Fall 2024 COS597R: Deep Dive into Large Language Models

Danqi Chen, Sanjeev Arora

Lecture 1: Overview and Course Structure

https://princeton-cos597r.github.io/

LLMs: Background

Survey of older ideas

- [Shannon 1950s] Information theory view. Language = distribution on strings of words.
- n-gram models (modern terminology: n = "context length")

$$\Pr[w_1 w_2 \ldots w_L] = \prod_i \Pr[w_{i+1} \mid w_1 w_2 \ldots w_i] \approx \prod_i \Pr[w_{i+1} \mid w_{i-n+1} \ldots w_i]$$

- Information-theoretic measure of goodness cross-entropy loss; estimator for KLdivergence

$$\sum_i \log \frac{1}{\Pr_\theta[w_{i+1} \mid w_1 w_2 \ldots w_i]}$$

- Can be viewed as a form of self-supervised learning : "predict the next word using previous n words". n = context length
- ELMO, BERT, ROBERTA (2017-19): Predict missing word.

Ideas leading up to current LLMs

- Deep learning attempts starting 1990s (Schmidhuber, Bengio, Collobert-Weston etc.)
- Recurrent RNs as attempt to get “arbitrary context length”; but optimization doesn’t scale
- Attention-based models. (First modern use in 2013, then in 2014 in the first LLM with attention.) **“Attention is all you need.”** [2017].
- Adapted architectural and optimization innovations of preceding years: normalization, residual connections, adaptive gradient methods, LR schedules, ..
- ELMO, BERT, ROBERTA, T5 (2017-19): “Predict the masked word”
- Scaling Laws (2019-21). Reason why LLMs (i) got scaled up (GPT1 to 4) (ii) are expensive
- **Next word prediction goes much further than almost anybody expected..**

Technical innovations since 2021

- Instruction-tuning (converts next-word predictor to useful chat agent)

  Idea (instruct-GPT):

  (i) Fine-tune on (Q, A) pairs. ("supervised fine-tuning")

  (ii) Self-improvement via simple RL ideas, e.g. RLHF, PPO, DPO,..

- "Alignment problem" (how to ensure the AI agent does not answer "inappropriately"). Current solutions sort-of-OK. Need work
- Improved high-level reasoning (math, code etc.). Involved all aspects of the training pipeline. (Current capabilities were conjectured to require 5-10 years of progress!)
- Long context length (millions). Sort-of works; still a work in progress.
- Rapid improvements in small models. (distillation, pruning, training on synthetic data) . Llama 3.1 8B > PaLM 540B.
- Fast inference/falling costs . (frontier model cost is down 10x in a year, cost for the same capability is 25x lower, e.g. free Llama3, Gemma 2 etc.)
- Agents, not simple Q&A

About the course

Goal : Detailed look at LLM Research

- model architecture
- data preparation
- algorithm details (learning rates, optimizers..)
- pre-training, including assembling a data corpus
- post-training/alignment
- deployment optimizations; fast inference
- enhancing models via post-training (eg math, reasoning, specialized knowledge)
- LLM evaluations: design, judging quality etc.
- how alignment may change as models get stronger (debate, weak-2-strong etc.)
- societal issues: legal frameworks, economic effects, copyright etc.
- some advanced topics TBD (eg distillation, long context, data selection, RL-based ideas)

Focus on conceptual understanding and research rather than engineering.
Highly interactive format

Course structure

- Some lecture + discussion every time.
- 30 min “debate” involving a group of 5 (12x times starting week 4)
- Project presentations (last 3 lectures)

Requirements

- Do the readings, participate in class discussions
- Maintain “journal” (responses to prompts about readings)
- Scribe notes for a lecture (group of 3)
- Participate (in small groups) in a “debate” about an assigned reading
- Course project (in groups of 2-3). Do in-class presentation + Long report

Grading in the course

- class participation + reading responses 30%
- debate 15%
- lecture scribing 10%
- final project 35%
- project presentation 10%

Debate Format

Inspiration1: “AI safety via Debate” [Irving, Christiano, Amodei’18] + followup works

Inspiration 2: Conference review/rebuttal/decision process.

- Readings of the lecture (1-2 papers). Everybody reads lightly.
- 1 “presenter” presents paper in 8 min
- Two teams of 2 debaters ea
- “Critics” point to potential flaws/unconvincing experiments/incomplete reasoning.
- “Proponents” defend, give justifications etc. (could draw upon follow-up papers)
- Later, everybody collaborates to write a “meta-review” (≈ 2 pages)

Your goal: Learn to read papers (including your own writing) with a critical eye.

