TPL Implementation Notes: Bayesian Neural Networks Based on the "Bayes By BackProp" Algorithm

Zach Gelbaum

February 7, 2019

Contents

1 Overview

1.1 Artificial Neural Networks

A neural net (NN) is described by a directed graph on M vertices (*neurons*). The graph structure is arranged into N *layers* L_k, with an *input layer* L_0, middle or *hidden layers* $L_1, L_2, \ldots, L_{N-1}$, and an *output layer* L_N. The network is typically depicted flowing from left to right, so that the input layer is leftmost, then the middle layers, and finally the output layer. The topology of the graph is defined by the specification of the set of edges, which may connect every neuron in layer L_{k-1} to every neuron in L_k, or possibly only a subset. There is then a collection of weights on the edges of the graph $w_{i,j} \in \mathbb{R}$, a collection

of shifts or *bias* terms b_j $j = 1, \ldots, N$, and a collection of *activation functions* $f_i : \mathbb{R} \to \mathbb{R}$, $i = 1, \ldots, N$, one for each layer except the input layer. The NN functions by iterativly applying affine transformations and activation functions to the data at each layer.

Data is (in our case) given by a collection of input variables $x \in \mathbb{R}^m$ and response variables $y \in \mathbb{R}^r$. The number of neurons in the input layer L_0 is equal to the number of input variables, and the number of layers in the output Layer L_N is equal to the number of response variables. The number of neurons in the middle layers may vary and this choice greatly affects the behavior of the net.

Typically, the activation functions f_i are the same for all neurons in each layer except for possibly the output layer L_N. A common choice for regression problems is all neurons in the middle layers L_1 through L_{N-1} use the sigmoid function

$$f_i = \frac{1}{1 + e^{-x}},$$

and for neurons in the output layer L_N is a linear function

$$f_i = x.$$

The affine transformations are given by a scaling factor and a shift, $(w_{i,j}, b_j)$, where there is a weight assigned to each edge and a scale factor is assigned to each neuron in layers L_k, $k > 0$.

Data propagates through the network as follows: Label the neurons in the input later $n_{i_1}, \ldots, n_{i_m}$. The input variables x are fed into the input layer and for each edge connecting each input neuron to a neuron in the first middle layer L_1, the affine transformation defined by the edge weights and bias terms is applied, giving a new vector of transformed data,

$$x_{i_h} \mapsto w^1_{i_h,j}(x_{i_h}) + b^1_j = W^1 x + \mathbf{b}^1,$$

$h = 1, \ldots, m$ and for j ranging over all neurons in L_1. This new vector, call it $\bar{x}^0$, is then fed into the activation functions of the first middle layer as above,

$$\bar{x}^0_{i_h} \mapsto f_j(\bar{x}^0_{i_h}) = f_j(W^1 x + \mathbf{b}^1).$$

This vector of transformed data is then fed to the next layer L_2 in the same way, applying the next set of affine transformations and activation functions, so that, denoting the vector output of L_k by $\bar{x}^k$, we have $\bar{x}^k_i = f_i(W^k \bar{x}^{k-1} + \mathbf{b}^k)$. After the data is output from L_{N-1} the last set of affine transformations is applied and the output layers simply give this result: $\bar{x}^{N-1} \mapsto w^N_{i,j}\bar{x}^{N-1}_{i,j} + b^N_j = W^N \bar{x}^{N-1} + \mathbf{b}^N \equiv \bar{x}^N$ where i ranges over all neurons in L_{K-1} and j ranges over all neurons in the output layer $j = 1, \ldots, r$ (we have reused notation for indices). The final output of the net $\bar{x}^N$ is then compared to the observed values of the response variables corresponding to the input data x.

Typically, NN's are fit by defining a cost function $F(\bar{x}^K, y)$ and performing an optimization to find the choices of $(w_{i,j}, b_i)$ that minimize F over all data samples in some training set $(x_k, y_k) \in D$. Common choices are a quadratic cost function

$$\sum_{(x_k,y_k)\in D} \|\bar{x}_k^K - y_k\|^2.$$

Since the parameter space can be quite large in practice, gradient descent methods that do not require evaluation over the whole parameter space are often used, such as stochastic gradient descent. The output of such optimization is a set of weights and scaling factors $(w_{i,j}, b_j)$.

