Continuous Optimization of Hyper-Parameters

Yoshua Bengio
Département d'informatique et recherche opérationnelle
Université de Montréal
Montréal, Québec, Canada, H3C 3J7
bengioy@iro.umontreal.ca

May 31, 1999

Technical report #1144,
Département d'informatique et recherche opérationnelle
Université de Montréal

Abstract

Many machine learning algorithms can be formulated as the minimization of a training criterion which involves (1) "training errors" on each training example and (2) some hyper-parameters, which are kept fixed during this minimization. When there is only a single hyper-parameter one can easily explore how its value affects a model selection criterion (that is not the same as the training criterion, and is used to select hyper-parameters). In this paper we present a methodology to select many hyper-parameters that is based on the computation of the gradient of a model selection criterion with respect to the hyper-parameters. We first consider the case of a training criterion that is quadratic in the parameters. In that case, the gradient of the selection criterion with respect to the hyper-parameters is efficiently computed by back-propagating through a Cholesky decomposition. In the more general case, we show that the implicit function theorem can be used to derive a formula for the hyper-parameter gradient, but this formula requires the computation of second derivatives of the training criterion.

1 Introduction

Machine learning algorithms pick a function f from a set of functions $\mathcal{F}$ in order to minimize something that cannot be measured, only estimated, that is the expected generalization performance of the chosen function. Many machine learning algorithms can be formulated as the minimization of a **training criterion** which involves, on the one hand, training errors on each training example and, on the other hand, some hyper-parameters, which are kept fixed during this minimization. For example, in the regularization framework (Tikhonov and Arsenin, 1977; Poggio, Torre and Koch, 1985), one hyper-parameter controls the strength of the penalty term and thus the *capacity* (Vapnik, 1995) of the system: a larger penalty term reduces the "complexity" of the resulting function (forces the solution f to lie in a subset of $\mathcal{F}$). A very common example is *weight decay* (Hinton, 1987) used with neural networks and linear regression (also known as ridge regression (Hoerl and Kennard, 1970), in that case): the penalty term is the hyper-parameter times the norm of the parameter vector. Increasing the penalty term (increasing the weight decay hyper-parameter) corresponds to reducing the *effective capacity* (Guyon et al., 1992), which may improve generalization. A regularization term can also be interpreted as an a-priori probability distribution on $\mathcal{F}$: in that case the weight decay is a scale parameter (e.g., inverse variance) of that distribution.

A **model selection criterion** is not the same as the training criterion: it is a criterion used to select hyper-parameters, more generally to compare and choose among models which may have a different capacity. Many model selection criteria have been proposed in the past (Vapnik, 1982; Akaike, 1974; Craven and Wahba, 1979). When there is only a single hyper-parameter one can easily explore how its value affects the model selection criterion: typically one tries a finite number of values of the hyper-parameter and picks the one which gives the lowest value of the model selection criterion.

In this paper we present a methodology to simultaneously select many hyper-parameters using the gradient of the model selection criterion with respect to the hyper-parameters. This methodology can be applied when some differentiability and continuity conditions of the training criterion are satisfied. The use of multiple hyper-parameters has already been proposed in the Bayesian literature: one hyper-parameter per input feature was used to control the prior on the weights associated to that input feature (MacKay and Neal, 1994; Neal, 1998). In this case, the hyper-parameters can be interpreted as scale parameters for the prior distribution on the parameters, for different directions in parameter space. In a neural network or a linear regression, this is equivalent to having a different weight decay for the weights from each of the inputs. A large weight decay on one of the inputs effectively forces the corresponding weights to very small values. In contrast to feature selection algorithms such as the classical forward and backward selection methods used for linear regression, an algorithm for selecting such hyper-parameters explores a continuous rather than a discrete space.

In Section 2, we formalize the notions of hyper-parameters, training criterion, and model selection criterion, and we give examples of training and selection criteria for which the proposed methodology could be applied. In Section 3, we show how to compute the gradient of a model selection criterion with respect to the hyper-parameters. In Section 4, we extend

the result to a more general setting, which requires computing second derivatives. In the conclusion, we briefly describe the results of preliminary experiments performed with the proposed methodology (described in more details in (Bengio and Latendresse, 1999; Bengio and Dugas, 1999)), and we raise some important open questions concerning the kind of "over-fitting" that can occur with the proposed methodology, and how the notion of capacity might be extended to deal with the additional degrees of freedom conferred by the choice of many hyper-parameters.