Develop taste, insight, rigorous reasoning

How LLMs are created: Overview

Recap: Transformer architecture

(How many people need this?)

Step 0. Create series of small LLMs

(e.g., 100M, 300M, 700M, ..)

Goal: Derive “scaling laws" to “predicting” best hyper parameters for large-scale LLM: Architecture, corpus size, data mixtures, learning rates, etc.

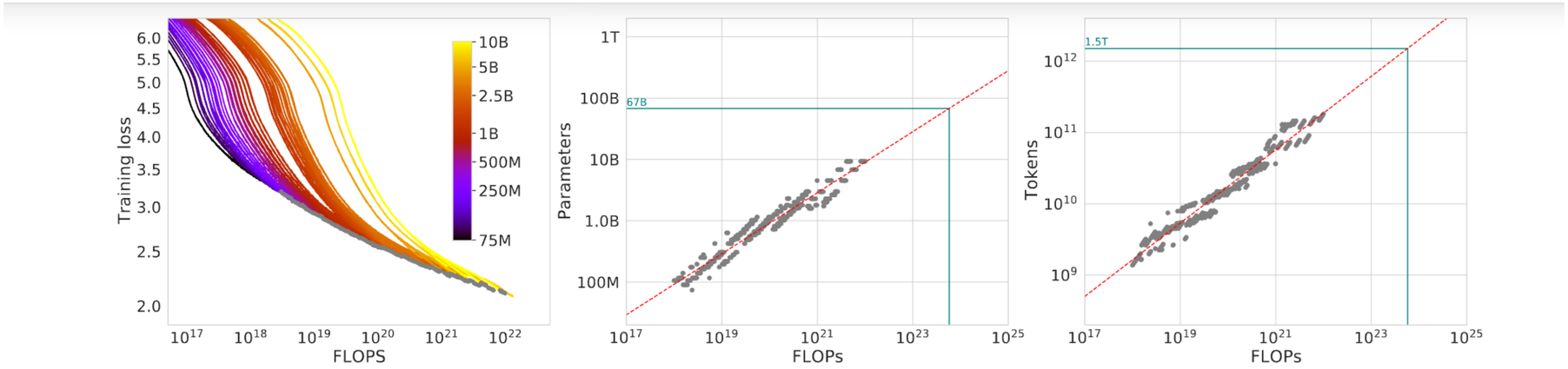


Figure 2 | **Training curve envelope.** On the **left** we show all of our different runs. We launched a range of model sizes going from 70M to 10B, each for four different cosine cycle lengths. From these curves, we extracted the envelope of minimal loss per FLOP, and we used these points to estimate the optimal model size (**center**) for a given compute budget and the optimal number of training tokens (**right**). In green, we show projections of optimal model size and training token count based on the number of FLOPs used to train *Gopher* (5.76×10^{23}).

Training compute-optimal LLMs; Hoffman et al’22

Step 1. Pre-training Corpus

Biggest component: Scraped text from web, cleaned up (e.g., remove adult content, copyrighted text, junk html etc.) and deduplicated. Cleaning uses tiny/small models.

Specialized Data: Code, books, wikipedia, journals, reddit etc.

How much data? Early "compute optimal" models were (in hindsight) undertrained

Current trend: Prioritize high-capability in small models; overtrain (as much data as possible)

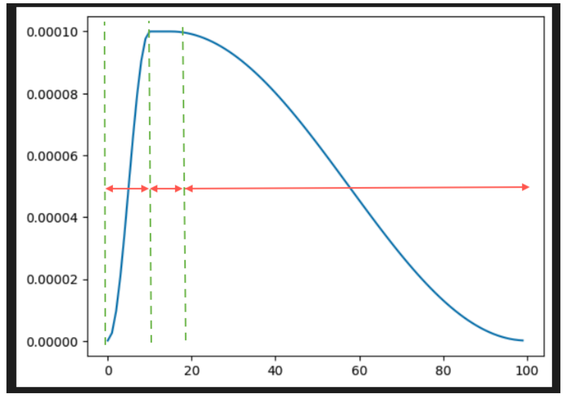
Llama 3 8B: # pre-training tokens/ # parameters = 600

Llama1 dataset (1.4T tokens)

