Bayesian model ensembling using meta-trained recurrent neural networks

Luca Ambrogioni
Radboud University
l.ambrogioni@donders.ru.nl

Julia Berezutskaya
University of Utrecht
y.berezutskaya@umcutrecht.nl

Umut Güçlü
Radboud University
u.guclu@donders.ru.nl

Eva W. P. van den Borne
Radboud University
e.vandenborne@student.ru.nl

Yağmur Güçlütürk
Radboud University
y.gucluturk@donders.ru.nl

Marcel A. J. van Gerven
Radboud University
m.vangerven@donders.ru.nl

Eric Maris
Radboud University
e.maris@donders.ru.nl

Abstract

In this paper we demonstrate that a recurrent neural network meta-trained on an ensemble of arbitrary classification tasks can be used as an approximation of the Bayes optimal classifier. This result is obtained by relying on the framework of ϵ-free approximate Bayesian inference, where the Bayesian posterior is approximated by training a neural network using synthetic samples. We denote the resulting model as neural ensembler. We show that a single neural ensembler trained on a large set of synthetic data achieves competitive classification performance on multiple real-world classification problems without additional training.

1 Introduction

Deep learning techniques have state-of-the art performance in most classification tasks involving big training sets. However, deep learning methods often overfit on smaller datasets, leading to low generalization performance. In these situations, ensemble and Bayesian methods have a better performance as the variance of the prediction is reduced by averaging. Ensemble methods, such as the random forest method, perform very well on small and medium datasets. The optimal way of ensembling a family of models is given by the Bayes optimal classifier [1]. Unfortunately, this optimal ensembler is intractable for most non-trivial situations. Recent works show that complex Bayesian posterior distributions can be approximated using deep networks trained on synthetic data [2]. In this paper, we construct a recurrent neural network (RNN) that can solve arbitrary classification tasks by approximating the Bayes optimal classifier. This is a form of meta-learning since the RNN is trained on the task of learning the specific classification task, as specified by a model sampled from the ensemble, from a series of predictors/label pairs. In the rest of the paper, we outline the theoretical foundations of our method and we validate its performance on three real-world datasets.

31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA.

2 Related Work

Our present work is similar to the meta-learning methods introduced in [3] and recently expanded in [4, 5]. In these works, a RNN receives as input the predictors of the current sample together with the label of the previous sample and is trained to adapt to a family of classification tasks. Our paper differs from this previous work in two ways. First, we introduce a theoretical connection between the meta-learning problem and Bayes optimal classifiers using ϵ-free approximate Bayesian inference. In particular, we show that the generator of synthetic training sets implicitly defines a prior distribution over an ensemble of possible classification problems and that the optimal posterior can be approximated by a meta-trained RNN. Second, previous works focused on training on either homogeneous sets of synthetic problems (e.g., linear, quadratic) [3] or on meta-datasets with homogeneous features such as Omniglot [4, 5]. In contrast, we meta-train our network on a heterogeneous ensemble of weakly structured synthetic classification tasks. In doing so, we show that it is possible to train complex neural architectures on an unbounded set of classification problems in a way that generalizes well to real-world problems. Thus, our main practical goal is to define a model which, once meta-trained, can be directly used in arbitrary real-world classification problems that do not share any similarities in their features.

3 Ensemble methods and Bayes optimal classifier

In a classification task, the aim is to estimate the probability of the target class assignments y given a set of predictors $\boldsymbol{x}$. In an ensemble learning setting, we assume that the classification task is sampled from a predefined family of classification models $M_1, M_2, ..., M_K$. In our notation, we consider two models with the same parametric form but different parameter values as different models. Therefore, usual training corresponds to finding a model within a large parametric family of models. An ensemble classifier has the following form:

$$p(y^*|\boldsymbol{x}^*, D) = \sum_{k=1}^{K} w_k(D) p(y^*|\boldsymbol{x}^*, M_k) \tag{1}$$

where x^* denotes a new vector of predictors, y^* denotes the corresponding label and D denotes the training data. Different ensemble models use different techniques for setting the weights $w_k(D)$. Most methods, such as bagging and random forests, start from a complex family of models, usually decision trees, and use some form of randomized training algorithm in order to obtain a set of K trained models that fit the data. The final distribution is then obtained by averaging the predictions of these models. In our previous expression, this corresponds to setting all the weights corresponding to the trained models to $1/K$ and all the other (infinitely many) weights to zero. The optimal way of setting the weights $w_k(D)$ can be formally obtained using Bayes' rule. The posterior probability of each model M_k given the training data is given by:

$$p(M_k|D) = \frac{p(D|M_k)p(M_k)}{p(D)}, \tag{2}$$

where D is a training set of predictors $\boldsymbol{x}$ and target class assignments y. Assuming that we know the prior over the family of classification models, the optimal solution to the classification problem is given by marginalizing the posterior distribution $p(y|\boldsymbol{x})$ over all models $M_1, M_2, ..., M_K$ [6]. This is known as the Bayes optimal classifier:

$$p(y^*|\boldsymbol{x}^*, D) = \sum_{k=1}^{K} p(y^*|\boldsymbol{x}^*, M_k) p(M_k|D) \tag{3}$$

In practice, computing the Bayes optimal classifier is intractable as it involves a sum (or an integral) over the whole (usually infinite) ensemble of models.