The overall fitting procedure is thus as follows: start by defining a topology on the directed graph by choosing the number of neurons, the number of middle layers, and a set of edges connecting each layer. Then choose the activation functions for each layer. Then given a training set of data D, choose a cost function and perform the above optimization to output the weights, thus yielding a complete specification of the NN that best fits the data in D under the cost function chosen.

Note that in addition to the above parameters chosen in defining the net, there are also choices of parameters in the optimization, e.g., the step size in the stochastic gradient descent, that can affect the overall fitting of the network to data.

1.2 Bayesian Neural Nets (BNN's)

The above procedure yields a single network specification, without any measure of uncertainty, and in practice such a process can lead to overfitting. In order to avoid overfitting and obtain a quantitative measure of confidence in the fitted network for use in predictions, a Bayesian framework is advocated. The output of a Bayesian algorithm is a collection of NN's and a density on this collection that one can sample from in order to obtain probabilistic estimates of the response variables.

Given a set of data/observations, BNN approaches start with a *prior* density and a *posterior* density on the parameters of the network, e.g., the weights. Given the data, an optimization is performed and at each step the posterior density is updated to reflect the information contained in the data. The hope is that the result is a posterior density on the parameters that optimally reflects the information contained in the data. These densities can be sampled from, and for each sample the network determined by the sample can be applied to input data. The resulting ensemble of outputs can then be viewed as a density and probabilistic statements made. The statistics of the final posterior density reflect the certainty of the network, e.g., in the variance of the parameter, and statistics of the observed density in the ensemble of outputs give confidence estimates, thus providing a quantification of the uncertainty in the output of the network and CYA capabilities when used in forecasting applications.

2 Bayes By Backprop Algorithm

2.0.1 Overview

In [1] the authors introduce an algorithm for BNN's based on a free energy minimization principle. The output of their algorithm is as above a collection of probability densities, one for each parameter $w_{i,j}^k$, $\mathbf{b}_j^k$ and thus yields a Bayesian density on a collection of networks given a data set.

Remark 2.1. In the paper, the authors use the convention of adding a "bias neuron" to each layer that always outputs 1, and assigning a weight to each edge emanating from it. This is equivalent to including shifts as above, but to match notation with that paper we will just denote all parameters by $\mathbf{w} \in \mathbb{R}^{np}$, where $np = \sum_{k=1}^{N}(|L_{k-1}| + 1)|L_k|$ with $|L_k|$ being the number of neurons in layer L_k.

The idea behind the algorithm is to choose a posterior density depending on parameters and optimize using a well chosen cost function to obtain a final posterior on the network parameters. Let the weights be given by $\mathbf{w}$, the candidate parametric posterior by $q(\mathbf{w}|\theta)$, and the prior distribution by $P(\mathbf{w})$. Assuming there is a "true" or optimal posterior on the weights given the data, $P(\mathbf{w}|D)$, the cost function the authors use is defined as the Kullback-Leibler divergence between $q(\mathbf{w}|\theta)$ and $P(\mathbf{w}|D)$:

$$\mathbf{KL}[q(\mathbf{w}|\theta)||P(\mathbf{w}|D)] = \int q(\mathbf{w}|\theta) \log \frac{q(\mathbf{w}|\theta)}{P(\mathbf{w})P(D|\mathbf{w})}\, d\mathbf{w} \tag{1}$$

$$= \mathbf{KL}[q(\mathbf{w}|\theta)||P(\mathbf{w})] - \mathbb{E}_{q(\mathbf{w}|\theta)}[\log P(D|\mathbf{w})], \tag{2}$$

where $P(D|\mathbf{w})$ is the density giving the conditional distribution of the data given the $\mathbf{w}$, and in this case is chosen to be Gaussian yielding the quadratic cost above,

$$\log(P(D|\mathbf{w})) = \sum_k \|\bar{x}_k^K - y_k\|^2, \tag{3}$$

with the sum over all samples in D. The quantit y (1) is called the variational free energy.

