Reading: Getting Started with Advanced Concepts of Transformer Models

Estimated time needed: 7 mins

Overview

This reading covers the three common transformer families and related concepts: decoder-only (e.g., GPT), encoder-only (e.g., BERT), and encoder–decoder (seq2seq, e.g., translation). You'll learn when to use each family, how attention and positional encodings work, and which masking strategies are used in training.

By the end of this reading, you will clearly understand when to use decoder, encoder, or encoder–decoder models, and how each powers real-world applications.

- **Decoder models (GPT-like):** Best for generating text (chatbots, story writing, autocomplete).
- **Encoder models (BERT-like):** Best for understanding tasks (search engines, sentiment analysis, question answering).
- **Encoder-Decoder models**: Best for translation and tasks where input and output sequence mapping is required.

Throughout this module, you'll see how **attention mechanisms** (queries (Q), keys (K), and values (V)) help models capture relationships between words, how **positional encoding** ensures order is preserved, and how **masking strategies** guide learning. Hands-on labs give you practical experience in building and training models like GPT and BERT, as well as preparing data and experimenting with translation tasks.

Decoder models (like GPT)

Imagine you're texting, and your phone suggests the next word, for example, you type, 'How are ___' and the system prompts (you). This is a typical application of decoder-only GPT models. Key training components for decoder models include **causal masking** and **loss computation**.
The model is trained to always guess the next word based only on what you already typed last. This is enforced by causal masking, which hides **future** tokens. You can build a decoder from scratch in PyTorch or leverage Hugging Face's GPT2LMHeadModel. With Hugging Face's Trainer, you fine-tune the model. For inference, Hugging Face's pipeline **text-generation** lets you generate text easily.

Encoder-only models (like BERT)

Unlike GPT, BERT captures context from both directions. Pretraining tasks in BERT include **Masked Language Modeling (MLM)** and **Next Sentence Prediction (NSP)**, along with data preparation strategies in PyTorch. These models excel at understanding text, powering applications like search, sentiment analysis, and question answering.

- **MLM (Masked Language Modeling):** Randomly mask tokens and train the model to recover them, forcing bidirectional understanding of context.
- **NSP (Next Sentence Prediction):** Teach the model to detect if sentence B follows sentence A — Useful for understanding relationships between sentences

To train BERT in PyTorch, you prepare inputs (tokens, segments, masks) so the model can do both tasks. Data preparation strategies in PyTorch includes

- Tokenization → Splits raw text into subword tokens with [CLS]/[SEP].
- Segment IDs → Tell BERT if a token is from sentence A or B.
- Attention Masks → Tell BERT which tokens are real, and which are just padding. Ensures BERT only attends to meaningful tokens.

Encoder–Decoder models (translation)

These combine both strengths, with the encoder reading the input sequence fully and the decoder generating the output step by step. You'll implement translation models in PyTorch, practice hands-on labs, and understand how cross-attention bridges the input and output sequences.

Hands-on labs

Lab 1: Decoder causal language models (GPT-like)

This lab is all about text generation using a decoder-only transformer—the GPT family of models. It demonstrates how transformers can produce language and explains why GPT is so powerful for creative tasks like writing, summarizing, or answering prompts.

The focus is on **causal language modeling**: predicting the next word given the previous ones. You start with a dataset (IMDB reviews in this lab), break text into tokens, and convert them into numbers (a vocabulary). To train a GPT-like model, you must ensure that when predicting the next token, the model doesn't **cheat** by looking ahead. That's why causal masking is introduced—each word can only attend to the ones before it, not after.

Another key concept is **positional encoding.** Unlike RNNs, transformers process all words at once, so they need a way to know the order. GPT can actually learn this order through trainable positional embeddings (this is different from the sinusoidal ones you'll see later in BERT/translation).

Finally, you build a **small GPT model**. After **training**, you try **inference**—feeding a prompt and letting the model generate text one token at a time. This introduces the idea of autoregressive text generation.

Lab 2: Pretraining BERT models (encoder-Only)

This BERT lab introduces the encoder side of transformers. Unlike GPT, BERT is about **reading** and **understanding** and doesn't generate text step by step. Instead, it looks at the entire sentence at once (bidirectionally) and learns to deeply understand context.

To train BERT, you use two pretraining tasks:

- **Masked Language Modeling (MLM)**: Hide some words and ask the model to guess them. This forces the model to use context from both left and right—for example, in 'The dog is chasing the ___', BERT must predict 'cat'.
- **Next Sentence Prediction (NSP):** Give the model two sentences and ask if the second one logically follows the first. This helps it understand relationships between sentences, useful in tasks like Q&A or summarization.

Segment embeddings: Since BERT often works with pairs of sentences, the segment embeddings helps to mark 'this token belongs to sentence A' and 'this one to sentence B.' Training shows how BERT learns representations useful for understanding language, not just generating it. By the end, you can test the model on sentence-pair tasks or masked word prediction.

Lab 3: Transformer model for language translation (encoder–decoder)

In this lab, you combine both worlds: an **encoder** (like BERT) + a **decoder** (like GPT) into a full **seq2seq transformer**. Like a human translator, you read the whole German sentence, then carefully produce English words one by one, always checking the original.

This lab demonstrates how transformers can map one sequence to another, not just generate or understand within the same language. Applications go beyond translation—summarization, dialogue, and even code-to-text work the same way.

- **Encoder-Decoder architecture:**
  - Encoder → Reads full source sentence (German).
  - Decoder → Generates target sentence word by word (English). Decoder looks at both the past words it generated AND the full source sentence.
  - To train BERT in PyTorch, you prepare inputs (tokens, segments, masks) so the model can do both tasks.
  - We compare prediction versus actual word → Compute 'loss' → Adjust model.

Author(s)

Shilpa Giridhar