2 Objective Functions for Hyper-Parameters

We are given a set of independent data points $D = \{z_1, \ldots, z_T\}$, each generated by an unknown distribution $P(Z)$. We want to choose a function f from a given set of functions $\mathcal{F}$ that minimizes the expectation $E_Z(Q(f, Z))$ of a given cost functional $Q(f, Z)$. In supervised learning problems, we have input/output pairs $Z = (X, Y)$, with $X \in \mathcal{X}$, $Y \in \mathcal{Y}$, and $f : \mathcal{X} \to \mathcal{Y}$. For example, we will consider the case of the quadratic cost, with real-valued vectors $\mathcal{Y} \subseteq \mathcal{R}^m$ and

$$Q(f, (X, Y)) = \frac{1}{2}(f(X) - Y)'(f(X) - Y).$$

Note that in this case, we are trying to pick $f \in \mathcal{F}$ which is closest in L_2 norm under $E_X[.]$ to the conditional expectation $E_Y[Y|X]$ of Y given X (see as a function of X).
Let us use the $\tilde{\mathbf{x}}$ notation for extending vectors $\mathbf{x} \in \mathcal{R}^n$ by one constant element:

$$\tilde{\mathbf{x}} = (\mathbf{x}, 1) = (x_1, \ldots, x_n, 1).$$

In the next section, we will provide a formulation for the cases of constant and affine function sets $\mathcal{F}$, e.g.,

$$\mathcal{F}^{constant} = \{f : \mathcal{R}^n \to \mathcal{R}^m | f(\mathbf{x}) = \theta, \mathbf{x} \in \mathcal{R}^n, \theta \in \mathcal{R}^m\}$$
$$\mathcal{F}^{affine} = \{f : \mathcal{R}^n \to \mathcal{R}^m | f(\mathbf{x}) = \Theta\tilde{\mathbf{x}}, \mathbf{x} \in \mathcal{R}^n, \Theta \in \mathcal{R}^{m \times (n+1)}\}$$

In section 4, we will consider more general classes of functions and cost functions, which may be applied to the case of multi-layer neural networks, for example.

2.1 Training Criteria

In its most general form, a **training criterion** C is any real-valued function of the set of empirical costs $Q(f, z_i)$:

$$C = c(Q(f, z_1), Q(f, z_2), \ldots, Q(f, z_T))$$

In the cases in which we are interested, C is parameterized by a vector of real-valued hyper-parameters $\lambda = (\lambda_1, \ldots, \lambda_q)$, and since we concentrate on gradient-based learning, we will consider that the choice of f within $\mathcal{F}$ is equivalent to the choice of a vector of parameters $\theta \in \Omega \subset \mathcal{R}^s$. We can rewrite the **training criterion** C in the more compact form

$$C = c(\theta, \lambda, D).$$

which emphasizes the dependence on the training data D and the hyper-parameters λ. The proposed method will rely on the assumption that C is continuous and differentiable almost everywhere with respect to θ and λ.
When the hyper-parameters are fixed, the learning algorithm attempts to perform the following minimization:

$$\theta(\lambda, D) = \text{argmin}_\theta c(\theta, \lambda, D).$$

More generally, a function f is chosen as follows,

$$f_{\lambda,D} = \text{argmin}_{f \in \mathcal{F}} c(f, \lambda, D)$$

where we have again changed the semantics of the first argument of c, from θ to f.
A particularly simple training criterion (without hyper-parameters) is the so-called *empirical risk*, which is simply the average of training errors:

$$\text{empirical risk} = \frac{1}{|D|} \sum_{z_i \in D} Q(f, z_i).$$

Bounds on the generalization error can be computed when this criterion is minimized and the capacity of $\mathcal{F}$ is smaller than $|D|$. These bounds depend on the capacity of $\mathcal{F}$, which measures the diversity of the functions f within $\mathcal{F}$ with respect to the cost criterion $Q(f, z)$. The larger the capacity, the larger the optimism in the estimate of generalization error provided by the empirical risk.
An example of training criterion with hyper-parameters is the following:

$$C = \sum_{(x_i, y_i) \in D} w_i(\lambda)(F(x_i, \theta) - y_i)^2 + \theta' A(\lambda)\theta \tag{1}$$

where the hyper-parameters provide different quadratic penalties to different parameters (with the matrix A), and different weights to different training patterns (with $w_i(\lambda)$), (as in (Bengio and Dugas, 1999; Bengio and Latendresse, 1999)).

2.2 Model Selection Criteria

The **model selection criterion** E is a criterion that is used to select hyper-parameters or more generally to choose one model among several models. Ideally, it should be the expected generalization error (when using a particular λ), but we don't know the true distribution of the data, $P(Z)$, so many alternatives have been proposed, which are either approximations, bounds, or approximate bounds.
Most model selection criteria have been proposed for selecting a single hyper-parameter that controls the "complexity" of the class of functions in which the learning algorithms finds a solution. For example, the minimum description length principle (Rissanen, 1990) states that the optimal trade-off will occur when the total cost of compressing the data using the model and compressing the model is minimized (however, this in general requires some a-priori choices about the distribution of the model, e.g. , of the parameters). Some methods

rely on theoretical bounds on generalization error, such as the structural risk minimization approach of (Vapnik, 1982; Vapnik, 1995), the Akaike Information Criterion (Akaike, 1974) and the generalized cross-validation criterion (Craven and Wahba, 1979) (not to be mixed with the **cross-validation** criterion).
The above criteria are based on performance on the training data and some measure of complexity that depends on the parameterization of $\mathcal{F}$. Another type of criteria are those based on held-out data, also known as cross-validation estimates of generalization error. These are almost unbiased estimates of generalization error (Vapnik, 1995) obtained by testing f on data not used to choose f within $\mathcal{F}$. If lots of data are available, the total data set is simply split into a training set S_1 and a validation set S_2, and the performance on the validation set is used to compare models and choose the value of the hyper-parameter(s). Otherwise, there are different resampling strategies that have been proposed in which several choices of the split $S_1 \cup S_2 = D$ are made and a different function is trained for each of these partitions, and the average of the errors on the validation sets is used as an estimate of generalization error. Note that it is an estimate of generalization error when training with a set of size $|S_1|$ rather than a set of size $|D|$. An extreme case is the well-known leave-one-out estimate, in which $|S_1| = |D| - 1$ and all the $|D|$ possible such partitions are considered. Another popular alternative is the K-fold cross-validation estimate (Efron and Tibshirani, 1993), in which we consider K partitions of D, $S_1^1 \cup S_2^1$, $S_1^2 \cup S_2^2$, ... and $S_1^K \cup S_2^K$.
Formally, the cross-validation criterion is

