

Foundations of machine learning

# Large language models: Training and architecture

Maximilian Kasy

Department of Economics, University of Oxford

Hilary term 2027

# Outline

- Language models and the training pipeline.
  - The decision problem at each stage: Pretraining, supervised fine-tuning, preference optimization, RL with verifiable rewards.
- Training algorithms.
  - Stochastic gradient descent, reward models, policy gradient, direct preference optimization.
- The transformer architecture.
  - Self-attention, transformer blocks.

Running example: Olmo 3, a fully open model (data, code, checkpoints).

## Takeaways for this part of class

- A language model is a model  $\pi_{\theta}(y_t|y_1, \dots, y_{t-1})$  for the next token.  
All stages of training choose  $\theta$  for this same model.
- Pretraining and supervised fine-tuning:  
Maximum likelihood on given text, i.e., minimizing KL divergence to the data.
- Preference optimization and RL with verifiable rewards:  
Responses that evaluators prefer should become more likely,  
without moving too far from a reference model.
- Policy gradient methods estimate the direction of improvement.  
Baselines reduce the variance of these estimates.
- Transformers stack self-attention layers (kernel smoothing across positions)  
and position-wise neural nets.

Language models and the training pipeline

Training algorithms

The transformer architecture

# Language models

- Text is split into *tokens*: elements of a finite vocabulary  $\mathcal{V}$ . Words, pieces of words, punctuation. Olmo 3:  $|\mathcal{V}| \approx 100,000$ .
- A language model is a distribution over token sequences  $y = (y_1, \dots, y_T)$ ,

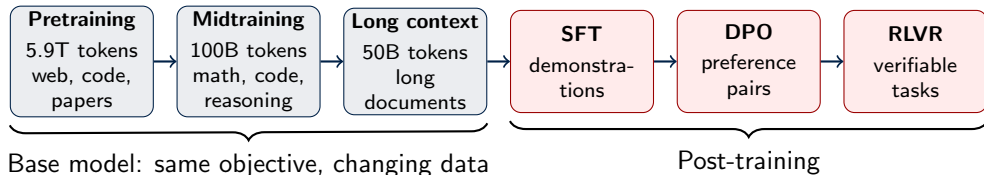
$$p_{\theta}(y) = \prod_{t=1}^T \pi_{\theta}(y_t | y_{<t}), \quad y_{<t} = (y_1, \dots, y_{t-1}).$$

- Prompts  $x$  (questions, instructions, conversations so far) and responses  $y$ :

$$\pi_{\theta}(y|x) = \prod_{t=1}^T \pi_{\theta}(y_t | x, y_{<t}).$$

- Generation: Sample  $y_1$ , then  $y_2$  given  $y_1$ , etc., until an end-of-text token.
- Recall the RL lecture:  $\pi_{\theta}$  is a policy.  
State: the text so far. Action: the next token.

# The training pipeline of Olmo 3



- Olmo 3 (Allen Institute for AI, 2025): 7B and 32B parameter models. Data, training code, and intermediate checkpoints are public.
- Similar pipelines are used for most current models.
- Next: The decision problem (data, objective) for each stage.

## Reminder: Kullback-Leibler divergence

- For two distributions  $P$  and  $Q$  of a discrete random variable  $Y$ :

$$D_{KL}(P\|Q) = E_{Y \sim P} \left[ \log \frac{P(Y)}{Q(Y)} \right] = \sum_y P(y) \log \frac{P(y)}{Q(y)}.$$

- $D_{KL}(P\|Q) \geq 0$ , with equality iff  $P = Q$  (by Jensen's inequality).  
Not symmetric:  $D_{KL}(P\|Q) \neq D_{KL}(Q\|P)$  in general.

- Connection to maximum likelihood:

For  $Y_i \sim P$  i.i.d. and a model  $Q_\theta$ ,

$$\frac{1}{n} \sum_{i=1}^n \log Q_\theta(Y_i) \rightarrow E_P[\log Q_\theta(Y)] = E_P[\log P(Y)] - D_{KL}(P\|Q_\theta).$$

$\Rightarrow$  Maximum likelihood minimizes an estimate of  $D_{KL}(P\|Q_\theta)$ .

