juliacon

Effortless Bayesian Deep Learning in Julia through Laplace

Patrick Altmeyer[1]

[1]Delft University of Technology

ABSTRACT

Treating deep neural networks probabilistically comes with numerous advantages including improved robustness and greater interpretability. These factors are key to building Artificial Intelligence (AI) that is trustworthy. A drawback commonly associated with existing Bayesian methods is that they increase computational costs. Recent work has shown that Bayesian deep learning can be effortless through Laplace approximation. We propose a light-weight Julia package, `LaplaceRedux.jl` that implements this new approach for deep neural networks trained in `Flux.jl`.

Keywords

Julia, Probabilistic Machine Learning, Laplace Approximation, Deep Learning, Artificial Intelligence

1. Background

Over the past decade, Deep Learning (DL) has arguably been one of the dominating subdisciplines of Artificial Intelligence. Despite the tremendous success of deep neural networks, practitioners and researchers have also pointed to a vast number of pitfalls that have so far inhibited the use of DL in safety-critical applications. Among other things, these pitfalls include a lack of adversarial robustness [5] and an inherent opaqueness of deep neural networks, often described as the black-box problem.

In deep learning, the number of parameters relative to the size of the available data is generally huge [11]: "[...] deep neural networks are typically very underspecified by the available data, and [...] parameters [therefore] correspond to a diverse variety of compelling explanations for the data." A scenario like this very much calls for treating model predictions probabilistically [11]. It is therefore not surprising that interest in Bayesian deep learning has grown in recent years as researchers have tackled the problem from a wide range of angles including MCMC (see `Turing`), Mean Field Variational Inference [2], Monte Carlo Dropout [4] and Deep Ensembles [7]. Laplace Redux [6, 3] is one of the most recent and promising approaches to Bayesian neural networks (BNN).

2. Laplace Approximation for Deep Learning

Let $\mathcal{D} = \{x, y\}_{n=1}^{N}$ denote our feature-label pairs and let $f(x;\theta) = y$ denote some deep neural network specified by its parameters θ. We are interested in estimating the posterior predictive distribution given by the following Bayesian model average (BMA):

$$p(y|x,\mathcal{D}) = \int p(y|x,\theta)p(\theta|\mathcal{D})d\theta \tag{1}$$

To do so we first need to compute the weight posterior $p(\theta|\mathcal{D})$. Laplace Approximation (LA) relies on the fact that the second-order Taylor expansion of this posterior amounts to a multivariate Gaussian $q(\theta) = \mathcal{N}(\hat{\mu}, \hat{\Sigma})$ centred around the maximum a posteriori (MAP) estimate $\hat{\mu} = \hat{\theta} = \arg\max_\theta p(\theta|\mathcal{D})$ with covariance equal to the negative inverse Hessian of our loss function evaluated at the mode $\hat{\Sigma} = -(\hat{\mathcal{H}}|_{\hat{\theta}})^{-1}$.

To apply Laplace in the context of deep learning, we can train our network in the standard way by minimizing the negative log-likelihood $\ell(\theta) = -\log p(y|x,\mathcal{D})$. To obtain Gaussian LA weight posterior we then only need to compute the Hessian evaluated at the obtained MAP estimate.

Laplace Approximation itself dates back to the 18th century, but despite its simplicity, it has not been widely used or studied by the deep learning community until recently. One reason for this may be that for large neural networks with many parameters, the exact Hessian computation is prohibitive. One can rely on linearized approximations of the Hessian, but those still scale quadratically in the number of parameters. Fortunately, recent work has shown that block-diagonal factorizations can be successfully applied in this context [10].