Another key component to the approach is a proposition (3.1 in the paper,) showing that under certain conditions, satisfied by design in the algo, taking expectations and derivatives commutes. This is important in enabling the optimization to be performed by gradient descent.

2.0.2 Description of Algorithm

The free parameters to choose are as a number of nodes, a number of layers, and the activation functions. Because the output will be a set of densities on the weights, all possible edges between layers are included, as those with low value will presumably be assigned a density concentrated near zero (though this statement could bear some investigation.).

Once the basic structure of the net is so specified and a training data set is given, the algorithm proceeds as follows:

First a choice of prior and posterior densities is made. Denote the prior density on the weights $\mathbf{w}$ by $P(\mathbf{w})$ and the parametric posterior density by $q(\mathbf{w}|\theta)$. In the paper the authors use for $P(\mathbf{w})$ a mixture of centered Gaussians:

$$P(\mathbf{w}) = \prod_j s\mathcal{N}(\mathbf{w}_j|0,\sigma_1^2) + (1-s)\mathcal{N}(\mathbf{w}_j|0,\sigma_2^2) \tag{4}$$

for $s \in (0,1)$, $\mathbf{w}_j$ ranges over all weights in the network, $\mathcal{N}(x|0,\sigma^2)$ denotes a Gaussian with zero mean and variance σ^2 evaluated at x, $\sigma_2 < \sigma_1$, and $\sigma_2 \ll 1$. This specific choice is not essential to the method and some ambiguity remains as to how the variances of the component Gaussians are to be chosen (see §3.3 and §5.1 in the paper). The authors do give the results of comparison with using Gaussian priors, and the mixture outperforms, but details of this comparison are not given.

For the posterior, the authors choose independent Gaussians on each weight $\mathbf{w_i}$, each with a mean μ_i and σ_i^2. They let $\sigma_i = \log(1+\exp(\rho))$ in order to ensure positivity is maintained during optimization, so that $\theta = (\mu_i, \rho_i)$. Thus in order to sample from $q(\mathbf{w}|\theta)$ one only needs an iid sample of standard normals.

Given an ensemble of samples of the weights $\{\mathbf{w}^i\}$, the authors approximate the cost function (2) by

$$\mathcal{F}(\mathbf{w}, D) = \sum_i \log(q(\mathbf{w}^i|\theta)) - \log(P(\mathbf{w}^i)) - \log(P(D|\mathbf{w^i}))$$

where $\mathbf{w}^i$ is the ith monte carlo sample of $\mathbf{w}$ from q. We write

$$\mathcal{F}(\mathbf{w}, D) = \sum_i f(\mathbf{w}^i, \theta),$$

$$f(\mathbf{w}, \theta) = \log(q(\mathbf{w}|\theta)) - \log(P(\mathbf{w})) - \log(P(D|\mathbf{w})). \tag{5}$$

We will use $f(\mathbf{w}, \theta)$ in the gradient descent below.

To fit the posterior to the data, a gradient descent optimization is run as follows: initialize starting values of θ: μ_i^0, ρ_i^0. Then each step $t = 0, 1, \ldots$ of the optimization proceeds as follows:

1. Generate n standard normals ϵ_i
2. Let $\mathbf{w}_i^t = \mu_i^t + \log(1+e^{\rho_i^t})\epsilon_i$
3. Let $\theta^t = (\mu^t, \rho^t)$
4. Let $f = f(\mathbf{w}^t|\theta^t)$

5. Calculate the gradient of f with respect to μ:

$$\frac{\partial f}{\partial \mu_i} = \sum_{j=1}^{n} \frac{\partial f}{\partial \mathbf{w}_j} \frac{\partial \mathbf{w}_j}{\partial \mu_i} + \sum_{k=1}^{2n} \frac{\partial f}{\partial \theta_k} \frac{\partial \theta_k}{\partial \mu_i} = \frac{\partial f}{\partial \mathbf{w}_i} + \frac{\partial f}{\partial \theta_v},$$

where $\theta_v = \mu_i$,

$$\Delta_\mu = \left\langle \frac{\partial f}{\partial \mu_i} \right\rangle \Big|_{(\mathbf{w}^t, \theta^t)}$$

