BAYESIAN LOW-RANK ADAPTATION FOR LARGE LANGUAGE MODELS

Type	Research Paper
Comments	Bayesian LoRA for LLM
Date	@June 30, 2025

Motivation

- **Overconfidence in fine-tuned LLMs.**

  While fine-tuning large language models is essential both for downstream tasks and for building instruction-following agents, it often induces severe overconfidence—models assign excessive probability to their predictions, even when they're wrong. This miscalibration is especially dangerous in safety-critical settings (e.g., medical diagnosis, finance, experimental design) and when only limited fine-tuning data is available .

- **The promise and challenge of Bayesian uncertainty.**

  Bayesian deep learning naturally addresses overconfidence by modeling posterior uncertainty, but naively applying it to all billions of LLM parameters is computationally infeasible. At the same time, parameter-efficient fine-tuning methods like LoRA train only a small, low-rank adapter on top of a frozen backbone—drastically reducing trainable parameters. This suggests a sweet spot: perform a **post-hoc** Bayesian (Laplace) approximation **only** over the low-rank LoRA weights, thereby gaining uncertainty estimates without altering standard fine-tuning pipelines or incurring prohibitive costs.

Compared to previous works?

- **Bayesian deep learning for language models**

Earlier Bayesian treatments focused almost exclusively on large-scale **pre-training**, where the benefits of Bayesian inference are muted by enormous datasets and reasonably well-calibrated pretrained models . In contrast, Laplace-LoRA targets the **fine-tuning** setting, where calibration is known to degrade substantially (even after instruction tuning) and data can be scarce.

- **Parameter-efficient fine-tuning (PEFT) methods**

  Methods like LoRA and other PEFT approaches train only a small "adapter" on top of a frozen backbone. Until now, there has been no Bayesian inference method designed specifically for these adapters—Laplace-LoRA is the first to bring a post-hoc Laplace approximation to **only** the low-rank LoRA weights, preserving PEFT's efficiency.

- **Bayesian approximations over attention weights vs. parameters**

  Previous work explore Bayesian priors over **attention weights**, which (a) require redefining training pipelines and (b) entail reasoning over up to a billion attention parameters—whereas Laplace-LoRA remains **post-hoc**, keeps all pipelines unchanged, and reduces the inference problem from billions of weights to only ~6 million LoRA parameters.

- **Laplace approximations at scale**

  Classical Laplace methods for neural networks have rarely been applied to language models, and then only at the final layer of much smaller models. Laplace-LoRA scales this idea to **all** LoRA adapters in models 100× larger by using Kronecker-factored, low-rank Hessian approximations.

- **Regularization-based calibration**

  Techniques such as Mixout, and KL/L2 regularization on BERT outputs improve calibration by altering the fine-tuning objective. These are **orthogonal** to Laplace-LoRA, since Laplace-LoRA does **not** change the MAP weights—it simply wraps a Bayesian uncertainty estimate around them.

Background

Instead of updating the full $W_0 \in \mathbb{R}^{n_{\text{out}} \times n_{\text{in}}}$—which would be $n_{\text{out}} \times n_{\text{in}}$ trainable parameters—we keep W_0 frozen and learn only a **low-rank** "correction" ΔW.

Concretely:

1. **Layer output before adaptation**

   $h = W_0\, a,$

   where $a \in \mathbb{R}^{n_{\text{in}}}$ is the input and $h \in \mathbb{R}^{n_{\text{out}}}$ the output.

2. **Introduce a perturbation**

   We set

   $W = W_0 + \Delta W, \quad \text{so} \quad h = W\, a = W_0\, a + \Delta W\, a.$

3. **Low-rank factorization of ΔW**

   Instead of learning the full ΔW, we write

   $\Delta W = B\, A,$

   with

   $B \in \mathbb{R}^{n_{\text{out}} \times n_{lr}}, \quad A \in \mathbb{R}^{n_{lr} \times n_{\text{in}}}, \quad n_{lr} \ll n_{\text{in}},\, n_{\text{out}}.$

   Then

   $h = W_0\, a + B\,(A\, a).$

4. **Parameter savings**

   - Full fine-tuning would train $n_{\text{out}} \times n_{\text{in}}$ parameters.
   - LoRA only trains the two small matrices A and B, i.e. $n_{lr}(n_{\text{in}} + n_{\text{out}})$ parameters, which is typically orders of magnitude smaller when n_{lr} is, say, 8 or 16.

5. **Memory and compute benefits**

   - We only compute gradients for A, B, not for the giant W_0.
   - We save both GPU memory (for optimizer states) and computation, yet still adapt the model effectively.

