

HIT391

MACHINE LEARNING: ADVANCEMENTS AND APPLICATIONS

- Lecturer: Dr. Yan Zhang
- Email: yan.zhang@cdu.edu.au





Week 11:

Large Language Models (LLMs)

- **Learning Outcomes**

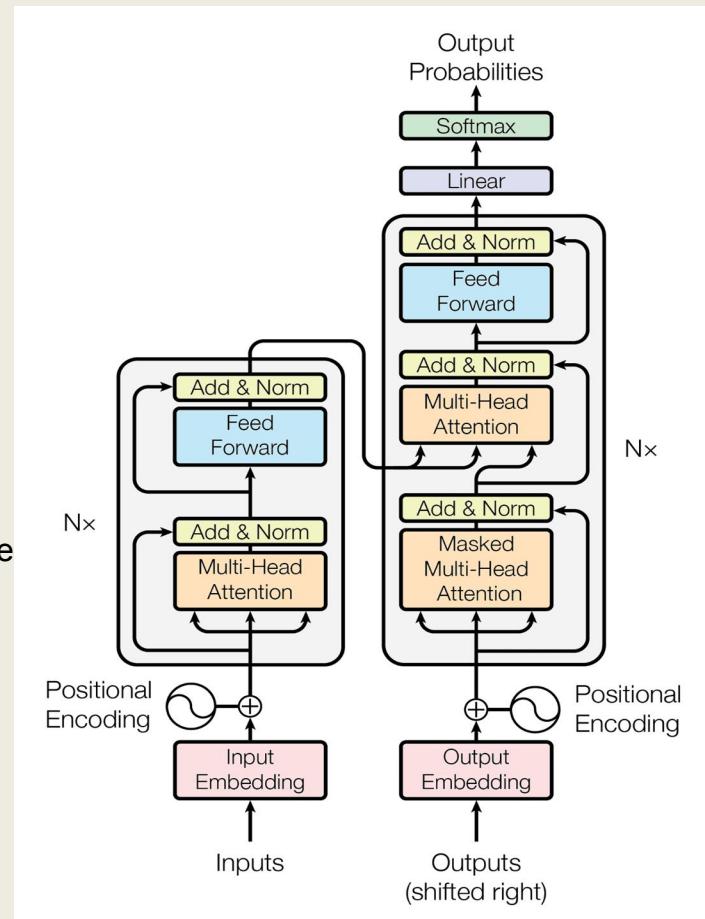
- What are LLMs?
- Transformer: Self-attention, Auto-regression training process
- LLMs: General model, Prompts
- LLMs: Training Steps - Pre-training, Fine-tuning, RLHF
- BERT, GPT
- Finetune LLMs for Specific Applications?

What are LLMs?

- Language models that have many parameters (over 1B) and can perform multiple tasks through prompting
- Eg. GPT, Llama2, Gemini, PaLM, Mistral, Mixtral etc.

Transformers: The Backbone of LLMs

- Introduced in "Attention Is All You Need"
(Vaswani et al., 2017).
- Parallelism
 - Unlike RNNs, Transformers process all tokens in a sequence at the same time, not step by step.
 - This makes training on GPUs much faster and more **scalable**.
- Self-Attention
 - Every token attends to every other token — capturing long-range dependencies efficiently, understanding word relationships.
 - This allows deep understanding of context across entire texts - LLMs to **handle many NLP tasks**
- Example: language translation from English to Chinese
 - Input (English): "How was your weekend?"
 - Target (Chinese): "你周末过得怎么样? "



Transformers: The Backbone of LLMs

- Transformer is composed of: Encoder-Decoder Structure (for seq2seq tasks)
 - Encoder: Processes input (e.g., source sentence)
 - Decoder: Generates output (e.g., translated sentence)
- LLMs like GPT use only the decoder, while T5/BART use both encoder and decoder.

