Bayesian and Maximum Likelihood DTU Neural Networks

Finn Årup Nielsen
Section for Digital Signal Processing
Department of Mathematical Modelling
Technical University of Denmark
DK-2800 Lyngby, Denmark

Email: fn@imm.dtu.dk
WWW: http://www.imm.dtu.dk/~fn
OVERVIEW

- Artificial neural networks
- Maximum likelihood, MAP, MPL neural networks
- Bayesian neural networks
  - MCMC Bayesian neural networks
    * Hybrid Monte Carlo
- lyngby matlab toolbox ML/MLP/MAP neural network
- Radford Neal's "flexible Bayesian models" (fbm).
- Comparison between fbm and lyngby
“Forward equation” or “network equation” for two-layer feedforward neural network:

$$Y = [\tanh ([X \ 1] V) \ 1] W$$

1. $X$: Input as a datamatrix
2. $Y$: Output
3. $V$: Input weights (parameters)
4. $W$: Output weights (parameters)
5. $1$: Bias units
6. $\tanh$: The nonlinearity: a sigmoidal function (hyperbolic tangent)
... NEURAL NETWORK

Neural networks uses:

• Regression, no modification of the outputs, linear output, Gaussian distribution $]-\infty; +\infty[$
• Binary (binary classification), tanh on output, binomial distribution. $]-1; +1[$
• Classification, softmax function on outputs [Bridle, 1990], multinomial distribution. $]0; +1[$
• Hybrid: regression and classification.

Types of estimation in neural network:
... NEURAL NETWORK

Modelling the regression output:

$y$ is the output of the neural network — hopefully the "truth". $\epsilon$ is additive noise. $t$ is the target, i.e. the empirical output:

$$t = y + \epsilon$$  \hspace{1cm} (2)

Gaussian additive noise:

$$p(t|x; u) = \frac{1}{\sqrt{(2\pi)^{n_0} |\Sigma_\epsilon|}} \exp \left( -\frac{1}{2} (y - t)^T \Sigma_\epsilon^{-1} (y - t) \right)$$  \hspace{1cm} (3)

$$p(t|x, u) = \sqrt{\frac{\tau}{2\pi}} \exp \left( -\frac{\tau}{2} (y - t)^2 \right)$$  \hspace{1cm} (4)

Typical prior / penalization:

$$p(u) = \frac{1}{\sqrt{(2\pi)^{nu} |\Sigma_u|}} \exp \left( -\frac{1}{2} u^T \Sigma_u^{-1} u \right)$$  \hspace{1cm} (5)
ML OR MPL/MAP NEURAL NETWORK

Maximum likelihood neural network (ML-NN)

- Cost function: $\propto - \log p(t|x; u)$
- Calculate derivatives: First and second order
- Optimize network with: gradient descent (“back-propagation”: First order derivative only depend on values in the proceeding layer), pseudo-Gauss-Newton, Gauss-Newton, Levenberg-Marquardt

Maximum penalized likelihood / Maximum a posteriori neural network

- Cost function: $\propto - \log p(t|x; u) - \log p(u)$
- Calculate derivatives: First and second order
- Optimize network
- Optimize the hyperparameters in $\log p(u)$ By cross-validation or analytic estimates (of the generalization error), e.g., Moody’s GPE [Moody, 1992] (Akaike’s FPE for regularized models)
  - Gaussian prior $\rightarrow$ weight decay
  - Individual priors $\{0, \text{inf}\}$ (“implicit prior” $p(u) \propto \exp(-\log(|u|))$ [Sporring, 1997]) $\rightarrow$ pruning
• Gaussian approximations to posterior
  – Type II maximum likelihood (ML-II) [Berger, 1985].
• Markov chain Monte Carlo
  – Radford Neal [Neal, 1996], Carl Edward Rasmussen [Rasmussen, 1996]
  – Integration over weights and hyperparameters. Gibbs sampling for hyperparameters
• “Sequential Monte Carlo” for on-line learning (?) [de Freitas and Niranjan, 1998].
MCMC IN BAYESIAN NEURAL NETWORKS

Posterior / costfunction

• Mirror modes: $2^{(n_h)}n_h!$
  – Sign switch between hidden and output layer weights
  – Permutation of hidden units
• Complex nonlinear: Hyperbolic tangent

“Optimization” / Integration

• Gibbs sampling not possible for weights because the posterior is complex
• Rejection sampling not good: Weight space is high dimensional
• MCMC possible: However, “stiff” valleys in posterior makes it slow.
• MCMC with momentum ("Hybrid Monte Carlo")
• Simulated annealing (tempering) on MCMC parameters: no speed-up [Neal, 1996, page 65]
• Ordered overrelaxation [Neal, 1995] [Neal, 1998b]
MCMC WITH MOMENTUM