Dataset	Sampling prop.	Epochs	Disk size
CommonCrawl	67.0%	1.10	3.3 TB
C4	15.0%	1.06	783 GB
Github	4.5%	0.64	328 GB
Wikipedia	4.5%	2.45	83 GB
Books	4.5%	2.23	85 GB
ArXiv	2.5%	1.06	92 GB
StackExchange	2.0%	1.03	78 GB

Step 2: Pre-training

Simplest version: Create batches as per data mix, train transformer via AdamW on cross-entropy loss with cosine LR schedule



Cosine LR schedule with linear warmup

Enhancements: (i) Add other losses (e.g., masked prediction);
(ii) vary data mix a few times (initial, middle and late stages)
(iii) special treatment of copyrighted or private data
(iv) lots others

e.g., “Middle stage” (around 30% -40%) may emphasize code.
“Late stage” (80% or 90%) may emphasize high-quality data-sources (wikipedia, books, arxiv)

Step 2: Pre-training (contd)

Simplest version: Create batches as per data mix, train transformer via GD on cross-entropy loss with cosine LR schedule

Llama3 paper:

3.4.1 Initial Pre-Training

We pre-train Llama 3 405B using AdamW with a peak learning rate of 8×10^{-5}, a linear warm up of 8,000 steps, and a cosine learning rate schedule decaying to 8×10^{-7} over 1,200,000 steps. We use a lower batch size early in training to improve training stability, and increase it subsequently to improve efficiency. Specifically, we use an initial batch size of 4M tokens and sequences of length 4,096, and double these values to a batch size of 8M sequences of 8,192 tokens after pre-training 252M tokens. We double the batch size again to 16M after pre-training on 2.87T tokens. We found this training recipe to be very stable: we observed few loss spikes and did not require interventions to correct for model training divergence.

Careful engineering everywhere, especially wrt efficient use of hardware (sharding, balancing communication/computation, etc.), numerical issues, recovery from errors and hardware failures etc..

Stage 3: Instruction-tuning

Turns next-word predictor into instruction-following agent

Simplest version: Fine-tune on (i) Q&A data of various kinds (eg science, math)
(ii) “general instruction-following” Q&A data (“Write me a 3-page essay on famines in South Asia and how they influenced historical events”)

Q&A data sourced from humans (potentially skilled ones). Expensive!

2nd stage usually involves self-improvement via some reinforcement learning
(may involve training additional “reward model” to rank the model’s answers by quality)

Simplest RL: “Best of n”

Stage 3: Instruction-tuning (contd)

Turns next-word predictor into instruction-following agent

Advanced version (e.g., Llama3 paper)

(i) extensive Q&A (some multi-turn) in code, math, reasoning, factuality, tool-use (e.g. using p
interpreter, web-search etc.).

(ii) Lots of synthetic data supplemented with human annotations

(iii) this stage is also used to enable “long context” (8K —> 128K)

Stage 4: Alignment/Safety

Goal: AI should be Helpful, Safe, Courteous, Accurate, Informative etc.
(HHH: “Helpful, Honest, Harmless”)

e.g., how should the AI answer “How many pills of 500mg tylenol would be lethal?”

General ideas (i) do pre-training carefully to avoid hallucination of facts (ii) instruction-tuning with “good” responses
(iii) RL-like technique or “Constitutional AI” to improve response quality wrt desired qualities
(iii) avoid “jailbreaks” by including an overseer/filter model to detect unsafe content.

(A General Language Agent Assistant as a Laboratory for Alignment. Askell et al ’21)
(Constitutional AI: Harmlessness from AI Feedback. Bai et al ’22)
(Training Language Models to follow Instructions with Human Feedback. Ouyang et al’22)

Constitutional AI

(Constitutional AI: Harmless from AI Feedback. Bai et al, 2022)

"Constitution": List of a couple dozen general principles of good behavior

Self-improvement step

- AI model is given constitution + tricky prompt (generated via "red teaming")
- Once it generates an answer, it is asked to self-reflect ("Chain of thought") on whether the answer was consistent with constitution. And suggest ways in which the answer could be improved.
- Fine-tune AI on many such prompts and its own past answers

"We find that as language model capabilities improve, AI identification of harms improves significantly. Furthermore, chain-of-thought reasoning improves this ability, and leads to evaluations that are becoming competitive with preference models trained on human feedback labels."

Caveat: As in rest of this lecture, many details omitted. The paper omits yet more..