If the model is misspecified, it targets the KL-closest model (pseudo-true parameter).

# Pretraining

- Data: A huge corpus of documents  $y \sim P$ .
- Objective: Minimize the expected log loss (negative log likelihood),

$$R(\theta) = E_{y \sim P} \left[ - \sum_t \log \pi_\theta(y_t | y_{<t}) \right].$$

- By the previous slide, this is equivalent to minimizing

$$\sum_t E [D_{KL}(P(\cdot | y_{<t}) \| \pi_\theta(\cdot | y_{<t}))].$$

- Minimized by the true conditional distribution:  
Log loss is a proper scoring rule.
- Pretraining is maximum likelihood estimation  
of a very flexible model for the distribution of text.

# Supervised fine-tuning (SFT)

- Data: Demonstrations  $(x, y)$ .  
Prompts and high-quality responses (written by people, or by stronger models).

- Objective: The same log loss, but only for the response tokens,

$$R_{SFT}(\theta) = E \left[ - \sum_t \log \pi_{\theta}(y_t | x, y_{<t}) \right].$$

- The prompt is conditioned on, not predicted.
- Teaches the format of interactions:  
Following instructions, multi-turn chat, step-by-step reasoning.
- This is *imitation learning*:  
Learn a policy from observing the actions of an expert.

# Learning from preferences

- For many prompts, good responses are easier to recognize than to write.
- Data: Prompts  $x$ , pairs of responses, and a judgment which one is better,  $y_w \succ y_l$ . Judgments by people, or by another model.
- Suppose there was a reward function  $r(x, y)$ . Objective:

$$\max_{\theta} E_x \left[ E_{y \sim \pi_{\theta}(\cdot|x)} [r(x, y)] - \beta \cdot D_{KL}(\pi_{\theta}(\cdot|x) \parallel \pi_{\text{ref}}(\cdot|x)) \right].$$

- $\pi_{\text{ref}}$ : Reference model, typically the SFT model.  $\beta > 0$ : Penalty weight.
- Why the penalty?  
The reward is estimated, and only reliable close to the data.  
Keep the language ability of the reference model.
- New: The expectation is over the model's *own* responses.

# Reinforcement learning with verifiable rewards (RLVR)

- For some tasks, correctness can be checked automatically:
  - Math problems with known answers.
  - Code with unit tests.
  - Instructions with checkable constraints (“answer in less than 50 words”).
- Reward:  $r(x, y) = 1$  if the verifier accepts the response, 0 otherwise.
- Objective: Same as before (possibly with  $\beta = 0$ ).  
No reward model needs to be estimated.
- Only the final answer is checked; the tokens before it are unconstrained.  
 $\Rightarrow$  Models learn to produce long chains of reasoning before answering.  
“Thinking” models (Olmo 3 Think, DeepSeek-R1).

## Summary: Four decision problems for one model

| Stage       | Data                    | Who writes $y$ ? | Objective                                                                                   |
|-------------|-------------------------|------------------|---------------------------------------------------------------------------------------------|
| Pretraining | documents               | people           | $\min_{\theta} E_P[-\log \pi_{\theta}(y)]$                                                  |
| SFT         | prompts, demonstrations | people / models  | $\min_{\theta} E[-\log \pi_{\theta}(y x)]$                                                  |
| Preferences | prompts, comparisons    | $\pi_{\theta}$   | $\max_{\theta} E_{\pi_{\theta}}[r] - \beta D_{KL}(\pi_{\theta} \parallel \pi_{\text{ref}})$ |
| RLVR        | prompts, verifier       | $\pi_{\theta}$   | $\max_{\theta} E_{\pi_{\theta}}[r] - \beta D_{KL}(\pi_{\theta} \parallel \pi_{\text{ref}})$ |

- First two rows: Fit the distribution of given data.  
Minimize  $D_{KL}(P \parallel \pi_{\theta})$ : Learning from examples.
- Last two rows: Improve the model's own behavior.  
Our actions determine what we observe, as in bandits and RL:  
Learning from feedback.

