoder part is similar to BERT, while the decoder part is similar to GPT
- Effective for text comprehension and text generation tasks
- It’s used for text summarization, question answering and text generation tasks

ROUGE Metrics

- Evaluating summary performance is difficult and requires specific tools
- ROUGE (Recall-Oriented Understudy for Gisting Evaluation) is a set of metrics for summary evaluation that can measure the similarity between the generated summary and the original input text
- **Rouge-1** is a score that measures the unigram overlapping
- **Rouge-2** is a score that measures bigram overlapping
- **Rouge-L** is a score based on the longest common subsequence

Natural Language Generation (NLG)

- Can produce synthetic text based on a given input/context
- Should be indistinguishable from human-generated text
- Can produce short (a single phrase) or long (multiple paragraphs) texts
- Popular architectures include GPT-X, BART, T5

Use cases:

- Content creation
- Conversational AI (chatbots/voice bots)
- Language translation
- Personalized e-mails/responses
- Report generation based on structured data
- Product description based on specifications

Conversation Generation (ChatBots)

- ChatBots can generate responses based on user input, answer question and ask follow-up questions
- They can maintain context/history of the conversation
- Real uses cases may include additional data with conversation
- Beside NLG, additional services may be used like Named Entity Recognition, Sentiment Analysis, Question Answering, etc.

Machine Translation

- Machine Translation can convert text from one language to another
- It produces human-like conversation, not word-to word translation
- GPT-3 and T5 are popular architecture for this task
- There are several challenges with the support of multiple languages
- Smaller custom models for use-case specific applications tend to perform better

Applications include:

- Speech translation for customers
- Document translation for enterprise
- Multilingual customer service (without language-trained agents)
- Multilingual text analytics (like reviews)

Training a custom model with Hugging Face

- Use transfer learning to customize a model for specific use cases
- A small use-case-specific dataset can be used for training
- It is possible to freeze the encoder-decoder weights and only train the classifier/Seq2Seq
- Artifacts like tokenizers, models and datasets from HF can be reused

Inference challenges with transformers

- In most cases the model needs to be customized with transfer learning in order to improve use-case specific performance
- Large memory requirements
- Compute resource requirements
- Latency could be critical for certain use cases
- Scaling concurrent requests can pose further challenges

Customizing Pretrained Models

When to customize?

Reasons can include:

- Low performance for use-case specific samples
- Custom/special vocabulary
- Variations in content structure or grammar
- The model’s size is huge

Good practices for Transfer Learning with Custom Datasets

- Small dataset sizes
- High quality datasets
- Iterative model architecture evolution
- Thorough validation and testing
- Model monitoring for drift and bias
- Constant model retraining with new data

Model Compression

- What to reduce in compression?
  - Number of layers, nodes, or parameters
  - Storage needs (ex: INT vs FLOAT16 vs FLOAT32)

- Benefits of compression:
  - Reduced number of math operations required
  - Lower memory footprint

- Things to keep in mind when compressing a model:
  - Ensure that the model continues to perform at desired accuracy levels for the specific use case
  - Use simple techniques like quantization and pruning first, before trying complex ones

Model Compression techniques

- Quantization
- Pruning
- Low Rank Approximation
- Knowledge Distillation
- Neural Architecture Search

Multiple Models for One Solution

An ML solution may combine multiple models to provide end-to-end user experience. Overall solution design and deployment should be optimized across models.

Popular model deployment configurations include:

- **Chained Models.** Models are chained in a sequence where the output oa model becomes the input of the following model. Ex; chatbots
- Single Input Independent Models. Models are independent of each other and all use the same input, each model producing its own output that contribute to the final joint output
- Alternate Models. The same input goes to all models, then one of the models available is chosen for inference based on a given condition. Ex: language-detecting translation

Multiple Models Best Practices

- Test models together as a solution
- Parallelize as much as possible in deployment
- Validate intermediate inputs and outputs
- Exit chains early if possible
- Scale each model independently
- Perform end-to-end latency measurements