4 Meta-training on classification ensembles

In this section, we show that the Bayes optimal classifier can be approximated by meta-training on classification tasks as sampled from the prior. The basic idea comes from the recently introduced

framework of ϵ-free approximate Bayesian inference [2]. For simplicity, we assume a Bayesian problem with a (categorical) latent variable of interest a, an arbitrary nuisance latent variable b and a likelihood function $p(x|a,b)$. In the ϵ-free framework, the posterior distribution (marginalized over b) is approximated by the forward pass of a neural network $\boldsymbol{f}(x;\boldsymbol{w})$:

$$p(a|x) \approx f(x;\boldsymbol{w}^*)_a \ , \tag{4}$$

where $f(x;\boldsymbol{w}^*)_a$ is the a-th entry the probability vector $\boldsymbol{f}(x;\boldsymbol{w})$, assumed to be the output of a softmax layer. The optimal weights $\boldsymbol{w}^*$ of this network are determined by the following optimization problem:

$$\boldsymbol{w}^* = \arg\min_{\boldsymbol{w}} E_{p(a,b)} E_{p(x|a,b)} \big[-f(x;\boldsymbol{w})_a \big] \tag{5}$$

Importantly, as far as we can draw samples from $p(a,b)$ and $p(x|a,b)$, we can easily compute unbiased estimates of the loss function. This differs from most situations in deep learning, where the gradient of the validation loss is biased since the training batches are drawn from a finite training set. This implies that, in the context of ϵ-free approximate Bayesian inference, overfitting is not an issue and, consequently, the network can have an arbitrarily complex architecture and be trained for an arbitrarily long number of iterations.

We can use this approach to approximate the predictive distribution of the optimal Bayes classifier of an ensemble. The idea is to sample both the model M and a series of predictors from our prior and to generate the corresponding series of labels by sampling from $p(y|\boldsymbol{x},M)$. Using these samples, we can train a RNN using a cross-entropy loss in order to approximate the predictive distribution $p(x|y)$. That is:

$$p(y^*|\boldsymbol{x}^*) \approx \text{RNN}(\boldsymbol{x}^*;\boldsymbol{w}^*,D)_{y^*} \ , \tag{6}$$

where $\text{RNN}(\boldsymbol{x}^*;\boldsymbol{w}_{opt},D)$ is a recurrent architecture that has received as input a training set D of training pairs $(\boldsymbol{x},y)$. The optimal weights are the solution of the following optimization problem:

$$\boldsymbol{w}^* = \arg\min_{\boldsymbol{w}} E_{p(\boldsymbol{x}^*,M)} E_{p(D|M)} E_{p(y^*|M,\boldsymbol{x}^*)} \big[-\text{RNN}(x^*;\boldsymbol{w},D)_{y^*} \big] \ , \tag{7}$$

which can be solved using stochastic gradient descent. Note that, since at each iteration we are sampling the model M from the prior, we are implicitly marginalizing over the whole ensemble.

5 Recurrent architecture

The architecture of our RNN is shown in Fig. 1. In order to be able to adapt to different classification problems, as specified by the different models in the ensemble, the network needs to receive feedback concerning the labels of the previous data points. This feedback is encoded as a vector of length $P \times C$, where P is the number of predictors and C is the numbers of classification classes. The feedback vector of the n-th sample takes the following form: $\boldsymbol{f^{n+1}} = (y_1^n \boldsymbol{x}^n \ , \ \ldots \ , \ y_C^n \boldsymbol{x}^n)$, where y_c^n is the c-th component of the one-hot encoded label vector of the n-th sample. This feedback vector is fed to a layer of LSTM units through a dense linear map. The output of this first LSTM layer is fed via a dense linear map to another smaller intermediate LSTM layer. The outputs of these two LSTM layers are concatenated together with the current vector of predictors $\boldsymbol{x}^{n+1}$ and fed into a non-recurrent hidden layer via a dense linear map followed by an entry-wise Swish activation function [7]. Finally, the probability vector of the current label is obtained with a softmax output layer.

6 A flexible ensemble of classification models

The performance of our approximate Bayes classifier on real-world data relies on the choice of the ensemble of models. Our aim is to define an ensemble prior that is appropriate for weakly structured classification tasks, where we do not have any prior information about the structure of the predictors and of the statistical relationship between predictors and class labels. We used an ensemble of probabilistic decision trees. In these models, each split is determined by the inner product of the predictor with a vector of weights: $\boldsymbol{w}^\top \boldsymbol{x}$. This allows for diagonal decision boundaries at each split. Instead of using a deterministic decision rule, we send the data to the left branch with probability $\sigma(\boldsymbol{w}^\top \boldsymbol{x} + b)$ and to the right branch with probability $1 - \sigma(\boldsymbol{w}^\top \boldsymbol{x} + b)$. The depth of the tree was randomly sampled from 1 to 15. The weights and the biases at each node were sampled from

univariate normal distributions with mean 0 and variance 1. In order to further increase the flexibility of our ensemble, we used a second family of classification models alongside the differentiable trees. In this second family, the classification dataset were generated as a set of samples normally distributed along the vertices of a 10-dimensional hypercube. Each class then comprises the samples associated with half of the vertices of the hypercube. These datasets were created using Scikit-learn toolbox (v0.18).

