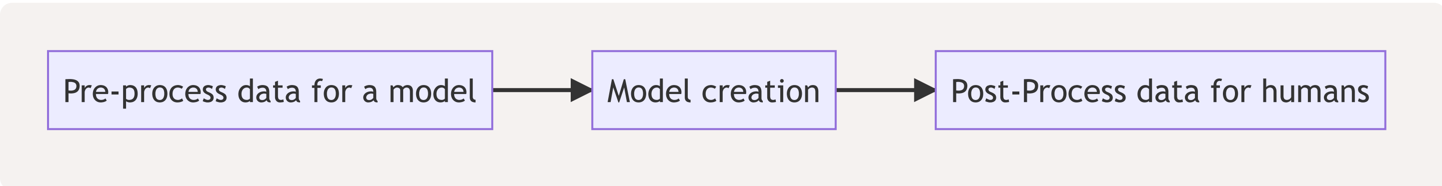
Hugging Face NLP course.

https://huggingface.co/course/chapter1/2?fw=pt

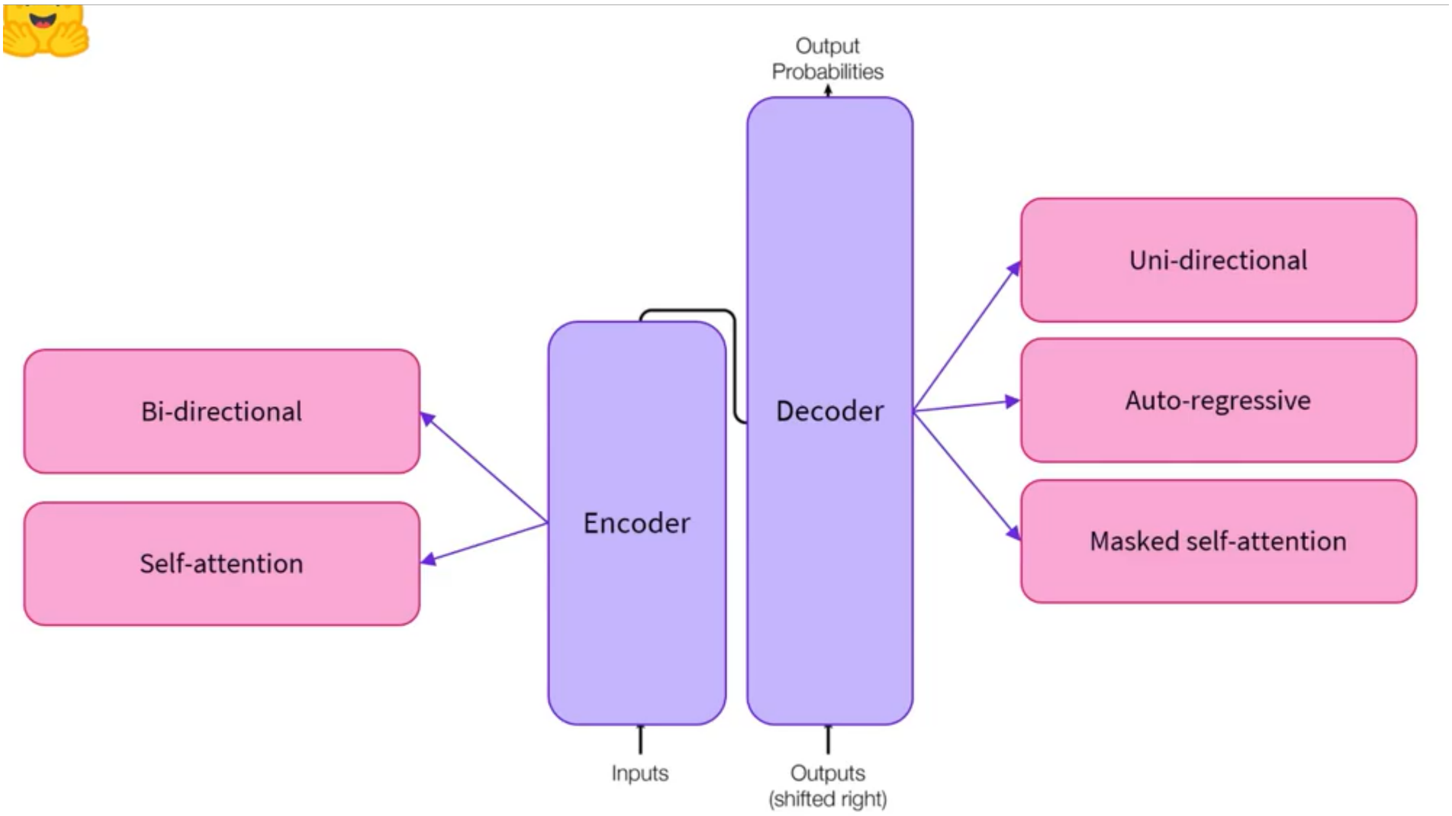
1. Transformer models

- NLP - Natural language processing
  - NLP = ML ∩ linguistics.
  - Tasks: language translation, summarizing text, filling removed words, generating new text, etc.
  - Challanging because we need to model text somehow.
- Possibilities of Transformer models:
  - Summary
  - Translation from one lang to another
  - Classify sentiments
  - Classify area
  - Classify words
  - Continuing, generating text
  - Fill in masked
  - other
- Pipeline function: final object for a task (sentiment, zero-shot)

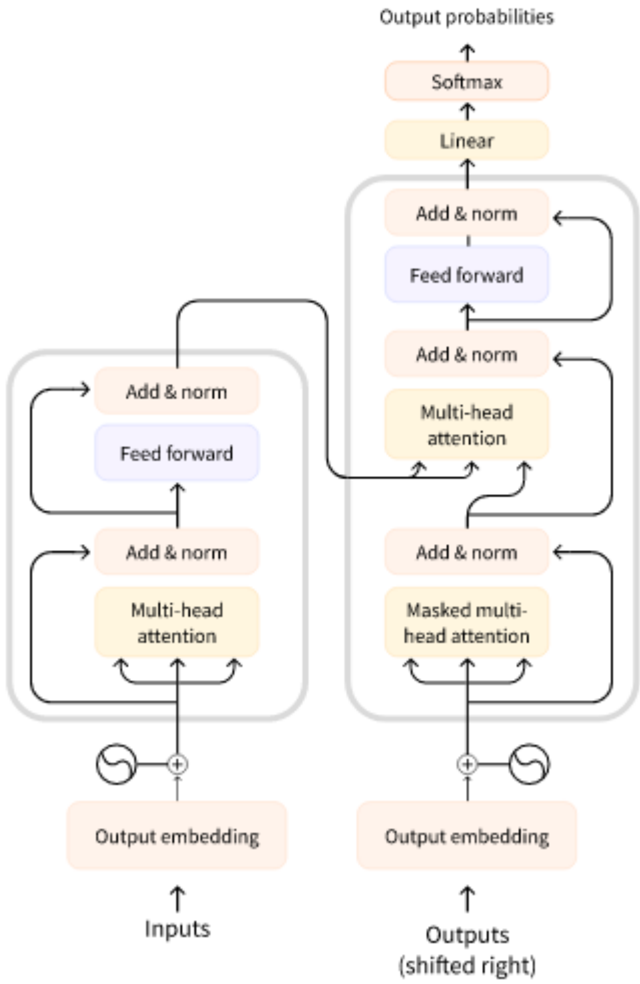


  - Uses diff models (gpt2, openai-gpt, etc.)
- Tasks: 'question-answering', 'summarization', 'translation', 'ner' (name entry recognition, classify words), 'zero-shot' (classify when no labels at train)
  - 'Zero-shot' - no need for fine-tuning for tasks.
- How Transformers work?
  - Categories of NLP models:
    - GPT-like. Auto-regressive.
    - BERT-like. Auto-encoding.
    - BART/T5 like. Seq-to-seq.
  - First they're trained on language model (not useful in practice) then they're fined tued for a specific task (masked text, causual language model(next word)) by using labled data.

- They are big. Billions of params.
    - A lot of CO2 emmisions like a car in its lifetime but depends on carbon footprint. That's why use fine-tuning, transfer learning, also use random hyperparams search. To measure - https://codecarbon.io/, https://mlco2.github.io/impact/.
- Transfer learning
    - Using other trained mode for other task - fine tuning. Why? 1) Better performance, 2) less data and 3) less resources (time, etc) for new task.
- Transformer Architecture



    - Parts:
        - Encoder - gets text input and generates 'understanding' of it.
        - Decoder - transforms input into another output.
    - 3 architecture types: encoder only, decoder only, encoder and decoder (seq 2 seq).
    - Attention layer. The main layer as it attends to specific context to build lanugage model, i.e. each word has not only its meaning but also depends on the surrounding context.
    - Original architecture:



        - Used for translation.
        - Encoder attends all text (left and right) so can translater.
        - Decoder attends only left output text till current i-th word and all input text.

