Machine Learning Notes 12.09

Notes Group

December 16, 2025

1 Review

1.1 Multi-Armed Bandits (MAB)

1.1.1 Problem Statement

Definition 1 (MAB instance). *A k-armed bandit instance consists of k unknown distributions $\nu_1, \dots, \nu_k$ with means $\mu(a) = \mathbb{E}_{X \sim \nu_a}[X]$. At each round $t = 1, \dots, T$:*

1. *the learner selects an arm $a_t \in [k]$;*
2. *the environment reveals a loss (or reward) $\ell_t(a_t) \sim \nu_{a_t}$;*
3. *the goal is to minimise the (expected) cumulative loss and, equivalently, the* ***regret*** *relative to the best fixed arm.*

Definition 2 (Regret). *Let $a^* = \arg\min_a \mu(a)$ be the optimal arm and $\Delta_a = \mu(a) - \mu(a^*) > 0$ the sub-optimality gap. The cumulative regret is*

$$R_T = \mathbb{E}\left[\sum_{t=1}^{T} \ell_t(a_t)\right] - T\mu(a^*) = \sum_{a:\Delta_a > 0} \Delta_a \cdot \mathbb{E}[N_T(a)],$$

where $N_T(a) = \sum_{t=1}^{T} \mathbb{I}\{a_t = a\}$ counts how many times arm a was pulled.

1.1.2 Exploration vs. Exploitation

- **Exploration:** gather information to reduce uncertainty about $\mu(a)$;
- **Exploitation:** choose the arm that currently looks best.

The UCB (Upper Confidence Bound) algorithm automatically balances the two by "being optimistic in the face of uncertainty".

1.2 Upper Confidence Bound (UCB)

1.2.1 Settings

Arms $1, \dots, k$ with unknown means $\mu(1), \dots, \mu(k)$. At round t we have

- $N_{t-1}(a)$: number of pulls of arm a up to round $t-1$;
- $M_{t-1}(a)$: empirical average loss of arm a up to round $t-1$.

Policy:

$$a_t = \arg\min_{a \in [k]} \left\{ M_{t-1}(a) - \sqrt{\frac{\ln T}{N_{t-1}(a)}} \right\}. \tag{1}$$

After observing $\ell_t(a_t)$, update $N_t(a_t)$ and $M_t(a_t)$.

1.2.2 Algorithm

Algorithm 1 UCB for losses (loss-minimisation)

```
1: Input: $k$, horizon $T$, confidence constant $c = 1$
2: Initialise $N_0(a) = 0$, $M_0(a) = 0$ for all $a \in [k]$
3: for $t = 1, \dots, T$ do
4:     if $\exists a : N_{t-1}(a) = 0$ then
5:         pick any such $a_t$                                ▷ forced exploration
6:     else
7:         $a_t = \arg\min_a \left\{ M_{t-1}(a) - c\sqrt{\frac{\ln T}{N_{t-1}(a)}} \right\}$
8:     end if
9:     incur loss $\ell_t(a_t)$ and observe it
10:    $N_t(a_t) = N_{t-1}(a_t) + 1$
11:    $M_t(a_t) = M_{t-1}(a_t) + \frac{\ell_t(a_t) - M_{t-1}(a_t)}{N_t(a_t)}$
12: end for
```

1.2.3 Regret Guarantee

Theorem 1 (Regret of UCB)**.** *With the choice $c = 1$ the expected regret satisfies*

$$R_T \le \sum_{a:\Delta_a > 0} \left(\frac{4 \ln T}{\Delta_a} + \Delta_a \right).$$

Hence $R_T = O\big(\sum_{\Delta_a > 0} \frac{\log T}{\Delta_a}\big)$ and, moreover, for any bandit instance the worst-case regret is

$$R_T = O(\sqrt{kT \log T}).$$

1.2.4 Worst-Case Regret: "Bad-Sub-Optimal-Arms" Argument

The inequality scribbled in the note

$$\Delta_a > 0 \quad \Longrightarrow \quad R_T = O\big(\sqrt{T \log T}\big)$$

is a short-hand for the following minimax statement.