Component	Function
Multi-Head Self-Attention	Lets each token attend to others (learn relationships)
Feed-Forward Network (FFN)	Learns deeper non-linear patterns
Layer Norm + Residuals	Stabilizes training and adds signal from earlier layers
Positional Encoding	Adds sequence order info to token embeddings

Self-Attention

- What does self-attention do?
- For a token at position i, the model computes:
 - How much attention should it give to every other token in the sequence?
 - This enables: Contextual understanding: The meaning of a word depends on its surroundings
 - Task flexibility: Same mechanism works whether the model is summarizing or answering questions
- Example (Q/A task): Input: "Q: Who wrote Hamlet? A:"

This is not task-specific logic — the self-attention layers **learn these relationships** from data.

"Hamlet" attends to "wrote"

"A:" attends to "Hamlet" to predict "Shakespeare"

Why the Transformer Enables Multi–Task Learning

- Transformer = Task-Agnostic Neural Network
- No task-specific modules or architecture changes are needed
- It simply learns:
 - How to attend to words
 - How to predict next tokens based on prompt context
- One architecture can **handle many NLP tasks**
- Different tasks are encoded in the **prompt** (e.g., "Translate...", "Summarize...", "Answer...")

Auto-regression Training – Parallelism

- For **decoder-only Transformers** (e.g., GPT models) which is trained to predict the next token at each position.
- Example: Sentence = "I love deep learning"
 - Tokenized as: ["I", "love", "deep", "learning"]
 - During the training, the model sees: Training in **parallel way**

Input	Target Prediction
["I"]	"love"
["I", "love"]	"deep"
["I", "love", "deep"]	"learning"

During the training, we know the full sentence, and compute **all token predictions simultaneously** applying a **masked self-attention** mechanism so the model can't “look ahead” at future words during training.

Training Objective

- Use **cross-entropy loss** to compare predicted vs actual tokens
- Train on massive corpora of unlabelled text
- Model learns **grammar, syntax, semantics, reasoning, and more**

Generalizing to Unseen Tasks

- LMs can be used for different tasks by pre-training a “base” model and then fine-tuning for the task(s) of interest.
- Practical Issues:
 - Too many copies of the model
 - Need for large-scale labeled data for fine-tuning
 - Can do only specific task
- Multi-task Training?
 - Data remains a challenge
 - Humans don’t need such large volumes of data to learn – can we do better?
- Train a model that can perform NLP tasks in a zero-shot manner

LLMs – General model

- Primary shift comes from modeling assumptions from **single-task to general model**
- Traditional NLP
 - Models were built for **one specific task**
 - E.g., one model for translation, another for sentiment analysis, another for summarization.
 - Each model had **a fixed input/output** format, and was trained **only on one type of data**.
- LLMs
 - LLMs (like GPT, T5, PaLM) are trained to perform **many different tasks**.
 - They're trained on **diverse text** with a **unified interface**: everything is just text.
 - The model doesn't “know” the task explicitly — it figures it out from the **prompt**

Tasks Described as **Prompts** (Text)

- **Text-based prompts** are the key enabler of general-purpose models.
 - Instead of hard-coding task logic, we describe the task in **plain language text** as part of the input.
 - The model is **prompted** into doing a task — rather than being trained exclusively for that task.

Task	Input format in modern LLMs
Translation	"Translate this French text to English: Bonjour le monde"
Summarization	"Summarize this article: ..."
Sentiment	"What is the sentiment of this review: I loved the movie!"
...	...

LLMs

- Language models can be used to not just perform a single task, but multiple tasks by learning to predict the next token or sentence through prompting.
- Eg. GPT, Llama2, Gemini, PaLM, Mistral, Mixtral etc.

LLMs Pre-training vs Fine-tuning

1. Pre-training

- Model is trained on billions of text tokens
 - Data includes books, articles, forums, Wikipedia, etc.
 - Model learns to predict the next token in a sentence (self-supervised)
- Example training datasets:
 - "How was your weekend?" → "It was great, thanks!"
 - "Translate 'Bonjour' to English → Hello"
 - Result:
 - A model like GPT-3 or GPT-4 that understands **grammar, logic, reasoning, general world knowledge.**

LLMs Pre-training vs Fine-tuning

2. Fine-tuning - making it **task-specific**

- suppose we want the model to specialize in university admissions.
- What you do: collect task-specific examples:

Q: What documents do I need for a master's application?

A: You need transcripts, letters of recommendation, and a personal statement.