Reviewed in [Neal, 1993, Chapter 5] and [Neal, 1996, section 3.1]

Stochastic dynamics + Metropolis = Hybrid Monte Carlo [Duane et al., 1987]

Stochastic dynamics

• \( E(u) \) = potential energy \( \propto \log(\text{posterior}(u)) \)
• \( K(z) \) = kinetic energy, fictitious momentum associated with \( u \).
• \( H(u, z) = E(u) + K(z) \) = Total energy = Hamiltonian
• \( p(u, z) = \frac{1}{Z_H} \exp(-H) = \text{posterior}(u) p(z) \), “canonical distribution”
• Sampling in a one dimensional path in phase space \( (u, z) \)
... MCMC WITH MOMENTUM

Leapfrog

Stochastic dynamics + discretization = “leapfrog”

1. Half step for kinetic energy
2. Full step for potential energy
3. Half step for kinetic energy

Hybrid Monte Carlo

1. Stochastic transition, sample new momenta
2. Dynamical transition
   (a) Perform $L$ leapfrog steps of size $\epsilon$
   (b) Negate the momenta
       $$(u^*, z^*) = (\hat{u}(\epsilon L), -\hat{z}(\epsilon L))$$       (6)
   (c) “Metropolis reject”. Eliminate the bias from the discretization approximation.
       $$\min[1, \exp(-(H(u^*, z^*) - H(u, z)))]$$       (7)
HYBRID MONTE CARLO VARIANTS

Hybrid Monte Carlo variants reviewed
[Neal, 1993, section 5.2] and
[Neal, 1996, section 3.5]

- Non-leapfrog discretization
  [Creutz and Gocksch, 1989]
- Partial gradients [Neal, 1996, section 3.5.1].
  - Each leapfrog step uses only the gradient computed from a part on the data set.
  - Should be faster to compute. However, lower acceptance rate, thus no advantage.
- Windows [Neal, 1994] [Neal, 1996, section 3.5.2]
- Persistence  [Horowitz, 1991] [Neal, 1996, section 3.5.3].
  - Partial remembering the momenta across Metropolis steps (and using only a single leapfrog step).
  - Used in [Rasmussen, 1996]
• lyngby: Matlab toolbox for mainly functional neuroimaging
• Available at http://hendrix.imm.dtu.dk/software
• Contains maximum likelihood neural network for regression, classification and binary outputs
• Optimization with: gradient descent, pseudo-Gauss-Newton, Levenberg-Marquardt, ...
• Pruning, weight decay (Grid search).
• Simple (single-blocked) cross validation

>> [V, W, Evaluation] = ...
   lyngby_nn_qmain(Xtrain, Ttrain, ...
       'GenOptim', 'Pruning', ...
       'Validation', 'Singleblocked', ...
       'HiddenUnits', 8, ...
       'Reg', 0.001);

>> Ytest = lyngby_nn_qforward(Xtest, V, W);
>> Etest = lyngby_nn_qerror(Ttest, Ytest);
RADFORD NEAL’S
“FLEXIBLE BAYESIAN MODELS”

• fbm: “Flexible Bayesian models” — Unix programs with source [Neal, 1998a].
• Available at http://www.cs.toronto.ca/~radford
• Contains Bayesian neural network, Gaussian process, mixture models
• More general feedforward architecture: Only forward network is required — no derivatives.
• Simple priors: Gaussians and Gamma densities.
• Stochastic integration with: Gibbs sampling (heatbath), hybrid Monte Carlo (hybrid)

> net-spec rlog.net 1 8 1 / - 0.05:0.5 0.05 - x0.05:0.5 - 100
> model-spec rlog.net real 0.05:0.5
> data-spec rlog.net 1 1 / rdata@1:100 . rdata@101:200 .

> net-gen rlog.net fix 0.5
> mc-spec rlog.net repeat 10 sample-noise heatbath hybrid 100:10 0.2
> net-mc rlog.net 1

> mc-spec rlog.net sample-sigmas heatbath hybrid 1000:10 0.4
> net-mc rlog.net 100
Radford Neal’s simple regression example

\[ y = 0.3 + 0.4x + 0.5 \sin(2.7x) + 1.1/(1 + x^2); \] \hspace{1cm} (8)

- From Neal’s software \cite{Neal, 1998a}
- 100 training examples, 100 test examples
- Noise on data: \( \sigma_\epsilon = 0.1 \)
PRIORING EXAMPLES …, (1/2)