Stage 5: Tricks for Fast Inference

- Quantize to reduce memory requirement (e.g., Llama3 8B run on MacBooks)
- Use pipelining to improve GPU utilization.
- Various generation heuristics (random sampling from distribution isn’t always best)

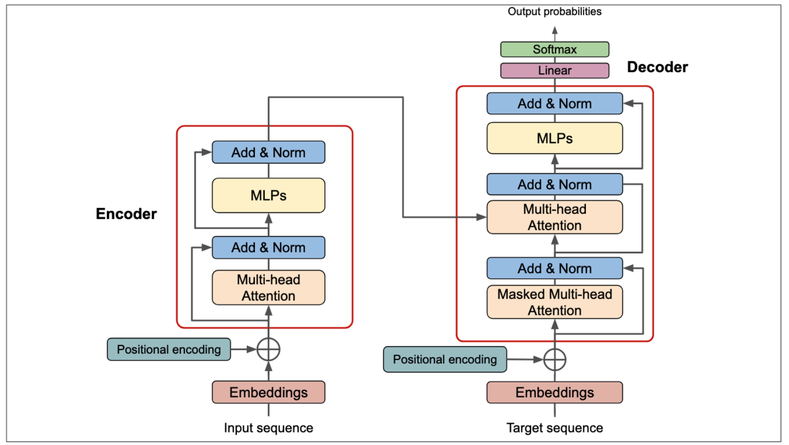
A few things that intrigue me

Some things that interest me..

1. **memorization**: necessary/good/bad/ugly? Copyright?
2. **many possible tradeoffs that can improve efficiency**
3. **Choosing data that improves training**. What order to use in learning?
4. **Code and reasoning, math**
5. **Is intelligence just statistical?** or something else? Does next-word prediction suffice?
6. **Developing higher-order conceptions of learning: skills, agentic (RL-based), self-directed,..** These are akin to "procedures", which is of higher-order than "knowledge."
7. **Is super-intelligence possible?** What does it mean? Where does it get the intelligence from?

That’s all for today. Questions?

(I have a few undergrad level slides on attention and transformers, if of interest)



LLMs: The transformer architecture

(Training = gradient descent (backprop!) on cross-entropy loss of the word probabilities computed by this architecture.)

"Attention is all you need" Vaswani et al 2017.

(Ideas appeared earlier, this paper identified the key core and made it sing)

Vision : convolution :: Language : ??

You

Please solve this problem. Give a justification in 5 lines or less.

Vision: convolution :: Language: ??

ChatGPT

Vision: convolution :: Language: **attention**

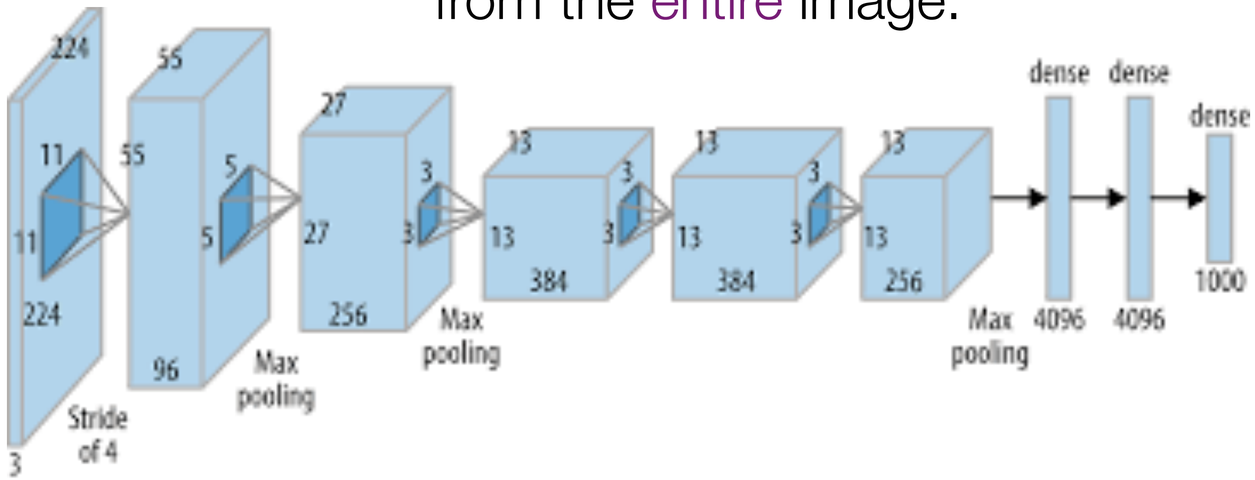
Justification: In the realm of deep learning, convolutional neural networks (CNNs) have become the standard for processing visual data, efficiently capturing spatial hierarchies in images. For language processing, the transformer architecture, which is built upon the concept of attention mechanisms, has set a new standard. Attention mechanisms allow models to focus on different parts of the input sequence when generating each part of the output, making them particularly effective for handling the complexities of language data.