- Fine-tune the pre-trained model on a few thousand examples like this.
- Results: a model that still understands general language — but now responds more accurately to university-related queries.

LLMs Pre-training vs Fine-tuning

- Difference

Stage	Purpose	Data Type	Scale
Pre-training	Learn general language and world knowledge	Massive, general text	Billions of tokens
Fine-tuning	Adapt to a specific task or domain	Labeled task data	Thousands to millions

Optional a 3rd stage: RLHF (*Used in ChatGPT*)

- Reinforcement Learning from **Human Feedback** (RLHF)
- This involves humans ranking model responses to fine-tune it for helpfulness, safety, and alignment.

Training Steps for LLMs

Step 1: Pre-training

- |-- Massive unlabeled text
- |-- Learns grammar, facts, reasoning
- |-- E.g., GPT-3



Step 2: Fine-tuning

- |-- Smaller, domain/task-specific data
- |-- Learns how to respond in specific contexts
- |-- E.g., fine-tune GPT-3 to answer admissions questions



Step 3: (Optional) RLHF

- |-- Human-rated feedback
- |-- Makes responses more helpful and safe
- |-- E.g., ChatGPT

LLMs Architectures

- Encoder-only (BERT)
 - Pre-training : Masked Language Modeling (MLM)
 - Great for classification tasks, but hard to do generation
- Decoder-only (GPT)
 - Pre-training: Auto-regressive Language Modeling
 - Stable training, faster convergence
 - Better generalization after pre-training
- Encoder-decoder (T0/T5)
 - Pre-training : Masked Span Prediction
 - Good for tasks like MT, summarization

BERT

- BERT (Bidirectional **Encoder** Representations from **Transformers**) is pretrained on large text corpora (e.g., Wikipedia + BookCorpus) using **two self-supervised tasks**:
 1. Masked Language Modeling (MLM)
 - Randomly mask out 15% of tokens in a sentence
 - Model learns to predict the original word from **both left and right context**

"The student [MASK] the exam."
 - BERT learns to predict: "passed"
 - This forces the model to understand the entire sentence context, not just left-to-right like GPT.

BERT

2. Next Sentence Prediction (NSP)

- Given two sentences A and B, BERT learns to predict whether B actually follows A.
- Example:

A: "He went to the store."

B: "He bought some milk."

#Task:

Is Next Sentence? → Yes

- This helps the model understand **sentence-level relationships**, useful for tasks like **question answering** and **classification**.

GPT

- GPT (Generative Pretrained Transformer) is a stack of Transformer **decoder** blocks.
 - **The goal** is to train a single model to learn language patterns so it can perform **many tasks** (e.g., translation, summarization, Q&A, coding, etc.) just by seeing **a text prompt** — even without task-specific fine-tuning.
- 1. Pretraining Phase: Autoregressive Language Modeling
 - Objective: Given a sequence of tokens, predict the next token.
 - This is called causal language modeling (CLM), and it's self-supervised — no human labels required.

Input: "The capital of France is"

Target: "Paris"

- GPT sees the input and learns to predict "Paris" based on the context.

GPT Training Process

Step	Description
Data	Massive internet-scale corpora (Common Crawl, books, Wikipedia, code, etc.)
Tokenization	Text is broken into subword tokens using Byte Pair Encoding (BPE)
Model Input	A sequence of tokens like: [The, capital, of, France, is]
Architecture	Decoder-only Transformer with masked self-attention
Loss Function	Cross-entropy loss between predicted token and actual next token
Optimization	Adam optimizer + learning rate scheduler + gradient clipping
Compute	Trained on massive GPU/TPU clusters (GPT-3 used 300B+ tokens and 175B parameters)

How GPT Becomes an LLM

- Once pretrained, GPT can:
 - Complete sentences
 - Answer factual questions
 - Generate code
 - Translate text
 - Summarize documents
- All without changing the architecture — it simply interprets the prompt.