$$E_{cv}(\lambda, D) = \frac{1}{K} \sum_i \frac{1}{|S_2^i|} \sum_{z_t \in S_2^i} Q(f_{\lambda, S_1^i}, z_t).$$

One way to understand this criterion is as an analogue of the empirical risk defined above, but with respect to hyper-parameters rather than with respect to parameters θ (or f). The analogy goes as follows: when f is fixed, the empirical risk is an unbiased estimate of the generalization error of f (but of course it becomes an optimistic estimate when f is chosen within $\mathcal{F}$ to minimize the empirical risk). Similarly, when λ is fixed, the cross-validation criterion is an almost unbiased estimate of the generalization error of $f_{\lambda,D}$ (actually it is a slightly pessimistic estimate because $|S_1^i| < |D|$). Likewise, when λ is chosen to minimize the cross-validation criterion, the value of this criterion becomes a more optimistic estimate. Likewise, we can expect that if the set of values that λ can take is large (or more precisely, when there is is a great diversity of functions $f_{\lambda,D}$ that can be obtained for different values of λ) we can expect that this optimism will be greater, i.e., there is more risk of overfitting the hyper-parameters.
In this paper, we discuss the case in which λ is a real-valued vector and we compute the gradient $\frac{\partial E}{\partial \lambda}$ in order to choose λ (with numerical optimization methods).

3 Optimizing Hyper-Parameters for a Quadratic Training Criterion

In this section we analyze the simpler case in which the training criterion C is a quadratic polynomial of the parameters θ. The dependence on the hyper-parameters λ can be of higher order, as long as it is continuous and differentiable almost everywhere (see for example (Bottou, 1998) for more detailed technical conditions sufficient for stochastic gradient descent):

$$C = a(\lambda) + b(\lambda)'\theta + \frac{1}{2}\theta' H(\lambda)\theta \tag{2}$$

where $\theta, b \in \mathcal{R}^s$, $a \in \mathcal{R}$, and $H \in \mathcal{R}^{s \times s}$. Later on in this section we will consider particular cases corresponding to linear regression with different weight decays on the parameters and in (Bengio and Dugas, 1999) we consider hyper-parameters that control the weights that C puts on different training examples.
For a minimum of the above quadratic training criterion to exist requires that H be positive definite. This minimum is obtained by solving the linear system

$$\frac{\partial C}{\partial \theta} = b + H\theta = 0 \tag{3}$$

which yields the solution

$$\theta(\lambda) = -H^{-1}(\lambda)b(\lambda). \tag{4}$$

In this paper we will use cross-validation types criteria to illustrate the application of the method, but any model selection criterion can be used, as long as it is continuous and differentiable with respect to θ and λ.
Therefore the gradient of the model selection criterion E with respect to λ is

$$\left.\frac{\partial E}{\partial \lambda}\right|_{\lambda} = \left.\frac{\partial E}{\partial \theta}\right|_{\theta,\lambda} \left.\frac{\partial \theta}{\partial \lambda}\right|_{\lambda} + \left.\frac{\partial E}{\partial \lambda}\right|_{\theta,\lambda}$$

where we have denoted by $\left.\frac{\partial y}{\partial x}\right|_{x,v}$ the partial derivative of y with respect to x when y is seen as a function of x and v (i.e, keeping x and v fixed, so that even if v is a function x, that dependency is not accounted in that derivative). Later on in this text we will drop the $|_{x,v}$ notation when all the dependencies are taken into account (i.e, this is $\left.\frac{\partial y}{\partial x}\right|_{x}$). In the above equation, we distinguish $\left.\frac{\partial E}{\partial \lambda}\right|_{\lambda}$, which takes into account the influence of λ on E through all paths, including θ, from $\left.\frac{\partial E}{\partial \lambda}\right|_{\theta,\lambda}$, in which the influence of λ through θ is not taken into account because we take θ as fixed.
For example, in the case of the cross-validation criteria,

$$\left.\frac{\partial E_{cv}}{\partial \lambda}\right|_{\theta,\lambda} = 0$$

and

$$\frac{\partial E_{cv}}{\partial \theta} = \frac{1}{K} \sum_i \frac{1}{|S_2^i|} \sum_{z_t \in S_2^i} \frac{\partial Q(\theta, z_t)}{\partial \theta}.$$

Since the gradient $\frac{\partial Q(\theta, z_t)}{\partial \theta}$ is commonly used to obtain a minimum of the empirical risk, the only difficulty that remains is the computation of the gradient of the parameters with respect to the hyper-parameters, $\frac{\partial \theta}{\partial \lambda}$.
In the quadratic case, the influence of λ on θ is spelled out explicitly by equation 4, yielding

$$\frac{\partial \theta_i}{\partial \lambda} = -\sum_j \frac{\partial H^{-1}_{i,j}}{\partial \lambda} b_j - \sum_j H^{-1}_{i,j} \frac{\partial b_j}{\partial \lambda} \tag{5}$$

The second sum can be readily computed by multiplying H^{-1} with the gradient of b with respect to λ. The first sum is less trivial: we consider two approaches in the next subsection, and in the subsection that follows the next one, we present a better alternative that is based on equation 3 instead of equation 4.