Bayesian neural network

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Parameter</th>
<th>Parameterprior</th>
<th>Hyperparameterprior</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input weights</td>
<td>v</td>
<td>v \sim N(0, \tau_v I)</td>
<td>\tau_v \sim G(400, 0.5)</td>
</tr>
<tr>
<td>Output weights</td>
<td>w</td>
<td>w \sim N(0, \tau_w I)</td>
<td>\tau_w \sim G(400, 0.5)</td>
</tr>
<tr>
<td>Input bias</td>
<td>v_0</td>
<td>v_0 \sim N(0, \tau_{v_0} I)</td>
<td>\tau_{v_0} \sim G(400, 0.5)</td>
</tr>
<tr>
<td>Output bias</td>
<td>w_0</td>
<td>w_0 \sim N(0, 100^2)</td>
<td></td>
</tr>
<tr>
<td>Output noise</td>
<td>\epsilon</td>
<td>\epsilon \sim N(0, \tau_\epsilon)</td>
<td>\tau_\epsilon \sim G(400, 0.5)</td>
</tr>
</tbody>
</table>

Table 1: Example on prioring from [Neal, 1998a] in the simple regression example. (See the file doc/Ex-netgp-r.html)

Point estimating (MPL) neural network

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Parameter</th>
<th>Parameterprior</th>
<th>Hyperparameterprior</th>
</tr>
</thead>
<tbody>
<tr>
<td>All weights</td>
<td>u</td>
<td>u \sim N(0, \tau_u I) + N(0, \tau_{u_0} I)</td>
<td>\tau_u \in {\ldots 0.001 0.01 0.1 \ldots}</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>\tau_u \in {0, \infty}^{n_u}</td>
</tr>
<tr>
<td>Input bias</td>
<td>v_0</td>
<td>v_0 \sim \mathcal{F}(\delta)</td>
<td></td>
</tr>
<tr>
<td>Output bias</td>
<td>w_0</td>
<td>w_0 \sim \mathcal{F}(\delta)</td>
<td></td>
</tr>
<tr>
<td>Output noise</td>
<td>\epsilon</td>
<td>\epsilon \sim N(0, 1)</td>
<td></td>
</tr>
</tbody>
</table>

Table 2: Example on prioring in a MPL-NN with the 

lyngby toolbox. - Pruning and common weight decay parameter on input and output weights with grid search on weight decay parameter. - No prior on input and output biases. - No modelling of output noise (assumed Gaussian)
... PRIORING EXAMPLES, (2/2)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Parameter prior</th>
<th>Hyperparameter prior</th>
<th>Hyper^2 parameter prior</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mathbf{v}$</td>
<td>$\mathbf{v} \sim \mathcal{N}(0, \tau_v \mathbf{I})$</td>
<td>$\tau_v \sim \mathcal{G}(\mu_{\tau_v}, 0.5)$</td>
<td>$\mu_{\tau_v} \sim \mathcal{G}(400, 1)$</td>
</tr>
<tr>
<td>$\mathbf{w}$</td>
<td>$\mathbf{w} \sim \mathcal{N}(0, \tau_w \mathbf{I})$</td>
<td>$\tau_w \sim \mathcal{G}(\mu_{\tau_w}, 0.5)$</td>
<td>$\mu_{\tau_w} \sim \mathcal{G}(400, 1)$</td>
</tr>
<tr>
<td>$\mathbf{v}_0$</td>
<td>$\mathbf{v}<em>0 \sim \mathcal{N}(0, \tau</em>{v_0} \mathbf{I})$</td>
<td>$\tau_{v_0} \sim \mathcal{G}(400, 0.5)$</td>
<td></td>
</tr>
<tr>
<td>$w_0$</td>
<td>$w_0 \sim \mathcal{N}(0, 1000^2)$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 3: Example on prioring from [Rasmussen, 1996].

$$\tau \sim \mathcal{G}(\mu, \alpha) \propto \tau^{\alpha/2-1} \exp(-\tau \alpha / 2\mu)$$  \hspace{1cm} (9)
Figure 1: Sampling path of output weights for Neal’s simple regression example for a 1–8–1 MCMC-BNN.

- Weights do not convergence to a single value
- After 300 steps: not all modes explored. E.g. the thick weight has not been negative
WEIGHT AUTOCORRELATION COEFFICIENT

Figure 2: Biased autocorrelation coefficient of one of the weights in Neal’s simple regression example.

- The autocorrelation continues to decrease (as the weights explore new space)
- Convergence might be of minor importance: The neural network already predicts quite well with samples from 30 to 100.
WEIGHTS AND HYPERPARAMETERS

Figure 3: Hyperparameter and size of weights.
LEARNING CURVE

Figure 4: Learning curve.

- (Almost) equal performance on large data sets (consistent with Rasmussen–Hansen density modeling)
- MCMC-BNN better when there is few data sets (consistent with Rasmussen–Hansen density modeling)
- Performance of MPL-NN critically dependent on hyperparameter.
References