Prompt	Task GPT Performs
"Translate to French: Hello, world"	Translation
"Summarize this: "	Summarization
"What is the capital of Japan?"	Question answering
"Write a Python function to reverse a string."	Code generation

GPT: Zero–Shot and Few–Shot Learning

- Zero-shot: GPT sees only the prompt (no examples) and performs the task.
- Few-shot: You provide a few examples in the prompt (e.g., Q&A pairs), and GPT learns the pattern on the fly - RLHF (Reinforcement Learning from Human Feedback)

GPT – Summary

Stage	Description
Pretraining	Autoregressive language modeling: predict next token from previous tokens
Data	Massive, diverse unlabeled text (web, books, code)
Architecture	Decoder-only Transformer with masked self-attention
Task Handling	Learns many tasks through prompt patterns
Output	Can perform many NLP tasks without explicit task training

BERT vs GPT

Feature	GPT (Generative)	BERT (Encoder)
Pretraining Objective	Predict next token	Predict masked token
Architecture	Decoder-only	Encoder-only
Output	Generates text	Classifies or embeds input
Task handling	Prompt-based (flexible)	Fine-tuning (task-specific)
Usage	Generation, dialogue	Classification, QA (input-only tasks)

How to Finetune LLMs for Specific Applications?

1. Choose the Right Base Model
2. Prepare Your Dataset
3. Choose a Fine-Tuning Strategy
4. Fine-Tune the Model
5. Evaluate and Test
6. Deploy

1. Choose the Right Base Model

- Choose a base model based on your tasks

Task Type	Recommended Model
Text generation	GPT-2, GPT-3, LLaMA, Falcon
Classification	BERT, RoBERTa, DeBERTa
Text-to-text tasks	T5, BART, FLAN-T5
Conversational	GPT-2, DialoGPT, LLaMA-2-Chat

- Use a smaller variant for experimentation (e.g., GPT-2 or T5-small), and scale up later.

2. Prepare Your Dataset

- Format your data appropriately for the model.
 - For classification (e.g., sentiment):
 - { "text": "The product is amazing!",
"label": "positive"}
 - For generation or instruction (e.g., summarization, Q&A):

Input: "Summarize: The stock market fell today due to..."
Target: "The stock market declined because of..."

3. Choose a Fine-Tuning Strategy

- You have 3 main options depending on your compute and data.

Method	Description
Full fine-tuning	Train all model weights on your data (best performance, most compute)
Adapter layers / LoRA	Add lightweight trainable modules on top of a frozen model (efficient)
Instruction tuning	Fine-tune the model with many different prompts and responses to improve task following

Instruction Tuning vs. SFT

Concept	Instruction Tuning	Supervised Fine-Tuning (SFT)
Definition	Fine-tuning a model to follow natural language instructions across many tasks	Fine-tuning on a specific task using labeled data
Input Format	"Summarize this article: ..." or "Translate: ..."	Raw task input (e.g., just a sentence or pair)
Training Objective	Teach the model to follow instructions phrased as prompts	Teach the model to perform a specific task better
Generalization	Enables zero-shot and few-shot task handling	Usually task-specific (limited generalization)
Examples	FLAN-T5, InstructGPT, LLaMA 2-chat	BERT fine-tuned on sentiment classification, T5 for QA
Data Source	Multi-task datasets with varied task descriptions	Labeled datasets for one task (e.g., IMDB, SQuAD)

4. Fine-Tune the Model

- Use libraries like:
 - T5 Transformers (most popular)
 - OpenLLM, PEFT (LoRA support)
 - LLaMA + Alpaca-style finetuning (for decoder models)

5. Evaluate and Test

- Use validation set during training
- After fine-tuning, test on real examples
- Measure performance with:
 - Accuracy / F1 (for classification)
 - BLEU / ROUGE (for generation)
 - Human evaluation (for chatbots)

6. Deploy

- Export the model and serve it via:
 - REST API
 - Chatbot interface (e.g., Streamlit, Gradio)
 - Batch prediction scripts
- You can also quantize the model (e.g., 8-bit) to improve inference speed

More Tips

- Start small with a smaller model (T5-small, GPT-2)
- Use LoRA/PEFT for efficiency if compute is limited
- Use checkpoints and monitor training loss
- Consider **instruction tuning** if your model should follow natural language prompts

References

- https://www.informatics-europe.org/images/ECSS/ECSS2023/slides/ECSS2023_Lapata.pdf
 - Academic paper: Attention is all you need, 2017. <https://arxiv.org/pdf/1706.03762>
-