Language models and the training pipeline

Training algorithms

The transformer architecture

## Pretraining: Stochastic gradient descent

- No closed form for the maximum likelihood estimator:  
Follow the gradient of the log likelihood.
- The network produces logits  $z = f_{\theta}(y_{<t})$ , and  $\pi_{\theta}(\cdot|y_{<t}) = \text{softmax}(z)$ .  
Gradient of the log loss with respect to the logits:

$$\nabla_z (-\log \pi_{\theta}(y_t|y_{<t})) = \pi_{\theta}(\cdot|y_{<t}) - e_{y_t}.$$

- Interpretation: Raise the probability of the token that actually came next, lower all others in proportion to their predicted probability.
- Gradient with respect to  $\theta$ : Chain rule, i.e., backpropagation.
- The gradient for a random batch of documents is an unbiased estimate of the full gradient, at a small fraction of the cost.

## One pass through the data: Online learning

- Pretraining makes (roughly) one pass through the data.  
Each gradient step uses new documents.
- This is the online convex optimization setting:  
Loss  $f_t(\theta)$  for batch  $t$ , and online gradient descent.
- Recall: For convex losses, regret of online gradient descent is  $O(\sqrt{T})$ .  
Online-to-batch conversion: Average iterate has excess risk  $O(1/\sqrt{T})$ .
- The loss on each new batch is an out-of-sample estimate of the risk.
- Caveat: Neural nets are not convex.  
The theory provides intuition, not guarantees.

# Why go beyond imitation?

- SFT uses the same algorithm, on prompt-response pairs.
  - Distribution shift:
    - In training, the context is always produced by the expert.
    - In use, the context includes the model's own previous tokens.
    - Errors compound: A mistake leads to contexts never seen in training.
  - Imitation cannot exceed the quality of the demonstrations.
- ⇒ Learn from feedback on the model's own responses:  
Responses that evaluators prefer should become more likely.  
Evaluators: people, a model trained on their judgments, or automated checks.

# Reward models

- Preference data  $(x, y_w, y_l)$ . Bradley-Terry model:

$$P(y_w \succ y_l | x) = \sigma(r(x, y_w) - r(x, y_l)), \quad \sigma(u) = \frac{1}{1 + e^{-u}}.$$

- This is a random utility model:  
Utility  $r(x, y) + \varepsilon$ , with  $\varepsilon$  i.i.d. type I extreme value  $\Rightarrow$  binary logit.
- Estimate a reward model  $r_\phi$  (a language model with scalar output) by maximum likelihood.
- Identification:  $r(x, y)$  and  $r(x, y) + c(x)$  imply the same choice probabilities.  
Only differences between responses to the same prompt are identified.

# Policy gradient: Making preferred responses more likely

- Goal: Increase the expected reward  $E_{y \sim \pi_\theta(\cdot|x)}[r(x, y)]$ .
- Policy gradient theorem (RL lecture), with one response as one episode:

$$\nabla_\theta E_{y \sim \pi_\theta(\cdot|x)}[r(x, y)] = E_{y \sim \pi_\theta(\cdot|x)}[r(x, y) \cdot \nabla_\theta \log \pi_\theta(y|x)].$$

- $\nabla_\theta \log \pi_\theta(y|x)$  is the direction that makes response  $y$  more likely.
- Unbiased estimate: Sample a response from the current model, and move in this direction, scaled by its reward (REINFORCE).
- KL penalty: Use  $r - \beta \log(\pi_\theta / \pi_{\text{ref}})$  as reward.
- Problem: If all rewards are positive, every sampled response is pushed up. Good responses gain only on average  $\Rightarrow$  noisy gradient estimates.