Theorem 2 (Minimax rate of UCB)**.** *For any k-armed bandit instance with losses in* $[0, 1]$*,*

$$R_T \le 4\sqrt{kT \log T} + k.$$

Hence the worst-case regret over all *possible gaps satisfies*

$$\sup_{\textit{instances}} R_T = O(\sqrt{kT \log T}).$$

Proof. Start from the gap-dependent bound

$$R_T \le \sum_{a:\Delta_a > 0} \Big(\frac{4 \log T}{\Delta_a} + \Delta_a \Big).$$

Split the sum into two groups:

1. **Small-gap arms:** $\Delta_a \le \varepsilon$ (possibly including the optimal arm in a tie). Their contribution is at most $(T \cdot \varepsilon)$, because each extra pull costs at most ε and there are at most T pulls.
2. **Large-gap arms:** $\Delta_a > \varepsilon$. Their contribution is at most $\sum_a \frac{4 \log T}{\Delta_a} \le \frac{4k \log T}{\varepsilon}$.

Optimise the free parameter ε:

$$R_T \le T\varepsilon + \frac{4k\log T}{\varepsilon}.$$

Taking $\varepsilon = \sqrt{\frac{4k\log T}{T}}$ yields

$$R_T \le 2\sqrt{4kT\log T} = 4\sqrt{kT\log T}.$$

Adding the unavoidable $+k$ term from forced exploration gives the advertised bound. $\square$

Interpretation of the scribble: Even if *every* sub-optimal arm is "bad" (i.e. all Δ_a are arbitrarily small), the worst-case regret cannot grow faster than $\sqrt{T\log T}$; only the *number* of arms k and the horizon T matter, not the concrete gaps.

1.3 Thompson Sampling

Maintain a posterior distribution for each mean reward: $\mu(a) \sim \mathcal{P}_{a,t}$. At round t:

1. Draw a sample $\theta_{a,t} \sim \mathcal{P}_{a,t}$ for every arm a.
2. Play $a_t = \arg\max_a \theta_{a,t}$ (or $\arg\min$ for losses after flipping the sign).
3. Observe reward r_t and update $\mathcal{P}_{a_t,t}$ with Bayes' rule.

For Bernoulli rewards $\text{Beta}(\alpha_a, \beta_a)$ the update is simply

$$\alpha_{a_t} \leftarrow \alpha_{a_t} + r_t, \qquad \beta_{a_t} \leftarrow \beta_{a_t} + (1 - r_t).$$

Regret bound: $R_T = O\big(\sum_a \frac{\log T}{\Delta_a}\big)$, matching UCB up to constants, but TS often exhibits lower empirical regret because posterior sampling performs smooth probability matching instead of deterministic optimism.

2 Expressiveness

2.1 Classic Machine Learning Models

In classical machine learning theory, **expressiveness** mainly concerns the choice of hypothesis classes. The central question is whether the function class used for training is rich enough to approximate the true underlying model. Two key aspects are typically studied:

- **Approximation ability**: whether the model class can approximate target functions.
- **Approximation efficiency**: how the model size grows as the approximation error decreases.

Some basic conclusions are as follows.

- **Shallow Neural Networks**:

  Universal Approximation Theorem: Single hidden layer can approximate any continuous function with compact support to arbitrary accuracy, when the width goes to infinity.[Hornik et al., 1989]
  Pf : The theorem can be proved by directly applying *Stone-Weierstrass Theorem.*
  The **Universal Approximation Theorem** states that a feedforward neural network with a single hidden layer (i.e., three layers: input–hidden–output) and suitable activation functions can approximate any continuous function on a compact set arbitrarily well. This result demonstrates the strong approximation power of shallow networks.

  However, the approximation rate of shallow networks can be poor. In the worst case, achieving an approximation error of order ε may require an exponential number of hidden units, i.e.,

$$\text{network size} = O\left(\exp\left(\frac{1}{\varepsilon}\right)\right).$$

  This limits their practical efficiency for high-precision approximation.