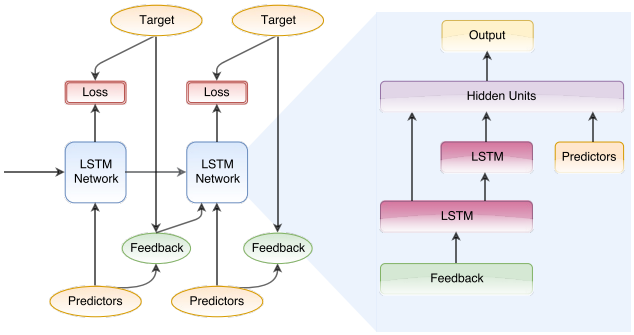

Figure 1: Meta-learning setting and the architecture of the recurrent neural network.

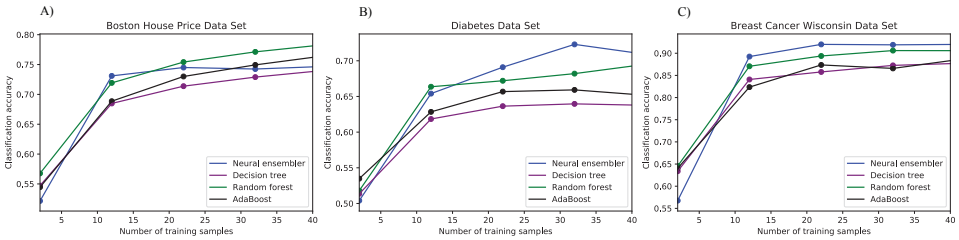

Figure 2: Classification performance on three real-world datasets: Boston house-prices dataset (A), diabetes dataset (B) and breast cancer Wisconsin dataset (C). The performance of the neural ensember is compared to the performance of the random forest, Adaboost and decision tree classifiers.

7 Experiments

We trained a single neural ensembler model on the ensemble described in the previous section. The network was trained on binary classification with 10 predictors. The Chainer deep learning toolbox [8] was used for model training. After training, the model was tested separately on three public datasets, available through Scikit-learn (v0.18): Boston house-prices dataset, the diabetes dataset and breast cancer Wisconsin dataset. In all datasets only the first 10 predictors were used. The Boston and diabetes datasets were originally a regression problem, but were treated as a classification task by replacing the value of the output variable with label 0 if it was less than the total median or label 1 otherwise. Thus, each dataset was a binary classification task. The datasets contained 506, 442 and 569 data points, respectively. However, in order to reliably evaluate the model performance on small data, in each dataset, we sampled data subsets of length N (from $N = 2$ to $N = 42$) at random. The model was tested by making a prediction for the $(N + 1)$-th sample. The sampling and testing was repeated 500 times for different re-samplings (without replacement) of the full dataset and the model performance scores were averaged. This procedure was repeated for multiple subset lengths $N = \{2, 12, 22, 32, 42\}$. Per dataset, the model performance was compared to three other models: random forest, AdaBoost and a simple decision tree. All three models were trained on the sampled subsets of data and tested on the $(N + 1)$-th sample. The training and testing was repeated for M samplings and N subset lengths. The performance was averaged over M samplings as in case of the neural ensembler. For all datasets we observed that the neural ensembler model achieved competitive performance when compared with other ensembling approaches (Fig. 2). For diabetes and Wisconsin datasets the neural ensembler model outperformed other classifiers when using subsets of 22-42 samples.

References

[1] T. G. Dietterich. *Ensemble Methods in Machine Learning*. Springer, 2000.

[2] G. Papamakarios and I. Murray. Fast ε-free inference of simulation models with Bayesian conditional density estimation. *Advances in Neural Information Processing Systems*, 2016.

[3] D. V. Prokhorov, L. A. Feldkarnp, and I. Y. Tyukin. Adaptive behavior with fixed weights in rnn: an overview. *International Joint Conference on Neural Networks*, 3, 2002.

[4] A. Santoro, S. Bartunov, M. Botvinick, D. Wierstra, and T. Lillicrap. Meta–learning with memory–augmented neural networks. *International Conference on Machine Learning*, 2016.

[5] O. Vinyals, C. Blundell, T. Lillicrap, and D. Wierstra. Matching networks for one shot learning. *Advances in Neural Information Processing Systems*, 2016.

[6] T. M. Mitchell. *Machine Learning*. McGraw Hill, 1997.

[7] P. Ramachandran, B. Zoph, and Q. V. Le. Swish: a self-gated activation function. *arXiv:1710.05941*, 2017.

[8] S. Tokui, K. Oono, S. Hido, and J. Clayton. Chainer: a next-generation open source framework for deep learning. *Workshop on Machine Learning Systems (NIPS)*, 2015.