3.1 Gradient of Matrix Inversion

One way to compute the above gradient is based on the computation of gradients through the inverse of a matrix. A general but inefficient solution is the following:

$$\frac{\partial H^{-1}_{i,j}}{\partial \lambda} = \sum_{k,l} \frac{\partial H^{-1}_{i,j}}{\partial H_{k,l}} \frac{\partial H_{k,l}}{\partial \lambda}$$

Let us consider a square matrix $B = A^{-1}$. We want to compute $\frac{\partial B_{i,j}}{\partial A_{k,l}}$. Consider the expression of the inverse and the determinant in terms of the cofactors and minors:

$$B_{i,j} = \frac{(-1)^{i+j} |minor(A, j, i)|}{|A|}$$

where $minor(A, j, i)$ denotes the "minor matrix", obtained by removing the j-th row and the i-th column from A, and the $|A|$ is the determinant of A, which can be written

$$|A| = \sum_k A_{kl} (-1)^{k+l} |minor(A, k, l)| = A_{kl} (-1)^{k+l} |minor(A, k, l)| + const$$

where *const* does not depend of A_{kl}. Applying the chain rule and simple algebra then yields

$$\frac{\partial B_{i,j}}{\partial A_{k,l}} = -B_{i,j} B_{l,k} + I_{i \neq l, j \neq k} B_{i,j} minor(A, j, i)^{-1}_{l',k'}, \tag{6}$$

where the indices (l', k') in the above equation refer to the position within a minor matrix that corresponds to the position (l, k) in the original matrix A (note $l \neq i$ and $k \neq j$).
Unfortunately, the computation of this gradient requires $O(s^5)$ multiply-add operations for an $s \times s$ matrix, which is much more that the computation of the inverse ($\mathrm{O}(s^3)$).

A better solution is based on the following equality:

$$AB = I$$

where I is the $s \times s$ identity matrix. This implies, by differentiating with respect to a scalar x,

$$\frac{\partial A}{\partial x} B + A \frac{\partial B}{\partial x} = 0.$$

Isolating $\frac{\partial B}{\partial x}$, we get

$$\frac{\partial B}{\partial x} = -B \frac{\partial A}{\partial x} B$$

so that in particular, for equation 5, we obtain the required gradient

$$\frac{\partial H^{-1}}{\partial \lambda} = -H^{-1} \frac{\partial H}{\partial \lambda} H^{-1} \tag{7}$$

which requires only $O(2s^3)$ multiply-add operations.

3.2 Gradient Through the Cholesky Decomposition

An even better solution [1] is to return to equation 3, which can be solved in $O(s^3/3)$ multiply-add operations (when $\theta \in \mathcal{R}^s$). The idea is to *back-propagate* gradients through each of the operations performed to solve the linear system (just as back-propagation of gradients is used to compute derivatives through a multi-layer neural network). The objective is to compute the gradient of θ with respect to H and b, which are themselves functions of λ, in order to compute $\frac{\partial E}{\partial \lambda}$ from $\frac{\partial \theta}{\partial \lambda}$. The back-propagation will just cost the same as the linear system solution, i.e., $O(s^3/3)$ operations, so this is the approach that we have kept for our implementation.

Since H is the Hessian matrix, with second derivatives of the training criterion with respect to the parameters θ, it will be not only positive definite but also **symmetric** (under adequate conditions of continuity of C with respect to θ). Therefore, the linear system in equation 3 enjoys a particularly simple and elegant solution through the Cholesky decomposition of the matrix H. To solve the system, we will also assume that H is full rank, which is likely if the hyper-parameters provide some sort of weight decay. The Cholesky decomposition of a symmetric positive definite matrix H gives

$$H = LL'$$

where L is a lower diagonal matrix (with zeros above the diagonal). It is computed in time $O(s^3)$ as follows:

`for` $i = 1, \ldots, s$

$$L_{i,i} = \sqrt{H_{i,i} - \sum_{k=1}^{i-1} L_{i,k}^2}$$

[1] which was suggested by Léon Bottou

for $j = i+1, \ldots, s$
$$L_{j,i} = (H_{i,j} - \sum_{k=1}^{i-1} L_{i,k} L_{j,k}) / L_{i,i}$$