6. Calculate the gradient of f with respect to ρ:

$$\frac{\partial f}{\partial \rho_i} = \sum_{j=1}^{n} \frac{\partial f}{\partial \mathbf{w}_j} \frac{\partial \mathbf{w}_j}{\partial \rho_i} + \sum_{k=1}^{2n} \frac{\partial f}{\partial \theta_k} \frac{\partial \theta_k}{\partial \rho_i} = \frac{\partial f}{\partial \mathbf{w}_i} \frac{\epsilon_i^t}{1 + e^{\rho_i^t}} + \frac{\partial f}{\partial \theta_v},$$

where $\theta_v = \rho_i$,

$$\Delta_\rho = \left\langle \frac{\partial f}{\partial \rho_i} \right\rangle \Big|_{(\mathbf{w}^t, \theta^t)}$$

7. Update the parameters:

$$\mu^{t+1} = \mu^t - \alpha \Delta_\mu$$
$$\rho^{t+1} = \rho^t - \alpha \Delta_\rho$$

where α is the learning rate. The authors are not explicit about the stopping criteria for the optimization, but there are a number of options for how to do this. One is tracking error rate, and in any case this is a standard problem with many ways to address it available.

Remark 2.2. The authors of the paper use only a single monte carlo sample in each step of the gradient descent. This would seem to be a poor approximation to the true cost, and being a linear combination of the f's over different samples, and so computing a single gradient in closed form enables us without too much more computation to add more monte carlo samples to each step of the optimization. This is a point that deserves some thought.

We list again the free parameters of this method:

- The number N of neurons and the choice of layers and activation functions
- the parametric form of the posterior $q(\mathbf{w}|\theta)$
- The form and parameters of the prior $P(\mathbf{w})$
- the learning rate α
- step size or method for computing partials
- the stopping criteria for the optimization

The output of the above optimization is a $\theta = (\mu, \rho)$ giving an informed posterior density $q(\mathbf{w}|\theta)$ that can then be used to generate samples, evaluate the uncertainty in the parameters, etc.

Remark 2.3. The authors include interesting discussion of removing nodes based on various measures of certainty, e.g. $|\mu_i|/\sigma_i$ (see §5.1 in their paper). This seems of potentially good use in our application below.

It is worth noting that this method doubles the number of parameters needing to be fit compared to a non-Bayesian NN.

3 TPL Implementation

We will be adapting the above algorithm for use in predicting variables related to HAB's in Detroit lake using various time series provided by Salem. These series are, relative to more common applications of NN's, rather sparse, and as such we must be mindful overfitting by using NN's with too many free parameters relative to the data. The need to mitigate the risk of overfitting, and to quantify the uncertainty in the data was the main motivation for using a BNN approach. Further, in implementing the BNN, we make certain choices specific to our setting, such as limiting the number of neurons N, and other choices below.

3.1 Modifications to Algorithm

The authors of the paper use a so-called "RELU" activation function, which is just a piecewise linear function (integral of heavyside). We choose to use the more standard sigmoid function, as the reason for the choice of RELU is to avoid issues that come from performing gradient descent over networks with thousands of units and hundreds of thousands or more of parameters, issues that we will not be encountering in our setting.

3.1.1 Priors and Posteriors

We let $\mathbf{w} = (\mathbf{w}, \mathbf{b})$, which as remarked above is equivalent to what the authors do and just a change in notation. This choice in notation will come out in the optimization where we compute partials. We use the same posterior as the authors, independent Gaussians for each parameter, but for the prior it is less clear that we want to push most edges towards zero weight. In a large network, this seems sensible, however in our case of much more limited data, and where most variables are being included on a scientific basis of understanding of their role in the physical system, it is less obvious that we should try to push weights towards zero. Moreover, using a prior with easily computable closed forms for derivatives is preferable and simpler for a first implementation. As such, for a first run,

we will use a simple multivariate Gaussian prior with zero mean:

$$P(\mathbf{w}) = \frac{1}{\left(\sqrt{2\pi \prod_j \bar{\sigma}_j}\right)^n} e^{-\frac{1}{2}\sum_j \frac{\mathbf{w}_j^2}{\bar{\sigma}_j^2}}. \tag{6}$$

