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 [**?**] 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:

> [...] 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 [**?**]. 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 [**?**], Monte Carlo Dropout [**?**] and Deep Ensembles [**?**]. Laplace Redux ([**?**],[**?**]) is one of the most recent and promising approaches to Bayesian neural networks (BNN).

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 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 [**?**].

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 **??** does not work when using approximations for the Hessian [**?**]. Instead, we can use a linear expansion of the predictive around the mode as demonstrated by Immer et al. (2020) [**?**]. 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 [**?**].

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

`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 listing **??**: the `Laplace` function simply wraps the Flux neural network `nn`. Here we have also provided two of the optional key arguments that determine the prior precision λ and the subset of network layers to be used. The returned instance can then be trained on data using the generic `fit!` method. Calling the generic `predict` method on the fitted instance will generate GLM predictions according to Equation **??**.

[language=Julia, escapechar=@, numbers=left, label=lst:laplace, caption=] la = Laplace(nn; @$\lambda = \lambda$@, subset$_of f_weights =: last_layer)fit!(la, data)$

?@fig-pred-mlp shows an example involving a synthetic data set consisting of two classes. Contours indicate the predicted probabilities using the plugin estimator (left) and Laplace approximation (right). 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.

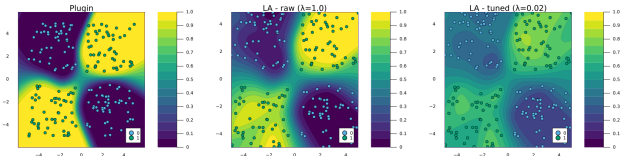

Figure 1: Binary classification: Plugin estimate (left), untuned LA (center) and optimized LA (right).

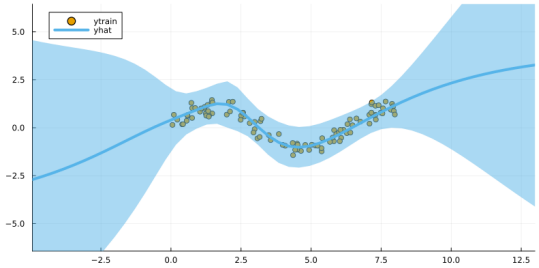

Figure 2: Regression: wide regions of the confidence interval (shaded area) indicate high predictive uncertainty.

The package is still in its infancy and its functionality is limited at the time of writing. For example, it currently still works with full Hessian approximations, as opposed to the less expensive (block-) diagonal variants. That being said, choices regarding the package architecture were made with these future development opportunities in mind. This should hopefully make the package attractive to other Julia developers interested in the topic.

Conclusions

Laplace Redux is arguably one of the most exciting and promising recent developments in Bayesian deep learning. The goal of this project is to bring this framework to the attention of the Julia machine-learning community. The package `LaplaceRedux.jl` offers a useful starting ground for a full-fledged implementation in pure Julia. Future developments are planned and contributions are very much welcome.

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. I am also grateful to the Julia community for being so kind, welcoming and helpful.