## Reducing variance: Baselines

- Among unbiased estimates of the gradient, we prefer those with smaller variance.
- For any baseline  $b(x)$  that does not depend on the response,

$$E_{y \sim \pi_\theta} [b(x) \nabla_\theta \log \pi_\theta(y|x)] = b(x) \sum_y \nabla_\theta \pi_\theta(y|x) = b(x) \nabla_\theta 1 = 0.$$

- So  $(r(x, y) - b(x)) \nabla_\theta \log \pi_\theta(y|x)$  is also unbiased.  
Its variance is small if  $b(x)$  is close to the expected reward.
- Then: Responses better than expected become more likely, responses worse than expected become less likely.
- GRPO (used in Olmo 3): Sample a group of responses for each prompt; baseline  $b(x) =$  their average reward.
- If all responses in a group get the same reward, there is no signal.  
Training prompts should be neither too easy nor too hard.

## Practice problem (Group baselines)

For a prompt  $x$ , sample  $G$  responses  $y_1, \dots, y_G$  i.i.d. from  $\pi_\theta(\cdot|x)$ , with rewards  $r_i = r(x, y_i)$ .

Let  $\bar{r} = \frac{1}{G} \sum_j r_j$ , and  $\bar{r}_{-i} = \frac{1}{G-1} \sum_{j \neq i} r_j$ .

1. Show that the leave-one-out baseline gives an unbiased gradient estimate:

$$E[(r_i - \bar{r}_{-i}) \nabla_\theta \log \pi_\theta(y_i|x)] = \nabla_\theta E_{y \sim \pi_\theta}[r(x, y)].$$

2. Show that  $r_i - \bar{r} = \frac{G-1}{G}(r_i - \bar{r}_{-i})$ .

What does this imply for the GRPO baseline  $\bar{r}$ ?

3. What is the update if all  $G$  responses get the same reward?

# The solution of the penalized problem

## Practice problem

Fix a prompt  $x$ . Show that the maximizer over all distributions  $\pi$  of

$$E_{y \sim \pi}[r(x, y)] - \beta \cdot D_{KL}(\pi \parallel \pi_{\text{ref}}(\cdot|x))$$

is given by

$$\pi^*(y|x) = \frac{1}{Z(x)} \cdot \pi_{\text{ref}}(y|x) \cdot \exp(r(x, y)/\beta),$$

where  $Z(x) = \sum_y \pi_{\text{ref}}(y|x) \exp(r(x, y)/\beta)$ .

Hint: Write the objective as  $-\beta \cdot D_{KL}(\pi \parallel \pi^*) + \text{const.}$

## Solution and interpretation

- Rearranging,

$$E_{\pi} \left[ r - \beta \log \frac{\pi}{\pi_{\text{ref}}} \right] = -\beta \cdot E_{\pi} \left[ \log \frac{\pi}{\pi_{\text{ref}} e^{r/\beta} / Z} \right] + \beta \log Z = -\beta \cdot D_{KL}(\pi \| \pi^*) + \beta \log Z.$$

$D_{KL} \geq 0$ , with equality iff  $\pi = \pi^*$ .  $\square$

- Interpretation as Bayes' rule:  
Prior  $\pi_{\text{ref}}$ , "likelihood"  $\exp(r/\beta)$ , posterior  $\pi^*$ .
- $\beta \rightarrow 0$ : Choose the reward-maximizing response.  $\beta \rightarrow \infty$ : Stay with  $\pi_{\text{ref}}$ .
- $Z(x)$  sums over all possible responses, so  $\pi^*$  cannot be computed directly.  
Policy gradient methods approach it by sampling.