    - *Checkpoint* (like BERT_base) means a particular set of weights used to train *architecture* (BERT) while *model* can mean both.

- Encoder models

    - AKA *bi-directional*, *auto-encoding* models as they access all the input. Examples: BERT, ALBERT, etc.
    - Good to extract meaningful info, classify, etc
        - MLM - guessing a randomly masked word.
        - Sentiment analysis

- Decoder models. AKA *auto-regressive models* (use previous output as input).

    - Attends only the words before current one. They mask the input, masking.
    - To generate text. NLG - natural lang generation. Causal Language Model.
    - Examples: GPT-2, etc.

- *Sequence-to-sequence* models. AKA encoder-decoder models.

    - Trains decoder and encoder.

    - To generate text based on other text
        - summary,
        - translation (encoder in English, and decoder in French, e.g.)
    - They don't share weights.
    - Examples: BART, T5.

- Bias and limitations. Can generate sexist, homophobic, etc. text as trained on data from all over the internet.

See quiz at https://huggingface.co/course/chapter1/10?fw=pt

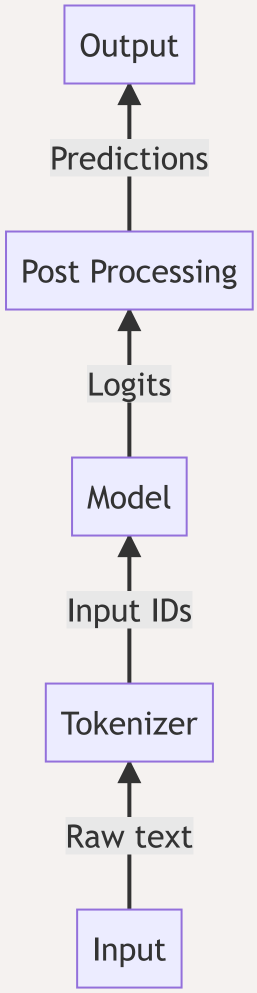
Questions:

- What's NLP?
- What Transformer models can do?
- Three types of Transformers.
- What's transfer learning? Fine-tuning?
- Original architecture from All you need is attention paper.
- Encoder models. Their main props?
- Decoder models. Their main props?
- Seq2seq modesl. Their main props?

2. Using Transformers

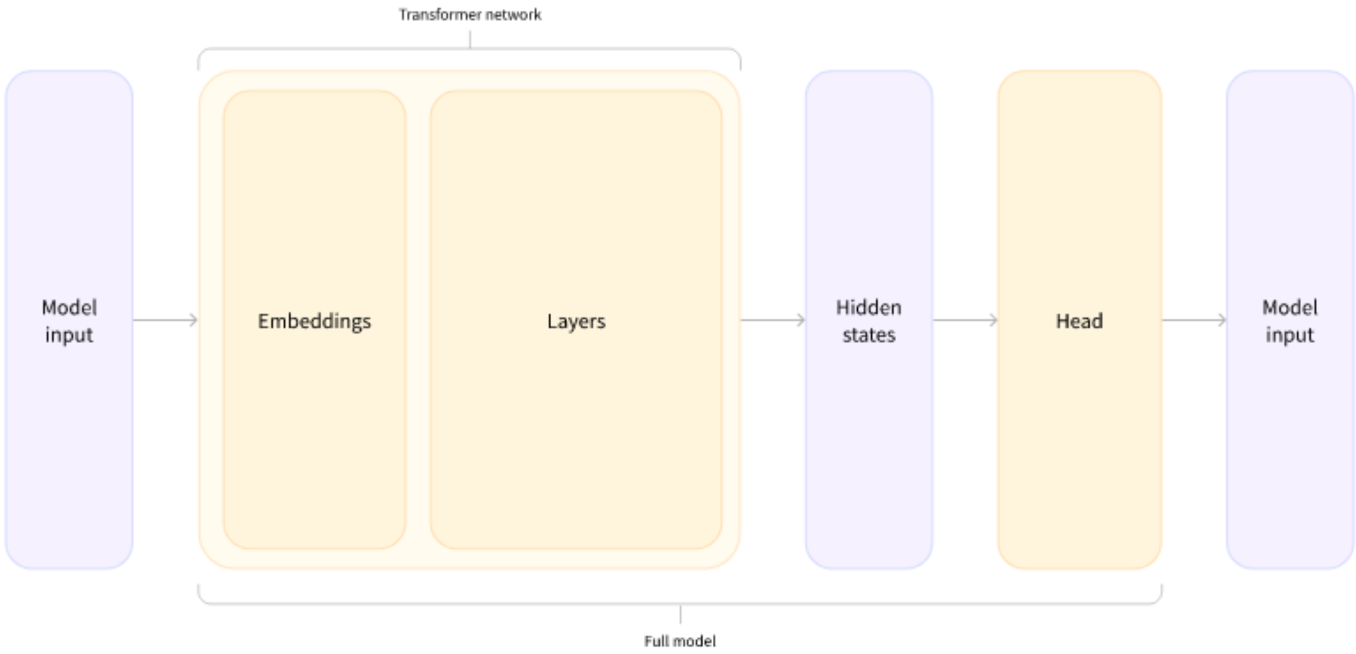
- The Tranformers lib to handle the zoo of many NLP models.