Initially, we choose all the variances $\bar{\sigma}_j = 1$, but given the available knowledge about the system, it is likely that with some thought we can adjust these priors the be better informed. On the other hand, care must be taken to not fit these priors to the data, as discussed in the paper (§3.3), and as best practice for making predictions under uncertainty.

3.1.2 Use in BMA

The output is a density on the affine transform parameters, and so in that regard could partially obviate the need for any model averaging procedure over the topology of the network (see the filtering by $|\mu|/\sigma$ mentioned above). However, the number of nodes in each layer and the number of layers is chosen and fixed, and these parameters could be varied and the output BNN's fed into a larger BMA scheme.

3.2 Implementation

3.2.1 Preprocessing of Data and Choice of Response

[discussion]

3.2.2 Algorithm

Given the above modifications, the algorithm proceeds as in the google paper. We divide the algorithm into a few modular components, given as follows: [description of functions in algorithm]

The explicit gradient descent goes as follows:

Supposing $\mathbf{w} \in \mathbb{R}^n$, we have $\theta \in \mathbb{R}^{2n}$, and $f : \mathbb{R}^{3n} \to \mathbb{R}$. Recall the definition of f from (5):

$$f(\mathbf{w}, \theta) = \log(q(\mathbf{w}|\theta)) - \log(P(\mathbf{w})) - \log(P(D|\mathbf{w})).$$

In evaluating the gradients of f below, the specific functional forms chosen for $q(\mathbf{w}|\theta)$ and $P(\mathbf{w})$ are used, while for the last term, the partials with respect to parameters for quadratic loss is computed using *backpropogation* ([2], Lemma 1), which is the origin of the title of the paper.

First, we have

$$q(\mathbf{w}|\theta) = \frac{1}{\left(\sqrt{2\pi \prod_j \sigma_j}\right)^n} e^{-\frac{1}{2}\sum_j \frac{(\mathbf{w}_j - \mu_j)^2}{\sigma_j^2}},$$

so that

$$\log(q(\mathbf{w}|\theta)) = -\frac{n}{2}\log(2\pi) - \frac{n}{2}\sum_j \log(\sigma_j) - \frac{1}{2}\sum_j \frac{(\mathbf{w}_j - \mu_j)^2}{\sigma_j^2}$$

where from our definition of σ above, $\sigma_j = \log(1 + e^{\rho_j})$. Therefor,

$$\frac{\partial}{\partial \mathbf{w}_j}\log(q(\mathbf{w}|\theta)) = -\frac{\mathbf{w}_j - \mu_j}{(\log(1 + e^{\rho_j}))^2}$$

$$\frac{\partial}{\partial \mu_j}\log(q(\mathbf{w}|\theta)) = \frac{\mathbf{w}_j - \mu_j}{(\log(1 + e^{\rho_j}))^2}$$

$$\frac{\partial}{\partial \rho_j}\log(q(\mathbf{w}|\theta)) = -\frac{n}{2}\frac{e^{\rho_j}}{(1 + e^{\rho_j})(\log(1 + e^{\rho_j}))} + \frac{(\mathbf{w}_j - \mu_j)^2 e^{\rho_j}}{(1 + e^{\rho_j})(\log(1 + e^{\rho_j}))^3}$$

Next, from (6), we have

$$\log(P(\mathbf{w}) = C - \frac{1}{2}\sum_j \left(\frac{\mathbf{w}_j}{\bar{\sigma}_j}\right)^2,$$

and so

$$\frac{\partial}{\partial \mathbf{w}_j}\log(P(\mathbf{w})) = -\frac{\mathbf{w}_j}{\bar{\sigma}_j}$$

$$\frac{\partial}{\partial \mu_j}\log(P(\mathbf{w})) = 0$$

$$\frac{\partial}{\partial \rho_j}\log(P(\mathbf{w})) = 0,$$

(recall that above, for a first run, we use $\bar{\sigma}_j = 1$).