# Direct preference optimization (DPO)

- Idea: We know the form of the optimal policy.  
Fit it to the preference data directly, without a reward model.
- Invert the solution:  $r(x, y) = \beta \log \frac{\pi^*(y|x)}{\pi_{\text{ref}}(y|x)} + \beta \log Z(x)$ .
- Substitute into Bradley-Terry.  
 $Z(x)$  cancels, since only reward differences matter.
- Maximum likelihood for the policy:

$$\min_{\theta} - \sum_i \log \sigma \left( \beta \log \frac{\pi_{\theta}(y_{w,i}|x_i)}{\pi_{\text{ref}}(y_{w,i}|x_i)} - \beta \log \frac{\pi_{\theta}(y_{l,i}|x_i)}{\pi_{\text{ref}}(y_{l,i}|x_i)} \right).$$

- Preferred responses become more likely, rejected ones less likely, relative to the reference model.
- Supervised learning: No reward model, no sampling. Used in Olmo 3 after SFT.

# Over-optimization

- The estimated reward  $r_\phi$  is only a proxy for what evaluators prefer.
- Optimizing hard against the proxy:  
True quality first rises, then falls, as  $\pi_\theta$  moves away from  $\pi_{\text{ref}}$ .
- Recall shrinkage and the winner's curse:  
Responses with the highest *estimated* reward are disproportionately those with the largest *estimation error*.
- The penalty  $\beta$  controls how far we move from  $\pi_{\text{ref}}$ .
- Verifiers can be gamed as well:  
E.g., code that special-cases the unit tests.

Language models and the training pipeline

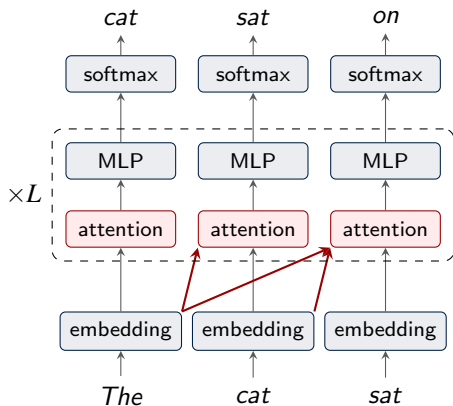
Training algorithms

The transformer architecture

# What does the architecture need to do?

- So far, we only needed  $\pi_{\theta}(y_t|y_{<t})$  to be differentiable in  $\theta$ .
- Which functional form? Requirements:
  1. Map contexts of varying length to a distribution over  $\mathcal{V}$ .  
Contexts can be long (Olmo 3: up to 65,536 tokens).
  2. Causality: The prediction of  $y_{t+1}$  may only use  $y_1, \dots, y_t$ .
  3. Relevant context can be far back in the text.
- Leading answer: The transformer (Vaswani et al., 2017).
- One vector per position, transformed in  $L$  rounds.  
The final vector at position  $t$  yields the prediction for  $y_{t+1}$ .

# Overview of the transformer



Prediction of the next token, at each position

Output: multinomial logit over  $\mathcal{V}$

MLP: neural net, each position separately

**Attention:** combines information across positions  $j \leq i$

Embedding: lookup table, token  $\mapsto$  vector in  $\mathbb{R}^d$

Input tokens  $y_1, \dots, y_t$

Dashed box: One transformer block, repeated  $L$  times (plus residual connections and normalization).

# Self-attention

- Linear maps of each position's vector  $h_i$ :  
query  $q_i = W^q h_i$ , key  $k_j = W^k h_j$ , value  $v_j = W^v h_j$ .
- Output at position  $i$ : Weighted average of values at positions  $j \leq i$ ,

$$o_i = \sum_{j \leq i} \alpha_{ij} v_j, \quad \alpha_{ij} = \frac{\exp(\langle q_i, k_j \rangle / \sqrt{d_k})}{\sum_{j' \leq i} \exp(\langle q_i, k_{j'} \rangle / \sqrt{d_k})}.$$

- Weight  $\alpha_{ij}$ : How relevant is position  $j$  as context for position  $i$ ?
- Restriction to  $j \leq i$  (causal masking):  
The output at position  $i$  can be used to predict  $y_{i+1}$ .
- Multi-head attention: Several such layers in parallel, with separate weights.  
Context matters in different ways (grammar, reference, topic).