Once the Cholesky decomposition is achieved, the linear system $LL'\theta = -b$ can be easily solved, in two back-substitution steps: first solve $Lu = -b$, then solve $L'\theta = u$. First step, iterating once forward through the rows of L:

for $i = 1, \ldots, s$
$$u_i = (-b_i - \sum_{k=1}^{i-1} L_{i,k} u_k) / L_{i,i}$$

Second step, iterating once backward through the rows of L:

for $i = s, \ldots, 1$
$$\theta_i = (u_i - \sum_{k=i+1}^{s} L_{k,i} \theta_k) / L_{i,i}$$

where it can be noticed that the diagonal elements of L must be positive (otherwise the Hessian H is not full rank).
The computation of the gradient of θ with respect to the elements of H and b proceed in exactly the reverse order. We start by back-propagating through the back-substitution steps, and then through the Cholesky decomposition.
The back-propagation through the two back-substitution steps is the following. First back-propagate through the solution of $L'\theta = u$:

initialize $\texttt{dEdtheta} \leftarrow \left.\frac{\partial E}{\partial \theta}\right|_{\theta_1,\ldots,\theta_s}$
initialize $\texttt{dEdL} \leftarrow 0$
for $i = 1, \ldots, s$
    $\texttt{dEdu}_i \leftarrow \texttt{dEdtheta}_i / L_{i,i}$
    $\texttt{dEdL}_{i,i} \leftarrow \texttt{dEdL}_{i,i} - \texttt{dEdtheta}_i \, \theta_i / L_{i,i}$
    for $k = i+1 \ldots s$
        $\texttt{dEdtheta}_k \leftarrow \texttt{dEdtheta}_k - \texttt{dEdtheta}_i \, L_{k,i} / L_{i,i}$
        $\texttt{dEdL}_{k,i} \leftarrow \texttt{dEdL}_{k,i} - \texttt{dEdtheta}_i \, \theta_k / L_{i,i}$

Then back-propagate through the solution of $Lu = -b$:

for $i = s, \ldots, 1$
    $\frac{\partial E}{\partial b_i} \leftarrow -\texttt{dEdu}_i / L_{i,i}$
    $\texttt{dEdL}_{i,i} \leftarrow \texttt{dEdL}_{i,i} - \texttt{dEdu}_i \, u_i / L_{i,i}$
    for $k = 1, \ldots, i-1$
        $\texttt{dEdu}_k \leftarrow \texttt{dEdu}_k - \texttt{dEdu}_i \, L_{i,k} / L_{i,i}$
        $\texttt{dEdL}_{i,k} \leftarrow \texttt{dEdL}_{i,k} - \texttt{dEdu}_i \, u_k / L_{i,i}$

The above algorithm gives us the gradient of the model selection criterion E with respect to

coefficient $b(\lambda)$ of the training criterion, as well as with respect to the lower diagonal matrix L.
Finally, we back-propagate through the Cholesky decomposition, to convert the gradients with respect to L into gradients with respect to the Hessian $H(\lambda)$. Note that for another application it has already been proposed to differentiate the Cholesky algorithm (Smith, 1995).

```
for i = s, ..., 1
        for j = s, ..., i+1
                dEdL_{i,i} ← dEdL_{i,i} − dEdL_{j,i} L_{j,i} / L_{i,i}
                ∂E/∂H_{i,j} ← dEdL_{j,i} / L_{i,i}
                for k = 1, ..., i−1
                        dEdL_{i,k} ← dEdL_{i,k} − dEdL_{j,i} L_{j,k} / L_{i,i}
                        dEdL_{j,k} ← dEdL_{j,k} − dEdL_{j,i} L_{i,k} / L_{i,i}
        ∂E/∂H_{i,i} ← (1/2) dEdL_{i,i} / L_{i,i}
        for k = 1, ..., i−1
                dEdL_{i,k} ← dEdL_{i,k} − dEdL_{i,i} L_{i,k} / L_{i,i}
```

Note that we have only computed gradients with respect to the diagonal and upper diagonal of H because H is symmetric. Once we have the gradients of E with respect to b and H, we use the functional form of $b(\lambda)$ and $H(\lambda)$ to compute the gradient of E with respect to λ:

$$\frac{\partial E}{\partial \lambda} = \left.\frac{\partial E}{\partial \lambda}\right|_{\theta,\lambda} + \sum_i \frac{\partial E}{\partial b_i}\frac{\partial b_i}{\partial \lambda} + \sum_{i,j} \frac{\partial E}{\partial H_{i,j}}\frac{\partial H_{i,j}}{\partial \lambda}$$

Using this approach rather than the one described in the previous subsection, the overall computation of gradients is therefore $O(s^3/3)$ rather than $O(s^5)$. The most expensive step is the back-propagation through the Cholesky decomposition itself (three nested $O(s)$ loops). Note that this step may be shared if there are several linear systems to solve with the same matrix H.

3.3 Weight Decays for Linear Regression

In this subsection, we consider in more detail the particular case of multiple weight decays for linear regression. The data examples are input/output pairs $Z = (X, Y)$, with $X \in \mathcal{R}^n$ and $Y \in \mathcal{R}^m$. The cost function is the squared error

$$Q(f, (X, Y)) = \frac{1}{2}(f(X) - Y)'(f(X) - Y).$$

The class of functions is the affine class defined earlier

$$\mathcal{F}^{affine} = \{f : \mathcal{R}^n \to \mathcal{R}^m | f(\mathbf{x}) = \Theta\tilde{\mathbf{x}}, \mathbf{x} \in \mathcal{R}^n, \Theta \in \mathcal{R}^{m\times(n+1)}\}$$

with the parameter vector $\theta = vec(\Theta)$ formed by concatenating the rows of the weight matrix Θ.