Recap: Why convolutions make sense for vision

The camera can be placed arbitrarily to capture the scene

$\Longrightarrow$ Nothing special about any specific pixel (i, j). With another camera position this pixel would have been at another location (i', j')

Important: This property holds only for individual convolution filters. Multilayered nets end up applying a sequence of convolutions (not to mention pooling layers) which have the effect of aggregating information from the entire image.



Attention layer in a transformer

How transformer makes sense of:

"***The trophy didn't fit in the suitcase because it was too big***."

Analogy we'll use: trading pit

Trading pit, Chicago Board of Trade. (c) AP 1995

Attention layer in a transformer

Input to transformer: "***The trophy didn't fit in the suitcase because it was too big***."

"key$_i$" = Info that ith trader has

"query$_i$" = Info that ith trader seeks

} Vectors in $\mathfrak{R}^d$

I have something that might be considered valuable, or large (maybe giant cup). Who needs info about this object to complete their action?

"I'm dealing with an issue of not fitting. I need to know which object is involved and what's causing the issue!"

"I see signs that something caused something else"

"Seeing situation where something was too large for some purpose. What?"

"The trophy" **"didn't fit"** **"in"** **"the suitcase"** **"because"** **"it"** **"was too big"**

The trading pit

Attention layer: Weight sharing!

Input to transformer: "***The trophy didn't fit in the suitcase because it was too big***."

"key$_i$" = Info that ith trader has

"query$_i$" = Info that ith trader seeks

} Vectors in $\Re^d$

Actually a single "trader" (i.e., neural net) is used to "trade" at all positions.

traders = # of words being processed at a time by LLM (denoted by C)

"The trophy" **"didn't fit"** **"in"** **"the suitcase"** **"because"** **"it"** **"was too big"**

Attention layer: Weight sharing!

Input to transformer: "***The trophy didn't fit in the suitcase because it was too big***."

Vector "purchased" by trader j in this round

$$= \sum_i s_i^j \ \overrightarrow{\text{value}_i}$$

where $(s_1^j, s_2^j, \ldots, s_C^j) =$

$$\text{softmax}(\overrightarrow{\text{query}_j} \cdot \overrightarrow{\text{key}_1}, \overrightarrow{\text{query}_j} \cdot \overrightarrow{\text{key}_2}, \ldots, \overrightarrow{\text{query}_j} \cdot \overrightarrow{\text{key}_C})$$

$\overrightarrow{\text{key}_i}$ = offer vector of ith trader

$\overrightarrow{\text{query}_i}$ = demand vector of ith trader

$\overrightarrow{\text{value}_i}$ = bundle of goods this trader offers

All vectors are in $\mathfrak{R}^d$

"The trophy" **"didn't fit"** **"in"** **"the suitcase"** **"because"** **"it"** **"was too big"**

Attention layer: Weight sharing!

Input to transformer: "***The trophy didn't fit in the suitcase because it was too big***."

Vector "purchased" by trader j in this round

$$= \sum_i s_i^j \ \overrightarrow{\text{value}_i}$$

where $(s_1^j, s_2^j, \ldots, s_C^j) =$

$$\text{softmax}\left(\frac{1}{\sqrt{d}}(\overrightarrow{\text{query}_j} \cdot \overrightarrow{\text{key}_1}, \overrightarrow{\text{query}_j} \cdot \overrightarrow{\text{key}_2}, \ldots, \overrightarrow{\text{query}_j} \cdot \overrightarrow{\text{key}_C})\right)$$