# Attention as kernel regression

- Recall the Nadaraya-Watson kernel regression estimator:

$$\hat{m}(x) = \frac{\sum_j K(x, X_j) \cdot Y_j}{\sum_j K(x, X_j)}.$$

- Attention has the same form:

Query  $q_i$  = evaluation point, keys  $k_j$  = regressors, values  $v_j$  = outcomes, kernel  $K(q, k) = \exp(\langle q, k \rangle / \sqrt{d_k})$ .

- Differences to standard kernel regression:

- The metric  $(W^q, W^k)$  and the outcomes  $(W^v)$  are learned.
- Weights depend on content, not on distance in the text.

Contrast to time series models, where weights depend only on  $|i - j|$ .

- Word order still matters. Olmo 3 rotates  $q_i$  and  $k_j$  by angles proportional to  $i$  and  $j$ , so that weights also depend on  $j - i$ .

# Transformer blocks

- Each block combines attention and an MLP, with residual connections:

$$\begin{aligned}u &= h + \text{MultiHeadAttention}(\text{Norm}(h)), \\ h' &= u + \text{MLP}(\text{Norm}(u)).\end{aligned}$$

- MLP: A two-layer neural net, applied to each position separately.
- Norm: Rescaling of vectors, for numerical stability.
- Residual connections: Each block *adds* to a running “residual stream,” rather than replacing it.

## Summary: The architecture of Olmo 3 7B

| Component               | Operation                                   | Parameters                         | Olmo 3 7B |
|-------------------------|---------------------------------------------|------------------------------------|-----------|
| Embedding               | $h_t^0 = e_{y_t}$                           | $ \mathcal{V}  \cdot d$            | 0.4B      |
| Attention ( $L$ blocks) | $o_i = \sum_{j \leq i} \alpha_{ij} W^v h_j$ | $4d^2$ per block                   | 2.1B      |
| MLP ( $L$ blocks)       | two-layer net, each position                | $\approx 8d^2$ per block           | 4.3B      |
| Output                  | $\text{softmax}(W^{\text{out}} h_t^L)$      | $ \mathcal{V}  \cdot d$            | 0.4B      |
| Total                   |                                             | $\approx 12Ld^2 + 2 \mathcal{V} d$ | 7.3B      |

Olmo 3 7B:  $d = 4096$ ,  $L = 32$  blocks, 32 attention heads,  $|\mathcal{V}| \approx 100,000$ .

- Training compute: About 6 operations per parameter per token.  
For 5.9T tokens:  $6 \cdot 7.3 \cdot 10^9 \cdot 5.9 \cdot 10^{12} \approx 2.6 \cdot 10^{23}$  operations.
- Attention compares all pairs of positions: Cost quadratic in context length.  
OLMo Hybrid (2026) replaces 3 of 4 attention layers by recurrent layers.

## Connections to earlier parts of class

- Decision theory: Log loss as a proper scoring rule.  
The penalized objective and its Bayesian interpretation.
- Neural nets: Softmax output layer, backpropagation, SGD.
- Online convex optimization: Single-pass SGD, online-to-batch conversion.
- Shrinkage: The winner's curse behind reward over-optimization.
- Kernel methods: Attention as Nadaraya-Watson regression.
- Bandits and RL: Learning from the model's own actions.  
Policy gradient, baselines, GRPO.

# Readings

*Jurafsky, D. and Martin, J. H. (2026). Speech and Language Processing, 3rd ed. draft, chapter 7.*

*Phuong, M. and Hutter, M. (2022). Formal algorithms for transformers. arXiv:2207.09238.*

*Karpathy A. (2026). microgpt. <https://karpathy.github.io/2026/02/12/microgpt/>*

*Lambert, N. (2026). Reinforcement Learning from Human Feedback. rlhf-book.com, chapters 4–8 and 14.*

*Rafailov, R. et al. (2023). Direct preference optimization: Your language model is secretly a reward model. NeurIPS.*