All the developments that precede in this section can be applied in a straightforward way. We will consider here the K-fold cross-validation criterion as our model selection criterion. The hyper-parameter λ_j will be a weight decay associated to the j-th input variable (j-th element of the input vector X). The training criterion for the k-th partition is

$$C_k = \frac{1}{|S_1^k|} \sum_{(x_t,y_t)\in S_1^k} \frac{1}{2}(\Theta x_t - y_t)'(\Theta x_t - y_t) + \frac{1}{2}\sum_j \lambda_j \sum_i \Theta_{i,j}^2$$

The objective is to penalize separately each of the input variables. Each of the hyper-parameters gives a penalty for using the corresponding input variable. The use of multiple hyper-parameters has already been proposed in the Bayesian literature in a similar setup (MacKay and Neal, 1994; Neal, 1998): one hyper-parameter per input feature was used to control the prior on the weights associated to that input feature, for multi-layer neural networks. The selection of the hyper-parameters amounts to a kind of "soft variable selection" in which variables that do not help to generalize significantly better are penalized but not necessarily turned off completely. See (Bengio and Latendresse, 1999) for more discussion and experiments with this setup, including comparisons with more conventional variable selection schemes.

The training criterion is quadratic, as in equation 2, with coefficients

$$\begin{aligned} a &= \frac{1}{2}\sum_t y_t' y_t \\ b_{(ij)} &= -\sum_t y_{t,i} x_{t,j} \\ H_{(ij),(i'j')} &= \delta_{i,i'} \sum_t x_{t,j} x_{t,j'} + \delta_{i,i'}\delta_{j,j'}\lambda_j, \end{aligned} \tag{8}$$

where $\delta_{i,j} = 1$ when $i = j$ and 0 otherwise, and (ij) is an index for the coefficient vector b, the parameter vector θ, or the Hessian matrix H, corresponding to indices (i, j) in the weight matrix Θ, e.g., $(ij) = (i-1) \times s + j$.

From the above definition of the coefficients of C, we obtain their partial derivatives with respect to λ:

$$\begin{aligned} \frac{\partial b}{\partial \lambda} &= 0 \\ \frac{\partial H_{(ij),(i'j')}}{\partial \lambda_k} &= \delta_{i,i'}\delta_{j,j'}\delta_{j,k} \end{aligned}$$

Plugging the above definitions of the coefficients and their derivatives in the equations and algorithms of the previous subsection, we have therefore obtained an algorithm for computing the gradient of the model selection criterion with respect to the input weight decays of a linear regression. Note that here H is block-diagonal, with m identical blocks of size $(n+1)$, so the Cholesky decomposition (and similarly back-propagating through it) can be performed in time $O((s/m)^3/3)$ rather than $O(s^3/3)$, where m is the number of outputs (the dimension of the output variable).

4 Optimizing Hyper-Parameters for a Non-Quadratic Criterion

If the training criterion C is not quadratic in terms of the parameters θ, it will in general be necessary to apply an iterative numerical optimization algorithm to minimize the training criterion. In this section we will consider what happens after this minimization is performed, i.e., at a value of θ where $\frac{\partial C}{\partial \theta}$ is approximately zero and $\frac{\partial^2 C}{\partial \theta^2}$ is positive definite (otherwise we would not be at a minimum of C). We will use the implicit function theorem to obtain the derivative of θ with respect to λ at this point. Under appropriate conditions of continuity and differentiability, we have

$$F(\theta, \lambda) = 0 \rightarrow \frac{\partial \theta}{\partial \lambda} = -(\frac{\partial F}{\partial \theta})^{-1} \frac{\partial F}{\partial \lambda}$$

for a vector-valued function F of the vectors θ and λ. In particular we consider here

$$F(\theta, \lambda) = \frac{\partial C}{\partial \theta} = 0$$

so we obtain a **general formula for the gradient of the parameters with respect to the hyper-parameters**:

$$\frac{\partial \theta}{\partial \lambda} = -(\frac{\partial^2 C}{\partial \theta^2})^{-1} \frac{\partial^2 C}{\partial \lambda \partial \theta} = -H^{-1} \frac{\partial^2 C}{\partial \lambda \partial \theta}. \tag{9}$$

where H is the second derivative of the training criterion with respect to the parameters.
Let us see how this result relates to the special case of a quadratic training criterion, $C = a + b'\theta + \frac{1}{2}\theta' H \theta$:

$$\frac{\partial \theta}{\partial \lambda} = -H^{-1}(\frac{\partial b}{\partial \lambda} + \frac{\partial H}{\partial \lambda}\theta) = -H^{-1}\frac{\partial b}{\partial \lambda} + H^{-1}\frac{\partial H}{\partial \lambda} H^{-1} b$$

where we have substituted $\theta = -H^{-1}b$. Using the equality 7, we obtain the same formula as in eq. (5).
Let us consider more closely the case of a neural network with one hidden layer and with a squared error cost function:

$$Q(f, (X, Y)) = \frac{1}{2}(f(X) - Y)'(f(X) - Y)$$

and the class of functions is for example

$$\mathcal{F}^{mlp} = \{f : \mathcal{R}^n \rightarrow \mathcal{R}^m | f(\mathbf{x}) = V\tilde{\mathbf{h}},\ \mathbf{h} = \tanh(W\tilde{\mathbf{x}}),\ \mathbf{x} \in \mathcal{R}^n, \Theta \in \mathcal{R}^{m \times (n+1)}.\}$$