Laplace Approximation

Bayesian Inference

$P(\theta \mid X, y) \;\propto\; P(y \mid X, \theta)\; P(\theta)$

- **Posterior** $P(\theta \mid X, y)$: updated belief about the model parameters θ after having observed inputs X (a batch of token sequences) and their targets y (either class labels or next tokens).
- **Likelihood** $P(y \mid X, \theta)$: the probability that the model with parameters θ assigns to seeing targets y given inputs X. In a classification task this is typically a softmax-categorical distribution; in language modeling it's the conditional probability of the next token.
- **Prior** $P(\theta)$: belief about θ before seeing any data. In this work it's chosen to be an isotropic Gaussian distribution

  $$P(\theta) = \mathcal{N}\big(0,\, \lambda^{-1} I\big),$$

  i.e. zero mean and precision λ.
  This is usually intractable.
  Laplace approximation turns an intractable Bayesian posterior into a tractable Gaussian around the MAP solution:

  $$\mathcal{L}(y, X;\, \theta) = \log P\big(y \mid X, \theta\big) + \log P(\theta) = \log P\big(\theta \mid X, y\big) + \text{const}$$

  $$\theta_{\text{MAP}} = \arg\max_\theta \, \mathcal{L}(y, X;\, \theta)$$

  In practice this is exactly what we get by standard fine-tuning (maximizing the posterior).

  Quadratic (second-order) Taylor expansion around the MAP

  $$\mathcal{L}(y, X; \theta) \approx \mathcal{L}(y, X; \theta_{\text{MAP}}) \; - \; \tfrac{1}{2}(\theta - \theta_{\text{MAP}})^\top \big[\nabla^2_\theta \mathcal{L}(y, X; \theta)\big]_{\theta_{\text{MAP}}} (\theta - \theta_{\text{MAP}}).$$

  Since we've approximated the log-joint by a **quadratic**, the posterior becomes Gaussian.

  Gaussian posterior approximation

  $$P(\theta \mid \mathcal{D}) \; \approx \; \mathcal{N}\big(\theta; \theta_{\text{MAP}},\, \Sigma\big), \quad \Sigma \; = \; -\big[\nabla^2_\theta \mathcal{L}(y, X; \theta)\big]^{-1}_{\theta_{\text{MAP}}}.$$

  Concretely, because

  $$\nabla^2_\theta \mathcal{L} = \nabla^2_\theta \log P(y \mid X, \theta) + \nabla^2_\theta \log P(\theta),$$

  and the prior's Hessian is just $-\lambda I$, we end up with

  $$\Sigma = -\big[\nabla^2_\theta \log P(y \mid X, \theta)\big]^{-1}_{\theta_{\text{MAP}}} \; + \; \lambda^{-1} I,$$

i.e. the inverse of the Hessian of the negative log-joint at the MAP.

- **Fisher as stand-in for Hessian**

  To ensure positive definiteness, instead of computing the exact Hessian of the log-likelihood

  $\nabla^2_\theta \log P(y \mid X, \theta)$, they replace it with the **Fisher Information**

  $\mathbb{E}_{y\sim P(y|x_n,\theta)}\Big[\nabla_\theta \log P(y \mid x_n, \theta)\ \nabla_\theta \log P(y \mid x_n, \theta)^\top\Big]$.

  Under mild conditions, Fisher is positive semi-definite and often a good proxy for the curvature.

- **Kronecker-factored approximation (KFAC)**

  Even restricting to only LoRA's ~6M parameters, F would still be a 6M×6M matrix. So they further exploit the fact that each LoRA update lives inside a **linear** layer. For each adapter-layer e, one can write its block of Fisher as

  $F_e = \sum_{n=1}^{N} \mathbb{E}\big[(a_e\, a_e^\top) \otimes (g_e\, g_e^\top)\big]$,

  where

  - a_e is that layer's input activation,
  - $g_e = \nabla_{b_e} \log P(y \mid X, \theta)$ is the gradient w.r.t. that layer's output,
  - and $\otimes$ is the Kronecker product.

    This **KFAC** structure lets them store and invert much smaller matrices for $a_e a_e^\top$ and $g_e g_e^\top$ instead of the full block.

- **Linearized model and posterior over logits**

  - We approximate the network $f_\theta(x)$ by its first-order Taylor expansion around the MAP point θ_{MAP}:

    $f_\theta(x) \approx f_{\theta_{\mathrm{MAP}}}(x)\ +\ \nabla_\theta f_{\theta_{\mathrm{MAP}}}(x)\,(\theta - \theta_{\mathrm{MAP}})$.

