Dichotomize and Generalize: PAC-Bayesian Binary Activated Deep Neural Networks

Gaël Letarte¹, Pascal Germain², Benjamin Gueldj³, François Laviolette¹

1 Département d'informatique et de génie logiciel, Université Laval, Québec, Canada
2 Equipe-projet Modal, Inria Lille - Nord Europe, Villeneuve d'Ascq, France
3 UCL Centre for Artificial Intelligence, University College London, London, England

Introduction

We present a comprehensive study of multilayer neural networks with binary activation, relying on the PAC-Bayesian theory.

Contributions

- An end-to-end framework to train a binary activated deep neural network (DNN).
- Nonvacuous PAC-Bayesian generalization bounds for binary activated DNNs.

Binary Activated Neural Networks

- L fully connected layers
- d_l denotes the number of neurons of the lth layer
- \text{sgn}(a) = 1 if a > 0 and \text{sgn}(a) = -1 otherwise
- Weights matrices : \( W_l \in \mathbb{R}^{d_{l-1} \times d_l}, \theta = \text{vec}((W_{1:n})_{n \in \mathbb{N}}) \in \mathbb{R}^{d_1 + \ldots + d_{n-1}} \)

Prediction

\[ f_{\theta}(x) = \text{sgn}(W_{1:n, \text{sgn}}(\ldots \text{sgn}(W_1 x))) \]

PAC-Bayesian Theory

Given a data distribution \( D \), a training set \( S = \{(x_1, y_1), \ldots, (x_n, y_n)\} \sim \mathcal{D}^n \), with \( x_i \in \mathbb{R}^d \) and \( y_i \in \{-1, 1\} \), a loss \( \ell : \{-1, 1\}^2 \to [0, 1] \), a predictor \( f \in \mathcal{F} \):

\[ \mathcal{L}(f) = \frac{1}{n} \sum_{(x,y) \in D} \ell(f(x), y) \leftarrow \text{Minimization} \]

\[ \mathcal{E}(f) = \frac{1}{n} \sum_{(x,y) \in D} \ell(f(x), y) \leftarrow \text{Empirical Loss} \]

PAC-Bayesian Theorem

For any prior \( P_0 \) on \( \mathcal{F} \), with probability \( 1-\delta \) on the choice of \( S \sim \mathcal{D}^n \), we have for all \( \beta > 0 \), and all posterior distribution \( P \) on \( \mathcal{F} : \)

\[ \mathcal{L}_{\beta}(f) \leq \frac{1}{2} \frac{1}{n} \log \left( 1 + \exp \left( -n \mathcal{D}(f) + \frac{1}{2} \mathcal{KL}(Q_{\beta} \mid \mid P) + \log \frac{2}{\delta} \right) \right) \]

Linear Classifier

PAC-Bayesian Learning of Linear Classifiers (Germain et al., 2009)

\[ f_{\beta}(x) = \text{sgn}(w \cdot x), \text{ with } w \in \mathbb{R}^d \]

PAC-Bayes analysis

- Space of all linear classifiers \( \mathcal{F} = \{ f_{\beta} \mid \beta \in \mathbb{R} \} \)
- Gaussian posterior \( Q_{\beta} = N(w, I_d) \) over \( f_{\beta} \)
- Gaussian prior \( P_0 = N(w_0, I_d) \) over \( f_{\beta} \)
- Predictor \( f_{\beta}(x) = E_{X \sim Q_{\beta}} f_{\beta}(x) = \text{sgn}(w \cdot x) \)
- Linear loss \( \ell(f_{\beta}(x), y) = \frac{1}{2} - y f_{\beta}(x) \)

Bound minimization

\[ C n \mathcal{L}(f_{\beta}) + KL(Q_{\beta} \mid \mid P_{\beta}) = C \frac{1}{2} \sum_{x \in \{-1,1\}} \text{sgn}(w \cdot x) + \frac{1}{2} \|w - u\|^2 \]

Shallow Learning

Posterior \( Q_{\beta} = N(\theta, I_d) \), over the family of all networks \( \mathcal{F}_\beta = \{ f_{\beta} \mid \beta \in \mathbb{R} \} \), where

\[ f_{\beta}(x) = \text{sgn}(w \cdot \text{sgn}(W_1 x)) \]

PAC-Bayesian bound ingredients

- Empirical loss : \( \mathcal{L}(F_{\beta}) = \mathbb{E}_{W \sim P_{\beta}} \mathcal{L}(f_{\beta}) = \frac{1}{2} \sum_{x \in \{-1,1\}} \|w - y\|_2^2 \)
- Complexity term : \( \mathcal{KL}(Q_{\beta} \mid \mid P_{\beta}) = \frac{1}{2} \beta \|\theta\|_2^2 \), with \( \beta \in \mathbb{R} \)
- Generalization bound : \( \frac{1}{2} \left( 1 - \exp \left( -C \mathcal{L}(F_{\beta}) + \frac{1}{2} \mathcal{KL}(Q_{\beta} \mid \mid P_{\beta}) + \log \frac{2}{\delta} \right) \right) \)

Visualisation

The proposed method can be interpreted as a majority vote of hidden layer representations.

Stochastic Approximation

\[ F_{\beta}(x) = \sum_{s \in \{-1,1\}^d} F_{\beta,s}(x) \Pr(s \mid x, W_1) \]

Monte Carlo sampling

We generate \( T \) random binary vectors \( \{s^t\}_T \), according to \( \Pr(s \mid x, W_1) \):

\[ F_{\beta}(x) \approx \frac{1}{T} \sum_{t=1}^{T} F_{\beta,s^t}(x) \]

This turns out to be a variant of REINFORCE algorithm (Williams, 1992).

Deep Learning (PBGNet)

To enable a layer-by-layer computation of the prediction function, we want the neurons of a given layer to be independent of each other. This is achieved with the tree architecture mapping function \( \theta(s) \).

Recurrent def. \( F_{\beta,s}(x) \) denotes the output of the \( s \)-th neuron of the \( k \)-th hidden layer:

\[ F_{\beta,s}(x) = \text{sgn} \left( \sum_{c \in \{-1,1\}} \left( \frac{1}{2} - 2^{-k} \cdot \sum_{t = c}^{2^k} w \cdot x \right) \right) \]

Kullback-Leibler regularization

\[ \mathcal{KL}(Q_{\beta} \mid \mid P_{\beta}) = \frac{1}{2} \left( \|W_{\text{rel}} - u\|_2^2 + \sum_{c = 0}^{d_1} \|W_{\text{int}} - U_{\text{int},c}\|_2^2 \right) \]

Experiment

We perform model selection using a validation set for a MLP with tanh activations, and using the PAC-Bayes bound for our PBGnet and PBGnet-bnd algorithms. PBGnet and PBGnet-bnd are intermediate variants.