Last, from [2] we compute the partials of the error term

$$\mathbf{Q}(\mathbf{w}) \equiv \log(P(D|\mathbf{w})) = \sum_{(x,y)_k \in D} \|\bar{x}_k^N - y_k\|^2 \equiv \sum_{(x,y)_k \in D} Q(\mathbf{w}, x, y)$$

by backpropogation: We first let $G(x) = \frac{1}{1+e^{-x}}$ and recall the input layer is L_0 and the output layer is L_N and that the activation function for L_N is $g(x) = x$. For each layer L_k let $|L_k|$ be the number of neurons it contains. Then for $1 \leq k \leq N$ we have $W^k \in \mathbb{R}^{|L_k| \times |L_{k-1}|}$ and $\mathbf{b}^k \in \mathbb{R}^{|L_k|}$. From Lemma 1 in [2], we have for each $(x, y) \in D$

$$\frac{\partial Q}{\partial \mathbf{b}_j^N} = (\bar{x}_j^N - y_j)$$

$$\frac{\partial Q}{\partial \mathbf{b}_j^k} = G'\left(\left[W^k \bar{x}^{k-1}\right]_j + \mathbf{b}_j^k\right)\left[W^{k+1}\frac{\partial Q}{\partial \mathbf{b}^{k+1}}\right]_j$$

for $1 \leq k \leq N-1$, and

$$\frac{\partial Q}{\partial W_{i,j}^k} = \frac{\partial Q}{\partial \mathbf{b}_i^k} \bar{x}_j^{k-1}$$

for $1 \leq k \leq N$. We then sum these over $(x, y) \in D$ to obtain the partials of $\mathbf{Q}(\mathbf{w})$. Here we see the reason for the term "backpropagation:" we iteratively compute the above partials by propagating backwards from the output of the NN. The pseudo code is (see page 15 of [2]), for a fixed $(x, y) \in D$

1. Given parameters defining the NN, propagate the data sample x through the network, storing the z^k and outputs at each layer (forward iteration):

   (a) Let $a[0] = x$

   (b)

   For $k = 1 : N-1$

   $z[k] = W^k a[k-1] + \mathbf{b}^k$

   $a[k] = G(z[k])$

   $S[k] = diag(G'(z[k]))$

   End

   (c)

   $z[N] = W^N a[N-1] + \mathbf{b}^N$

   $a[N] = z[N]$

   $S[N] = I$

2. Compute the partials using backward propagation above (backward iteration).

   (a) $\frac{\partial Q}{\partial \mathbf{b}^N} = S[N](a[N] - y)$

   (b)

   For $k = N-1 : 1$

   $$\frac{\partial Q}{\partial \mathbf{b}^k} = S[k](W^{k+1})^T \frac{\partial Q}{\partial \mathbf{b}^{k+1}}$$

   End

   (c)

   For $k = 1 : N$

   $$\frac{\partial Q}{\partial W_{i,j}^k} = \frac{\partial Q}{\partial \mathbf{b}_i^k} a[k-1]_j$$

   End

Remark 3.1. Here we see the reason for stochastic gradient descent: When we attempt to optimize below, every time we compute the gradient we need to run the forward and backward iterations through the net for each data sample. If there are many, this can be prohibitive. The authors mention the so called *minibatching* technique as a compromise between a fully stochastic gradient calculation, where a single data sample is chosen at random, and a full summation over all data samples. Any technique can be used in the optimization below, and testing will need to be done to see if speedup is needed. Given the sparsity of the data (at least relative to what is considered large in the NN literature), this may prove unnecessary.