Another reason why LA may have been neglected in the past is that early attempts at using it for deep learning failed: simply sampling from the Laplace posterior to compute the exact BNN posterior predictive distribution in Equation 1 does not work when using approximations for the Hessian [8]. Instead, we can use a linear expansion of the predictive around the mode as demonstrated by Immer et al. (2020) [6]. Formally, we locally linearize our network,

$$f_{\text{lin}}^{\hat{\theta}}(x;\theta) = f(x;\hat{\theta}) + \mathcal{J}_\theta(\theta - \hat{\theta}) \tag{2}$$

which turns the BNN into a Bayesian generalized linear model (GLM) where $\hat{\theta}$ corresponds to the MAP estimate as before. The corresponding GLM predictive,

$$p(y|x,\mathcal{D}) = \mathbb{E}\left[p(y|f_{\text{lin}}^{\hat{\theta}}(x;\theta_n))\right], \quad \theta_n \sim q(\theta) \tag{3}$$

has a closed-form solution for regression problems. For classification problems it can be approximated using (extended) probit approximation [3].

Immer et al. (2020) [6] provide a much more detailed exposition of the above with a focus on theoretical underpinnings and intuition. Daxberger et al. (2021) [3] introduce Laplace Redux from more of an applied perspective and present a comprehensive Python implementation: laplace.

3. `LaplaceRedux.jl` — a Julia implementation

The `LaplaceRedux.jl` package is intended to make this new methodological framework available to the Julia community. It is interfaced with the popular deep learning library, `Flux.jl`.
Using just a few lines of code the package enables users to compute and apply Laplace Redux to their pre-trained neural networks. A basic usage example is shown in Code 1: the `Laplace` function simply wraps the Flux neural network `nn`. Since the underlying model is a classifier, we need to specify the likelihood accordingly. The returned instance is then fitted to the data using the generic `fit!` method. Note that the `fit!` method also accepts a `DataLoader` as its second positional argument and mini-batch training is supported.

Code 1: Fitting a pre-trained neural network to data using Laplace Redux.

```
la = Laplace(nn; likelihood=:classification)
fit!(la, data)
```

The `la` object is a mutable and callable struct that wraps the pre-trained neural networks along with hyperparameters relevant to the Laplace approximation. Simply calling the instance with new data as in Code 2 will generate GLM predictions according to Equation 3. In the classification case, softmax outputs are returned by default following the convention in the Python implementation, but this can be changed using the `predict_proba` keyword argument. It is also possible to recover the original MAP estimate directly by setting the `link_approx` keyword argument to `:plugin`.

Code 2: Predictions using the fitted Laplace Redux instance.

```
la(X)                          # GLM predictions
la(X; predict_proba=false)     # no softmax
la(X; link_approx=:plugin)     # MAP predictions
```

Additional methods can be used to optimize the prior precision λ and to visualize the predictions (Code 3). The `optimize_prior!` method optimizes the prior precision λ through Empirical Bayes [3]. The `plot` method visualizes the predictions of the fitted instance. It is provided through the TaijaPlotting meta package.

Code 3: Prior optimization and visualization of the predictive distribution.

```
optimize_prior!(la)            # optimize λ
using TaijaPlotting
plot(la, X, y)                 # plot predictions
```

Figure 1 shows an example involving a synthetic data set consisting of two classes. Contours indicate the predicted probabilities using the plugin estimator (left), untuned Laplace Approximation (center) and finally optimized LA (right). For the latter two, the respective choices for the prior precision parameter λ are indicated in the title. Relying solely on the MAP estimate, the plugin estimator produces overly confident predictions. Conversely, the GLM predictions account for predictive uncertainty as captured by the Laplace posterior.

Figure 2 presents a regression example with optimized LA. Wide regions of the confidence interval (shaded area) indicate high predictive uncertainty. Intuitively, the estimated predictive uncertainty increases significantly in regions characterized by high epistemic uncertainty: epistemic uncertainty arises in regions of the domain that have not been observed by the classifier, so regions that are free of training samples.

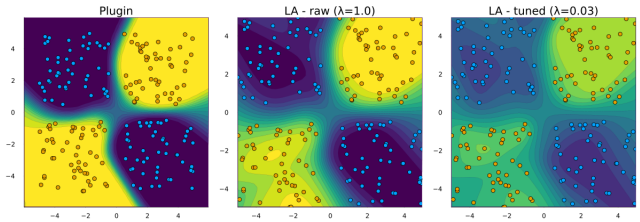

Fig. 1: Posterior predictive distribution for binary classifier: plugin estimate (left), untuned LA (center) and optimized LA (right). The colour of the contour indicates the predicted class probabilities: the more yellow a region, the more confident the classifier that samples belong to the orange class.