  - Because θ is now Gaussian (mean θ_{MAP}, covariance Σ from the inverse Fisher + prior), the output logits become Gaussian as well:

    $f(x) \sim \mathcal{N}\big(f_{\theta_{\mathrm{MAP}}}(x),\, \Lambda\big)$,

    with Λ computed in closed form.

- We can then sample from this Gaussian over logits to get uncertainty—just draw $\xi \sim \mathcal{N}(0, I)$ and compute

  $$\tilde{f}(x) = f_{\theta_{\text{MAP}}}(x) \; + \; L\,\xi,$$

  where $LL^T = \Lambda$.

- **Why this matters**
  - We never have to re-train or hold out a validation set for calibration—everything is "post-hoc."
  - We only pay the cost of accumulating a few Kronecker factors per layer (and inverting them), not of dealing with the full billions×billions Hessian.
  - The result is a lightweight, efficient Laplace approximation over your low-rank adapters that produces well-calibrated uncertainty estimates.

Methodology

Laplace-LoRA makes a full Laplace approximation over the LoRA adapters both tractable and memory-efficient:

- **Adapter as two linear layers.** Rather than viewing the low-rank update $\Delta W = BA$ as one single "big" low-rank weight matrix, they treat it as two consecutive linear layers with weight matrices

  $$A \in \mathbb{R}^{n_{lr} \times n_{in}}, \quad B \in \mathbb{R}^{n_{out} \times n_{lr}},$$

  so that the usual LoRA forward is $h = W_0 a + B\,(A\,a)$.

- **Kronecker-factored curvature with low-rank compression.** The Laplace approximation requires the Hessian (or Fisher) of the log-likelihood, which for a single LoRA layer has a Kronecker-factor structure $\sim (AA^T) \otimes (GG^T)$. One of these factors is size $d \times d$ (with $d = 4096$ in LLaMA2-7B), which would erase all memory savings if materialized explicitly. Instead, they approximate that large factor by a low-rank proxy of rank n_{kfac}, chosen independently of the adapter rank n_{lr}.

- **Incremental, end-to-end low-rank pipeline.** To preserve LoRA's tiny memory footprint and plug-and-play workflow, Laplace-LoRA performs all three steps in low-rank form (never forming a full $d \times d$ matrix):

  1. **Incremental factor computation** (never build the full factor first),

2. **Marginal-likelihood optimization** under the low-rank Laplace, and
3. **Low-rank linearized predictions** by propagating uncertainty through the network's Jacobian without full Hessian inversion.

Together, these design choices let Laplace-LoRA wrap a Bayesian Gaussian posterior around just the few-million LoRA parameters—yielding well-calibrated uncertainty estimates with only a few percent extra memory and compute—while leaving every other part of the fine-tuning pipeline unchanged.

Experimental Results

In-distribution fine-tuning and evaluation

- **Setup:** LLaMA2-7B is fine-tuned with LoRA on six tasks (Winogrande-S/M, ARC-Challenge/Easy, OBQA, BoolQ) for 10 k steps (batch size 4), saving checkpoints every 1 k steps. Post-hoc Laplace approximations (LA on all adapter weights; LLLA on just the output layer) are applied at each checkpoint.
- **Baselines:** MAP (standard LoRA), MC dropout, checkpoint ensembles, deep ensembles, and temperature scaling.
- **Metrics & Findings:** Across all six tasks, LA maintains the same accuracy as MAP but cuts expected calibration error (ECE) roughly in half (from ~30% down to ~7%) and lowers negative log-likelihood (NLL) by ~0.5 nats versus MAP, outperforming all baselines; LLLA yields smaller gains .

Evaluations under distribution shift

- **Smaller shifts:** Models fine-tuned on OBQA are tested on ARC-C and ARC-E.
- **Larger shifts:** Same checkpoint is evaluated on four MMLU subjects (CS, Eng, Law, Health).
- **Results:** LA delivers substantial ECE and NLL improvements over MAP and other post-hoc methods while keeping accuracy unchanged (or slightly improved in CS), for both small and large distribution shifts .

Memory and runtime cost

- Adding Laplace-LoRA on top of standard LoRA incurs only a **1–5% memory** overhead and about **10% extra compute** when accumulating factors at every 1

k steps (practically reducible to ~1% if done once) . This confirms that the low-rank KFAC Laplace pipeline preserves LoRA's efficiency.