With these partials calculated, our gradient descent runs as follows:

At each step we have two $1 \times N$ arrays of arrays $W_{i,j}^k$ and $\mathbf{b}^k$, where $W^k \in \mathbb{R}^{|L_k| \times |L_{k-1}|}$ and $\mathbf{b}^k \in \mathbb{R}^{|L_k|}$. We reindex the W^k and the $\mathbf{b}^k$ to a single vector $\mathbf{w} \in \mathbb{R}^{np}$ where $np = \sum_{k=1}^{N} (|L_{k-1}| + 1)|L_k|$. This reindexing will need to be reversed each time compute the gradients of the quadratic terms using backpropagation as above.

We first initialize μ and ρ, both in $\mathbb{R}^{np}$. At each step t of the optimization we

1. Generate standard normals $\epsilon \in \mathbb{R}^{np}$
2. Let $\mathbf{w}_i^t = \mu_i^t + \log(1 + e^{\rho_i^t})\epsilon_i$
3. Let $\theta^t = (\mu^t, \rho^t)$
4. Let $f = f(\mathbf{w}^t|\theta^t) = \log q(\mathbf{w}^t|\theta) - \log P(\mathbf{w}^t) - \mathbf{Q}(\mathbf{w}^t)$
5. Compute the partial derivatives of $\mathbf{Q}(\mathbf{w}^t)$ using backpropagation:
   (a) Extract the W^k and $\mathbf{b}^k$
   (b) Run the forward and backward iterations as above and store the outputs
   (c) Reindex as above and store as $\left(\frac{\partial \mathbf{Q}}{\partial \mathbf{w}_j^t}\right)$
6. Calculate the gradient of f with respect to μ (recalling calculations above):

$$\begin{aligned}\frac{\partial f}{\partial \mu_i} &= \frac{\partial f}{\partial \mathbf{w}_i} + \frac{\partial f}{\partial \theta_v} \\ &= -\frac{\mathbf{w}_i - \mu_i}{(\log(1 + e^{\rho_i}))^2} - \frac{\mathbf{w}_i}{\bar{\sigma}_i} + \frac{\partial \mathbf{Q}}{\partial \mathbf{w}_i^t} + \frac{\mathbf{w}_i - \mu_i}{(\log(1 + e^{\rho_i}))^2} \\ &= \frac{\partial \mathbf{Q}}{\partial \mathbf{w}_i^t} - \frac{\mathbf{w}_i}{\bar{\sigma}_i}\end{aligned}$$

$$\Delta_\mu = \left\langle \frac{\partial f}{\partial \mu_i} \right\rangle \Big|_{(\mathbf{w}^t, \theta^t)}$$

7. Calculate the gradient of f with respect to ρ:

$$\begin{aligned}\frac{\partial f}{\partial \rho_i} &= \frac{\partial f}{\partial \mathbf{w}_i}\frac{\epsilon_i^t}{1+e^{\rho_i^t}} + \frac{\partial f}{\partial \theta_v} \\ &= \left(-\frac{\mathbf{w}_i - \mu_i}{(\log(1+e^{\rho_i}))^2} - \frac{\mathbf{w}_i}{\bar{\sigma}_i} + \frac{\partial \mathbf{Q}}{\partial \mathbf{w}_i^t}\right)\frac{\epsilon_i^t}{1+e^{\rho_i^t}} + \frac{(\mathbf{w}_i - \mu_i)^2 e^{\rho_i}}{(1+e^{\rho_i})(\log(1+e^{\rho_i}))^3} \\ &- \frac{n}{2}\frac{e^{\rho_i}}{(1+e^{\rho_i})(\log(1+e^{\rho_i}))}\end{aligned}$$

$$\Delta_\rho = \left\langle \frac{\partial f}{\partial \rho_i} \right\rangle \Big|_{(\mathbf{w}^t, \theta^t)}$$

8. Update the parameters:

$$\mu^{t+1} = \mu^t - \alpha \Delta_\mu$$
$$\rho^{t+1} = \rho^t - \alpha \Delta_\rho$$

where α is the learning rate.

For stopping criteria...

References

[1] Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. Weight uncertainty in neural networks. In *Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, pages 1613–1622. JMLR.org, 2015.

[2] Catherine F. Higham and Desmond J. Higham. Deep Learning: An Introduction for Applied Mathematicians. *arXiv e-prints*, page arXiv:1801.05894, January 2018.