- **Deep Neural Networks**:

  Deep neural networks (DNNs) significantly improve approximation efficiency. Informally, if the network width is greater than data dimension, deep neural networks also enjoy universal approximation properties. And more importantly, for many function classes, deep networks can achieve better approximation rates than shallow ones.
  Universal Approximation Theorem (revised): A network of infinite depth with a hidden layer of size d+1 neurons, where d is the dimension of the input space, can approximate any continuous function. [Lu et al., NIPS 2017]

- **Other Neural Architectures**:

  Various modern architectures such as **multilayer perceptrons (MLPs)**, **residual networks (ResNets)**, and **Transformers** have been shown to possess universal approximation properties under appropriate assumptions. Structural constraints, such as certain sparse attention patterns in **Sparse Transformers**, may reduce expressiveness and lead to loss of universal approximation properties.
  Universal Approximation Theorem (further revised): ResNet with a single hidden unit and infinite depth can approximate any continuous function. [Lin and Jegelka, NIPS 2018]

More Visual Example and Proof of UAT: http://neuralnetworksanddeeplearning.com/chap4.html

2.2 Large Language Models (LLMs)

Compared to classical models, Large Language Models are typically based on auto-regressive Transformer architectures and emphasize *reasoning*.

The **Chain-of-Thought (CoT)** framework highlights how explicitly generated intermediate reasoning steps can enhance the effective expressiveness of LLMs. This topic will be discussed in more detail in later lectures.

3 Optimization

3.1 Optimization (Classic)

In classical machine learning or deep learning, optimization refers to the process of minimizing the training loss.

3.1.1 Convergence

In high-dimensional parameter spaces, optimization algorithms usually converge to (Approximate) Stationary Points (i.e., points with gradient $\nabla L \approx 0$).

- **Saddle Point vs. Local Minimum**: In high-dimensional spaces, convergence points are **almost certainly saddle points** rather than local minima.
- **Brief Proof/Intuition**: Consider the eigenvalues of the Hessian matrix (H).
  - For a stationary point to be a local minimum, all eigenvalues of H must be non-negative (positive definite or positive semi-definite).
  - In very high dimensions d, the probability that randomly generated eigenvalues are all non-negative is extremely low (decreases exponentially).
  - Therefore, eigenvalues typically exhibit a mixture of positive and negative values, making the point highly likely to be a saddle point.

3.1.2 Rate of Convergence

For detailed theoretical derivations and bound analyses, refer to other optimization-related courses in the curriculum.

3.2 Optimization in Large Language Models (LLMs)

In the context of large language models (LLMs), discussing traditional "optimization" concepts—especially asymptotic convergence—is often of limited significance.

Key Insight

Single Pass / Few Pass Training
Current LLM training typically follows a **Few Pass Training** or even **Single Pass Training** paradigm (i.e., Epoch ≈ 1).

This means the model does not iteratively optimize on the same dataset until $t \to \infty$ for convergence analysis as in traditional optimization problems. Instead, the model primarily "traverses" the data rather than "converging" on a fixed dataset.

Implications:

- The training process emphasizes data coverage and exposure over asymptotic convergence to a local optimum.
- Optimization focuses more on effective gradient steps and batch utilization rather than convergence guarantees.
- Empirical performance is prioritized over theoretical convergence rates in most practical LLM training scenarios.

4 Generalization

4.1 Classic

Classic generalization theory primarily focuses on bounding the difference between empirical risk and true risk. It can be categorized into two main perspectives:

1. **VC Theory (Model Complexity):** This is an *algorithm-independent* approach that focuses on the inherent capacity of the hypothesis space. It suggests that the generalization gap is bounded by the model's complexity (represented by the VC dimension) and the number of training samples n:
$$\text{Generalization Gap} \leq \sqrt{\frac{VC}{n}}$$
In this framework, a more complex model (higher VC dimension) requires more data to guarantee generalization.
2. **Algorithm-dependent Generalization:** This perspective considers how the specific optimization process affects the final model's ability to generalize.
    - **Example: Stochastic Gradient Descent (SGD).** The **stochastic** nature of the gradient updates introduces "noise" during training. This stochastic gradient helps the model avoid over-fitting to specific data points and encourages the optimization to find flatter minima, which ultimately *helps generalization.*

Note: While these classic theories provide a solid foundation, they often become less predictive for Large Language Models (LLMs), where the number of parameters far exceeds the number of training samples, leading to the "meaningless" bound issue mentioned in the lecture.

4.2 LLMs

LLMs don't excel at generalization. Their ability to address everyday needs primarily stems from the extensive coverage of their training datasets, which reduces the need for generalization when performing specific tasks.