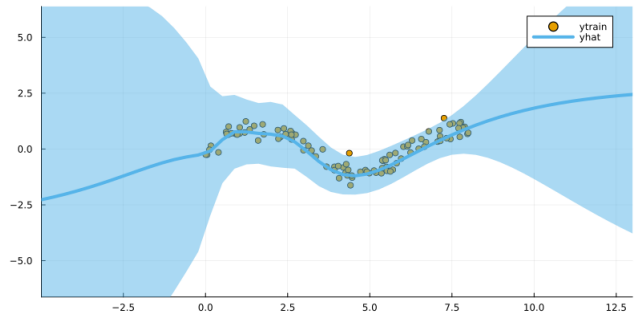

Fig. 2: Posterior predictive distribution for regressor: wide regions of the confidence interval (shaded area) indicate high predictive uncertainty.

4. Scaling Up

As mentioned in Section 2, Laplace Redux hinges on linear approximations of the Hessian, which scale quadratically in the number of network parameters [3]. Our package currently supports two broad approaches to address this issue: the first approach is to compute LA over a subnetwork with explicit control over the number of parameters; the second approach is to use more scalable approximations of the Hessians. For the second approach, the package currently offers support for Kronecker-factored approximate curvature (KFAC) estimation [9]. A third approach is to use sample-based linearised Laplace [1], which is not yet supported.

5. Discussion and Outlook

Laplace Redux is an exciting and promising recent development in Bayesian deep learning. The package `LaplaceRedux.jl` brings this framework to the Julia ecosystem. Future developments are planned and contributions are very much welcome. At the time of writing, we are particularly interested in streamlining the package's interface to the larger Taija ecosystem and improving our support for scalable LA.

6. Acknowledgements

I am grateful to my PhD supervisors Cynthia C. S. Liem and Arie van Deursen for being so supportive of my work on open-source developments. Furthermore, I would like to thank the group of students who contributed to this package through a course project: Mark Ardman, Severin Bratus, Adelina Cazacu, Andrei Ionescu and Ivan Makarov.

References

[1] Javier Antorán et al. *Sampling-based inference for large linear models, with application to linearised Laplace*. 2023. arXiv: 2210.04994 [stat.ML].

[2] Charles Blundell et al. "Weight Uncertainty in Neural Network". In: *International Conference on Machine Learning*. PMLR, 2015, pp. 1613–1622.

[3] Erik Daxberger et al. "Laplace Redux-Effortless Bayesian Deep Learning". In: *Advances in Neural Information Processing Systems* 34 (2021).

[4] Yarin Gal and Zoubin Ghahramani. "Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning". In: *International Conference on Machine Learning*. PMLR, 2016, pp. 1050–1059.

[5] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. "Explaining and Harnessing Adversarial Examples". 2014. arXiv: 1412.6572.

[6] Alexander Immer, Maciej Korzepa, and Matthias Bauer. "Improving Predictions of Bayesian Neural Networks via Local Linearization". 2020. arXiv: 2008.08400.

[7] Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. "Simple and Scalable Predictive Uncertainty Estimation Using Deep Ensembles". 2016. arXiv: 1612.01474.

[8] Neil David Lawrence. "Variational Inference in Probabilistic Models". PhD thesis. University of Cambridge, 2001.

[9] James Martens and Roger Grosse. *Optimizing Neural Networks with Kronecker-factored Approximate Curvature*. 2020. arXiv: 1503.05671 [cs.LG].

[10] James Martens and Roger Grosse. "Optimizing neural networks with kronecker-factored approximate curvature". In: *International conference on machine learning*. PMLR. 2015, pp. 2408–2417.

[11] Andrew Gordon Wilson. "The Case for Bayesian Deep Learning". 2020. arXiv: 2001.10995.