For example, if we want to use hyper-parameters for penalizing the use of inputs, we have as before

$$C_k = \frac{1}{|S_1^k|} \sum_{(x_t, y_t) \in S_1^k} \frac{1}{2}(f_\theta(x_t) - y_t)'(f_\theta(x_t) - y_t) + \frac{1}{2}\sum_j \lambda_j \sum_i W_{i,j}^2.$$

In this case, the cross-derivatives are easy to compute:

$$\frac{\partial^2 C}{\partial W_{i,j} \partial \lambda_k} = \delta_{k,j} W_{i,j}.$$

The Hessian and its inverse require more work, but can be done respectively in at most $O(s^2)$ and $O(s^3)$ operations. See for example (Bishop, 1992; Bishop, 1995) for the exact computation of the Hessian for multi-layer neural networks. See (Becker and LeCun, 1989; LeCun, Denker and Solla, 1990) for a diagonal approximation which can be computed and inverted in $O(s)$ operations. Another kind of approximation is the Gauss-Newton approximation also called the *outer product* approximation (Bishop, 1995; Hassibi and Stork, 1993), and applied when the cost function is the squared error:

$$\begin{aligned} \frac{\partial^2 C}{\partial \theta^2} &= \text{term due to penalty} + \sum_t \frac{\partial}{\partial \theta}((f_\theta(x_t) - y_t)^2 \frac{\partial f_\theta(x_t)}{\partial \theta}) \\ &\approx \text{term due to penalty} + \sum_t \frac{\partial f_\theta(x_t)}{\partial \theta}' \frac{\partial f_\theta(x_t)}{\partial \theta}. \end{aligned}$$

In that case, it is possible compute the inverse of the Hessian efficiently in time $O(Ts^2)$ where T is the number of training examples (Hassibi and Stork, 1993).

5 Conclusions

In this paper, we have presented a new methodology for simultaneously optimizing several hyper-parameters within gradient-based machine learning algorithms. It is based on the computation of the gradient of a model selection criterion with respect to the hyper-parameters, taking into account the influence of the hyper-parameters on the parameters. We have considered both the simpler case of a training criterion that is quadratic with respect to the parameters and the more general non-quadratic case. In both cases the Hessian of training cost must be computed and inverted (implicitly or explicitly). We have shown a particularly efficient procedure in the quadratic case that is based on back-propagating gradients through the Cholesky decomposition and back-substitutions. This was an improvement: we have arrived at this $O(s^3/3)$ procedure after studying first an $O(s^5)$ procedure and then an $O(2s^3)$ procedure for computing the gradients taking into account the inflence of λ on θ. In the particular case of input weight decays for linear regression, the computation can even be reduced to $O((s/m)^3/3)$ operations when there are m outputs.

We have performed preliminary experiments with the proposed methodology in several simple cases. First, the application to linear regression with weight decays for each input is described in (Bengio and Latendresse, 1999). The hyper-parameter optimization algorithm is used to perform a soft selection of the input variables. A large weight decay on one of the inputs effectively forces the corresponding weights to very small values. In contrast to feature selection algorithms such as the classical forward and backward selection methods used for linear regression, the algorithm for selecting hyper-parameters explores a continuous rather than

a discrete space. Comparisons are made in (Bengio and Latendresse, 1999) with ordinary regression as well as with stepwise regression methods and the adaptive ridge (Grandvalet, 1998) or LASSO (Tibshirani, 1995), suggesting that the proposed method gives better results when thre are many true non-zero regression coefficients and the correlation between the inputs is large.

Another type of application of the proposed method has been explored, in the context of non-stationary time-series prediction (Bengio and Dugas, 1999). In this case, an extension of the cross-validation criterion to sequential data which may be non-stationary is presented. Because of this non-stationarity, recent data may sometimes be more relevant to current predictions than older data. The squared error criterion is weighted by a parameterized function of the time indices (as the $w_i(\lambda)$ in eq. 1), and the parameters of that weighting function are the hyper-parameters discussed in this paper. Two hyper-parameters control from what point in the past the examples become more relevant to current predictions, and how much this point should be trusted. Using simply a constant model (i.e., the prediction is a weighted average of past desired outputs), we obtained statistically significant improvements in predicting one-month ahead future volatility of Canadian stocks. The comparisons were made against several linear, constant, and ARMA models of the volatility.