$\overrightarrow{\text{key}_i}$ = offer vector of ith trader

$\overrightarrow{\text{query}_i}$ = demand vector of ith trader

$\overrightarrow{\text{value}_i}$ = bundle of goods this trader offers

All vectors are in $\mathfrak{R}^d$

Operation at each layer: For each location j compute its "purchase" vector

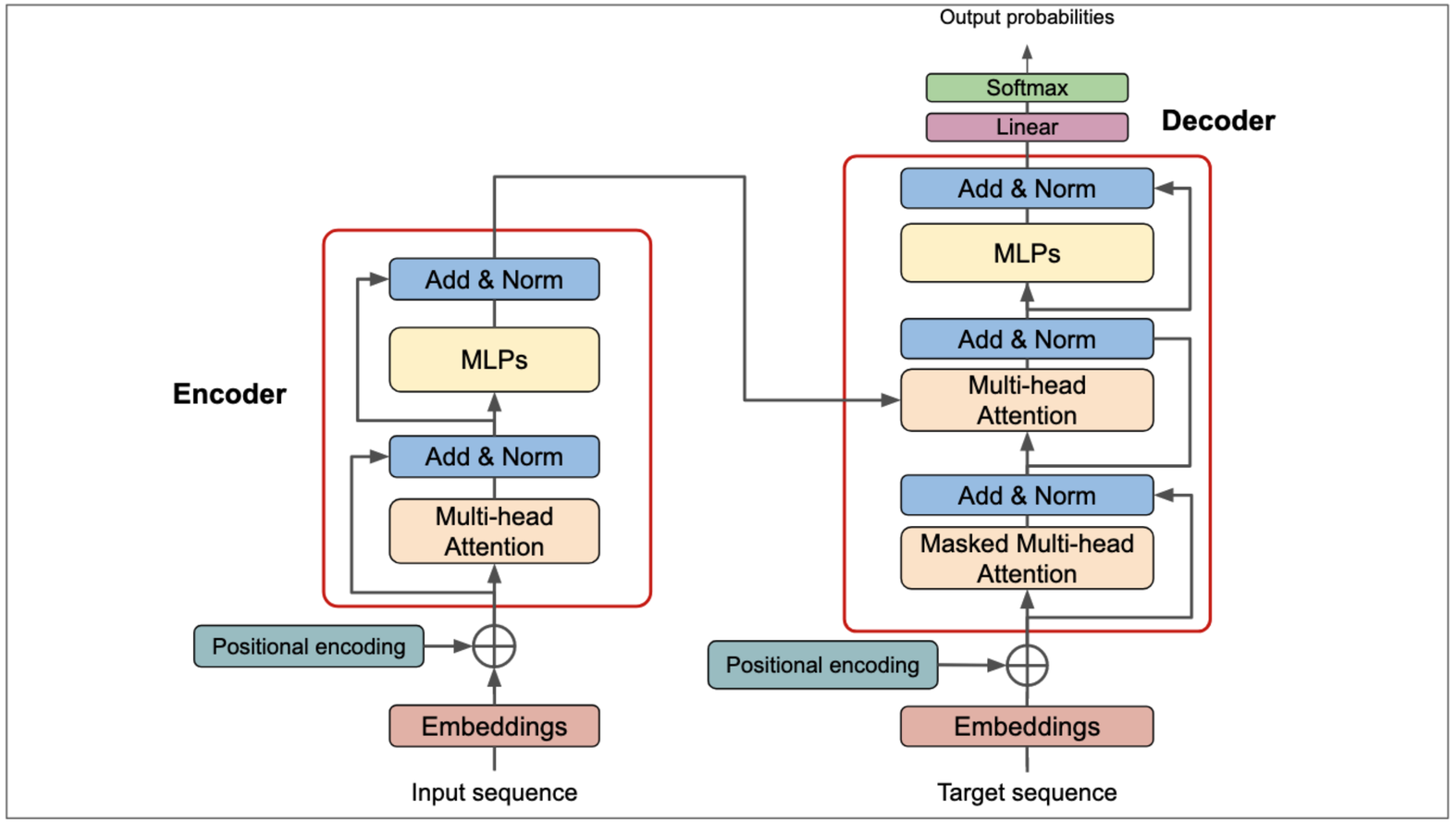
Pass its current three vectors and "purchase" vector through a small FC net

to get three vectors for this location for use in the trading at **next higher** layer.

NB: At the output layer, the "trading" is followed by softmax that produces a probability distribution for predicting the next word

Basic attention unit

"Attention is all you need" Vaswani et al 2017.



Output of attention module

$$\text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Q = matrix of $\overrightarrow{\text{query}_i}$'s

K = matrix of $\overrightarrow{\text{key}_i}$'s

V = matrix of $\overrightarrow{\text{value}_i}$'s

This unit is replicated at each word position (analogous to convolution)