- Pipeline func



`transformers.AutoTokenizer` and its `from_pretrained(checkpoint)` to load tokens of a model.
Logits: it is from a model. Those are not predictions (it is not normalized).
Predicions: after softmax, those are probabilities.

- Tokenizer: splits words, maps those words (now tokens) to integers.
- Embeddings - to convert token numbers into vectors
- Head - receives hidden layers input, to specialize in tasks. Hidden state - how model understands input; then hidden state passed to the head to perform a task.
- logits - un-normalized predictions (before softmax)



- Models.
  - How to create a model? Use `transformers.<Model>.from_pretrained(..)`
  - How to configure model? Use `transformers.<Model>Config`
  - How to save model? Use `save_pretrained`
  - How to use model? Use `transformers.<Model>(<model config>)(<model_inputs>)`
- Tokenizers
  - Models needs number not words. So translate those to numbers. How? Depends on models.k
  - How:
    - Word-based. By spaces or by punctuation. Use a vocab to map to tokens. Limitted by vocab (10K or 500K), unknown words.
    - Character based.
      - Less size of vocab
      - Fewer unknown chars then.
      - How punctuation?
    - Subword tokenizer.
      - To fix issues in word-based and char based tokenizers.
      - Break into meaningful subwords if larger word.
    - Other: byte-level, wordpiece, other.
  - Loading / saveing
    - `AutoModel.from_pretrained(<checkpoing>)`
    - `transformers.<Model>Tokenizer` , etc.
  - Encoding.
    - How we make numbers from tokens: text to numbers.
    - It is `ids = tokenizer.convert_tokens_to_ids(tokens)`
  - Decoder.

        - Reverse process. Use `decoded_string = tokenizer:decode(<Vector here>)`

- Multiple sentences.

    - Batching for parallel proc.
        - Models expect batche.
    - If diff length? Padding (with '0') or truncate.
    - Attention layer should get attention mask to avoid this padding.
    - Long sentences? Truncates. Typically - 512 or 1024.

- All together.

3. Fine-Tuning a pretraine model.

- This section is about how to load, preprocess a dataset used for our specific task. Then how to fine-tune our model for this dataset, how to train it.
- Processing the data. This is from a row text to batches. So that we can fine-tune our model for a task.
    - Loading datasets fom the Hub: Use `DatasetDict` from `datasets.load_dataset()` to download datasets from th eHub.
    - Preprocess: use `transformers.AutoTokenizer`. Check `attention_mask`. Use tokenizer of a checkpoint.
    - Dynamic padding. This makes the same size batches.
- Fine-tuning a model with the Trainer API or Keras.k
    - Use `transformers.Trainer` when have tokenizer, datasets (valid, train). Then `.train()`. Also give it an evaluation function as below.
    - Evaluation and metrics (to give it into `Trainer()`)
        - Use `trainer.predict()` to get predictions.
        - Use `evaluate` lib to evaluate accuracy andother.
- Full training
    - (This repeats the steps above in one run)
    - Then we run training loop: batch, backward loss, learning rate, etc.
        - optimizations: AdamW, learning rate decay, etc.
    - Evaluation loop: on batches, then accumulate.
    - Speed up with Accelerate: trains multiple batches simultanuously on GPU.

...

...

...

...

...

...

...

...

...

...

...

...

...