What remains to be done? first, on the practical side, we still have no experiments with the non-quadratic case (e.g., multi-layer neural networks), and no experiments with model selection criteria other than cross-validation. The cross-validation estimate of generalization error is unbiased but has a large variance (Breiman, 1996), and this might be hurtful to our procedure. Second, there are important theoretical questions that remain unanswered concerning the amount of overfitting that can be brought (or more precisely to the difference between the true generalization error and the value of the model selection criterion that is minimized) when too many hyper-parameters are optimized. As we have outlined in the introduction, the situation with hyper-parameters may be compared with the situation of parameters and the class of functions in which a solution is chosen. However, whereas the form of the training criterion as a sum of independent errors allows to define the capacity for a class of functions and relate it to the difference between generalization error and empirical error, it does not appear clearly to us how a similar analysis could be performed for hyper-parameters.

Acknowledgments

The author would like to thank Léon Bottou, Pascal Vincent, François Blanchette, and François Gingras, as well as the NSERC Canadian funding agency.

References

Akaike, H. (1974). A new look at the statistical model identification. *IEEE Transactions on Automatic Control*, AC-19(6):716–728.

Becker, S. and LeCun, Y. (1989). Improving the convergence of back-propagation learning with second order methods. In Touretzky, D., Hinton, G., and Sejnowski, T., editors,

Proceedings of the 1988 Connectionist Models Summer School, pages 29–37, Pittsburg 1988. Morgan Kaufmann, San Mateo.

Bengio, Y. and Dugas, C. (1999). Learning simple non-stationarities with hyper-parameters. Technical Report 1145, Département d'informatique et recherche opérationnelle, Université de Montréal.

Bengio, Y. and Latendresse, S. (1999). Soft variable selection with numerical optimization of weight decays. Technical report, Département d'informatique et recherche opérationnelle, Université de Montréal. in preparation.

Bishop, C. (1992). Exact calculation of the Hessian matrix for the multi-layer perceptron. *Neural Computation*, 4(4):494–501.

Bishop, C. (1995). *Neural Networks for Pattern Recognition*. Oxford University Press, London, UK.

Bottou, L. (1998). Online algorithms and stochastic approximations. In Saad, D., editor, *Online Learning in Neural Networks*. Cambridge University Press, to appear, Cambridge, UK.

Breiman, L. (1996). Heuristics of instability and stabilization in model selection. *Annals of Statistics*, 24 (6):2350–2383.

Craven, P. and Wahba, G. (1979). Smoothing noisy data with spline functions. *Numerical Mathematics*, 31:377–403.

Efron, B. and Tibshirani, R. J. (1993). *An introduction to the Bootstrap*. Chapman and Hall, New-York.

Grandvalet, Y. (1998). Least absolute shrinkage is equivalent to quadratic penalization. In Niklasson, L., Boden, M., and Ziemske, T., editors, *ICANN'98*, volume 1 of *Perspectives in Neural Computing*, pages 201–206. Springer.

Guyon, I., Vapnik, V., Boser, B., Bottou, L., and la, S. S. (1992). Structural risk minimization for character recognition. In Moody, J., Hanson, S., and Lipmann, R., editors, *Advances in Neural Information Processing Systems 4*, pages 471–479, San Mateo CA. Morgan Kaufmann.

Hassibi, B. and Stork, D. G. (1993). Second order derivatives for network pruning: Optimal brain surgeon. In Hanson, S. J., Cowan, J. D., and Giles, C. L., editors, *Advances in Neural Information Processing Systems 5*, pages 164–171, San Mateo, CA. Morgan Kaufmann.

Hinton, G. (1987). Learning translation invariant in massively parallel networks. In de Bakker, J., Nijman, A., and Treleaven, P., editors, *Proceedings of PARLE Conference on Parallel Architectures and Languages Europe*, pages 1–13, Berlin. Springer-Verlag.

Hoerl, A. and Kennard, R. (1970). Ridge regression: biased estimation for non-orthogonal problems. *Technometrics*, 12:55–67.

LeCun, Y., Denker, J., and Solla, S. (1990). Optimal brain damage. In Touretzky, D., editor, *Advances in Neural Information Processing Systems 2*, pages 598–605, Denver, CO. Morgan Kaufmann, San Mateo.

MacKay, D. and Neal, R. (1994). Automatic relevance determination. Unpublished report. See also MacKay D., 1995, Probable Neworks and Plausible Predictions – A Review of Practical Bayesian Methods for Supervised Neural Networks, in *Neutwork: Computation in Neural Systems*, v. 6, pp. 469–505.

Neal, R. (1998). Assessing relevance determination methods using delve. In Bishop, C., editor, *Neural Networks and Machine Learning*, pages 97–129. Springer-Verlag.

Poggio, T., Torre, V., and Koch, C. (1985). Computational vision and regularization theory. *Nature*, 317(26):314–319.

Rissanen, J. (1990). *Stochastic Complexity in Statistical Inquiry*. World Scientific, Singapore.

Smith, S. (1995). Differentiation of the cholesky algorithm. *Journal of Computational and Graphical Statistics*, 4:134–147.

Tibshirani, R. (1995). Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society B*, 58:267–288.

Tikhonov, A. and Arsenin, V. (1977). *Solutions of Ill-posed Problems*. W.H. Winston, Washington D.C.

Vapnik, V. (1982). *Estimation of Dependences Based on Empirical Data*. Springer-Verlag, Berlin.

Vapnik, V. (1995). *The Nature of Statistical Learning Theory*. Springer, New-York.