The limited generalization capacity of LLMs can be demonstrated in three aspects:

1. Creativity: LLMs show a significant decline in performance when dealing with less frequent content in their training data, exhibiting a limited capacity for innovation in scientific research.
2. Data efficiency: Although LLMs possess a knowledge reserve far exceeding that of humans, their problem-solving ability in specialized domains still falls short of human-level performance.
3. Long context processing: Since the training data predominantly consist of Short context, whereas inference often requires handling long context, this inconsistency in data distribution also impacts the capabilities of LLMs.

5 Brief Intro

5.1 Chain of Thought

5.1.1 Initial Motivation for CoT

The Chain of Thought (CoT) technique was initially proposed to enhance the reasoning capabilities of LLMs.Inspired by how human break down problems into smaller,thereby improving accuracy in problem-solving,it aims to mimic human-like reasoning tasks that require multi-step logical deduction by encouraging the model to generate intermediate reasoning steps before arriving at a final answer.

Formally, for an input query x, instead of directly predicting the output y, the model generates a sequence of intermediate steps $s_1, s_2, \ldots, s_n$ that lead to y. This can be represented as:

$$P(y|x) = \sum_{s} P(s|x)P(y|s,x)$$

where s denotes the chain of thought. In practice, CoT is often implemented through prompt engineering or fine-tuning, where models are trained to produce step-by-step explanations.

5.1.2 CoT in Current Large Models

CoT has evolved from a prompting strategy to an integral component in SOTA LLMs.Key developments are listed below:

- **Prompt**: Initially, CoT was introduced via few-shot prompting in models, where examples with reasoning steps were included in prompts.
- **Fine-tuning**: Models like Codex were fine-tuned on CoT-style datasets to inherent reasoning abilities.
- **Integration into Architectures**: Models like PaLM and GPT-4, incorporate CoT implicitly through scaled-up training and RLHF, enabling more robust reasoning.
- **Automated CoT**: Techniques like Auto-CoT automate the generation of reasoning chains, reducing reliance on manual exemplars.
- **Specialized Models**: Models like Minerva (for mathematics) leverage CoT for domain-specific reasoning.

5.2 Position Encoding

Position encoding is crucial in Transformer-based models to inject information about the order of tokens in a sequence, since Transformers(or self-attn modules,more exactly) are inherently permutation-invariant.Two prominent methods are sinusoidal (trigonometric) position encoding and rotary position encoding.

2.1 Trigonometric Position Encoding

Introduced in the original Transformer paper, trigonometric position encoding uses sine and cosine functions of different frequencies to encode positions. For a position *pos* and a dimension i, the encoding is defined as:

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d_{\text{model}}}}\right)$$
$$PE_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{2i/d_{\text{model}}}}\right)$$

where d_{model} is the model's embedding dimension. This encoding strategy can make sure that each position gets a unique encoding and stability during training time due to the fact that values are between -1 and 1.

The trigonometric allow the model to learn relative positions due to the linear transformations: $sin(pos+k)$ and $cos(pos+k)$ can be expressed as linear functions of $sin(pos)$ *and* $cos(pos)$.

$$\sin(pos+k) = \sin(pos)\cos(k) + \cos(pos)\sin(k)$$
$$\cos(pos+k) = \cos(pos)\cos(k) - \sin(pos)\sin(k)$$

This shows that $\sin(pos+k)$ and $\cos(pos+k)$ are linear combinations of $\sin(pos)$ and $\cos(pos)$. In matrix form, this transformation can be written as:

$$\begin{bmatrix}\sin(pos+k)\\ \cos(pos+k)\end{bmatrix} = \begin{bmatrix}\cos(k) & \sin(k)\\ -\sin(k) & \cos(k)\end{bmatrix}\begin{bmatrix}\sin(pos)\\ \cos(pos)\end{bmatrix}$$

In practice, these encodings are added to the token embeddings before input to the Transformer layers:

$$\mathbf{h}^{(0)} = \mathbf{x} + \mathbf{PE}$$

where $\mathbf{x}$ is the token embedding matrix and $\mathbf{PE}$ is the position encoding matrix.

2.2 Rotary Position Encoding (RoPE)

Rotary Position Encoding (RoPE), proposed in RoFormer, encodes absolute position information via rotation matrices in complex space. It operates on query and key vectors in self-attention by rotating them based on their positions. This preserves relative position dependencies inherently.

Given a query or key vector $\mathbf{q}_m$ at position m, and a vector $\mathbf{k}_n$ at position n, RoPE applies a rotation matrix $\mathbf{R}_{\theta,m}$ to incorporate position. In complex form, for each head dimension, let $\mathbf{q}_m^{(j)}$ be the j-th component of $\mathbf{q}_m$, represented as a complex number. The rotation is defined as:

$$\mathbf{q}_m^{(j)} e^{im\theta_j} \quad \text{and} \quad \mathbf{k}_n^{(j)} e^{in\theta_j}$$

where $\theta_j = 10000^{-2j/d}$, with d being the dimension per attention head. The dot product in attention becomes:

$$\mathbf{q}_m \cdot \mathbf{k}_n = \text{Re}\left(\sum_j \mathbf{q}_m^{(j)} \mathbf{k}_n^{(j)*} e^{i(m-n)\theta_j}\right)$$

This shows that the attention score depends only on the relative position $m-n$, capturing relative information effectively.

In matrix form, for a d-dimensional vector $\mathbf{x} = [x_1, x_2, \ldots, x_d]$, RoPE groups dimensions into pairs and applies a rotation. For each pair (x_{2i}, x_{2i+1}), the rotated version is:

$$\begin{bmatrix}x_{2i}\\ x_{2i+1}\end{bmatrix} \otimes \begin{bmatrix}\cos(m\theta_i) & -\sin(m\theta_i)\\ \sin(m\theta_i) & \cos(m\theta_i)\end{bmatrix} = \begin{bmatrix}x_{2i}\cos(m\theta_i) - x_{2i+1}\sin(m\theta_i)\\ x_{2i}\sin(m\theta_i) + x_{2i+1}\cos(m\theta_i)\end{bmatrix}$$

where $\theta_i = 10000^{-2i/d}$.

5.3 Autoregressive Transformer

Large Language Models are most commonly built on **autoregressive** (AR) Transformers. The key idea is to model a sequence by predicting the next token conditioned on all previous tokens, and generate text *from left to right.*

5.3.1 Autoregressive Factorization

Given a token sequence $x_{1:T} = (x_1, \dots, x_T)$, the AR assumption factorizes the joint likelihood as:

$$p(x_{1:T}) = \prod_{t=1}^{T} p(x_t \mid x_{<t}),$$

where $x_{<t} = (x_1, \dots, x_{t-1})$. In language modeling, this corresponds to learning **next-token prediction**.

5.3.2 Training Objective (Maximum Likelihood)

Training is typically done via maximum likelihood estimation (MLE) with **teacher forcing**, i.e. at step t the model conditions on the ground-truth prefix $x_{<t}$. The negative log-likelihood (cross-entropy) loss is:

$$\mathcal{L}(\theta) = -\sum_{t=1}^{T} \log p_\theta(x_t \mid x_{<t}).$$

Equivalently, if the model outputs logits $\mathbf{z}_t \in \mathbb{R}^{|\mathcal{V}|}$ and $p_\theta(x_t \mid x_{<t}) = \mathrm{softmax}(\mathbf{z}_t)$, then the loss is the standard token-level cross entropy.

5.3.3 Causal Self-Attention (Why Transformer can be Autoregressive)

To ensure the model cannot "peek" at future tokens, AR Transformers use a **causal mask** in self-attention. For queries Q, keys K, values V, the masked attention can be written as:

$$\mathrm{Attn}(Q, K, V) = \mathrm{softmax}\left(\frac{QK^\top}{\sqrt{d_k}} + M\right) V,$$

where the mask M satisfies:

$$M_{t,j} = \begin{cases} 0, & j \le t, \\ -\infty, & j > t, \end{cases}$$

so token t only attends to positions $\le t$. This makes each conditional $p(x_t \mid x_{<t})$ well-defined.

5.3.4 Inference / Generation

At inference time, the model generates tokens sequentially:

$$x_t \sim p_\theta(\cdot \mid x_{<t}),$$

until an end-of-sequence token is produced or a length limit is reached. Common decoding strategies include greedy decoding ($\arg\max$), sampling, top-k / nucleus (top-p) sampling, and beam search. In practice, **KV-cache** is often used to reuse previously computed keys/values, reducing repeated computation during long generation.