Input Decay: Simple and Effective Soft Variable Selection

Nicolas Chapados and Yoshua Bengio
Department of Computer Science and Operations Research
University of Montreal
C.P. 6128 Succ. Centre-Ville,
Montréal, Québec, Canada, H3C 3J7
{chapados,bengioy}@iro.umontreal.ca

Abstract

To deal with the overfitting problems that occur when there are not enough examples compared to the number of input variables in supervised learning, traditional approaches are weight decay and greedy variable selection. An alternative that has recently started to attract attention is to keep all the variables but to put more emphasis on the "most useful" ones. We introduce a new regularization method called input decay *that exerts more relative penalty on the parameters associated with the inputs that contribute less to the learned function. This method, like weight decay and variable selection, still requires to perform a kind of model selection. Successful comparative experiments with this new method were performed both on a simulated regression task and a real-world financial prediction task.*

1 Introduction

In a large number of applications of machine learning algorithms, we face an implementation dilemma: a great number of input features is often available to solve the problem, but the limited size of the training set makes it seemingly impossible to use them all without running the risk of severely overfitting the data. This dilemma provides the rationale justifying classical variable selection procedures, such as stepwise selection [1] or branch-and-bound. These methods attempt to select the "good variables", those yielding good generalization performance, to the exclusion of the others. We can alternatively define "good variables" to be those that are part of the generative model of the data; however, some variables that are part of the generative model may not, by themselves, be predictive enough to justify additional parameters in the model. In this paper, we argue that *penalized parameters* might allow to take such variables into account, albeit to a lesser extent. This enables to account for the fact that, in many situations, the distinction between "good variables" and "bad variables" is not nearly so clear cut. Some variables are certainly clearly useful; others are less so, *but they are not totally useless*.

Instead of reducing capacity by selecting a particular subset of variables, one can use regularization methods to reduce the capacity of the model. The most classical example is the weight decay [2] or "ridge" regression, which penalizes the squared norm of the parameter vector. However, weight decay penalizes all the input variables in the same way. More recently, several methods have been proposed to penalize input variables in different ways, depending on how "useful" they are. Examples of this class of algorithms are the adaptive ridge estimation procedure [3], the LASSO [4], and instances of hyper-parameter tuning such as those done in [5, 6] or in [7]. In this paper, we introduce a new approach to regularization for performing a "soft" selection of the variables, which we call *input decay*. It is well-suited to neural networks as well as classical linear regression, and is extremely easy to implement. Furthermore, contrarily to the combinatorial variable selection methods, it is computationally very cheap, requiring only a modest amount of effort over that normally required for a ordinary neural network training.

In section 2 we introduce notation and formalize and justify the proposed penalty. In section 3 we describe simulations in which we compare the proposed penalty method with more classical approaches, in a controlled setup where we can easily measure performance. In section 4, we present an application of the proposed method to a neural network regression problem occurring in financial decision-making.

2 Input Decay

Input decay is a method for performing "soft" variable selection during the regular training of a linear regression or non-linear neural network. The basic idea is that the training criterion penalizes the network connections coming from the inputs that have a less important role in determining the value of the output prediction.

Input decay works by adding a regularization term to the cost function used for training the network; the same principle can by applied to linear regression but we shall describe the general case of multi-layer perceptrons (MLPs). For a network trained to minimize the mean-squared error on a length-N training set $\{\langle x_i, y_i\rangle\}$, the cost function incorporating input decay is

$$C = \frac{1}{2N}\sum_{i=1}^{N}(f(x_i;\boldsymbol{\theta}) - y_i)^2 + C_{\mathsf{ID}}(\boldsymbol{\theta}), \qquad (1)$$

where $f(\cdot;\boldsymbol{\theta})$ is the function computed by the MLP and $C_{\mathsf{ID}}(\boldsymbol{\theta})$ is the input decay term. The fundamental idea behind input decay is to impose a penalty on the squared-norm of the weights linking a particular network input to all the hidden units. Let $\boldsymbol{\theta}^{(i)}$ be the parameters on the i-th MLP layer, and $\theta^{(1)}_{jh}$ the first-layer network weight linking input j to hidden unit h; the squared-norm of the weights from input j is:

$$C^{(j)}_{\mathsf{ID}}(\boldsymbol{\theta}) = \sum_{h=1}^{H}\left(\theta^{(1)}_{jh}\right)^2, \qquad (2)$$

where H is the number of hidden units in the network. The weights that are part of $C^{(j)}_{\mathsf{ID}}(\boldsymbol{\theta})$ are illustrated in figure 1. The complete contribution $C_{\mathsf{ID}}(\boldsymbol{\theta})$ to the cost function is obtained by a non-linear combination of the $C^{(j)}_{\mathsf{ID}}$:

$$C_{\mathsf{ID}}(\boldsymbol{\theta}) = \phi\sum_{j}\frac{C^{(j)}_{\mathsf{ID}}}{\eta + C^{(j)}_{\mathsf{ID}}(\boldsymbol{\theta})}, \qquad (3)$$

where the hyper-parameter ϕ governs the relative import
tant of the input decay term in the overall cost function. The behavior of the function $x^2/(\eta + x^2)$ is shown in figure 2. Intuitively, this function acts as follows: if the weights emanating from input j are small, the network must absorb a high marginal cost (locally quadratic) in order to increase the weights; the net effect, in this case, is to bring those weights closer to zero. On the other hand, if the weights associated with that input have become large enough, the penalty incurred by the network turns into a constant independent of the value of the weights; those are then free to be adjusted as appropriate. The hyper-parameter η acts as a threshold that determines the point beyond which the penalty becomes constant.

Input decay is similar to the *weight elimination* procedure [8] sometimes applied for training neural networks, with the difference that input decay applies in a collective way to the weights associated with a given input.

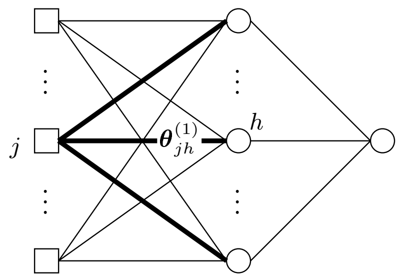

FIGURE 1. *Network weights affected by the input decay term $C^{(j)}_{\mathsf{ID}}(\boldsymbol{\theta})$, for an input j in a one-hidden-layer MLP (thick lines). This input decay penalty is lower when all the weights associated with a given input j go* ***together*** *to zero.*

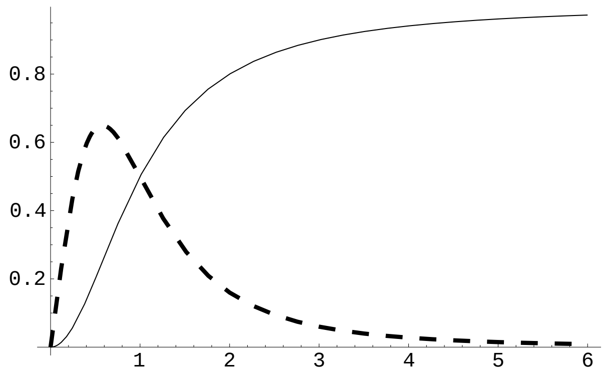

FIGURE 2. *The input decay penalization function $x^2/(\eta + x^2)$ (solid) and its first derivative (dashed), for $\eta = 1$. Here x represents $C^{(j)}_{\mathsf{ID}}(\boldsymbol{\theta})$, the sum of squared weights associated with input j.*

3 Experiments with Simulated Data

To ascertain the effectiveness of the input decay regularizer in principle, we performed experiments with generated data in a "difficult" linear regression setting. We compared the results obtained with the input decay method to stepwise (forward) variable selection and to the benchmark ordinary least-squares (OLS) regressor that uses all the variables. Experiments with MLPs on real data are described in section 4.

3.1 Experimental Setting

Data Generation We made use of the experimental framework described by Breiman [9], which consists in a linear regression problem. The generating model is

$$y^i = \boldsymbol{\beta}'\mathbf{x}^i + \epsilon^i, \qquad (4)$$

where $\epsilon^i \sim \mathcal{N}(0,1)$, and $\boldsymbol{\beta}$ and $\mathbf{x}^i$ are length-M vectors (the prime denotes the transpose operation). The number of variables is fixed to $M = 30$. We describe below how the coefficients vector $\boldsymbol{\beta}$ (fixed during an experiment) is chosen. The input vector $\mathbf{x}^i$ is drawn from a multivariate normal distribution of mean zero, and covariance matrix whose (i,j)-th element is $\rho^{|i-j|}$. Our experiments focused on the moderate-correlation ($\rho = 0.5$) and high-correlation ($\rho = 0.9$) situations.

In keeping with [9], we used three sets of $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_M)'$ coefficients. Within each set, we have $K = 3$ clusters of non-zero coefficients, centered at $c_1 = 5$, $c_2 = 15$, and $c_3 = 25$. Given the cluster centers $\{c_k\}$, the coefficients $\{\beta_m\}$ are computed as

$$\beta_m = C \sum_{k=1}^{K} [(h - |m - c_k|)^+]^2, \quad m = 1, \ldots, 30, \quad (5)$$

where $(x)^+ \stackrel{\text{def}}{=} \max(0, x)$; $h = 1, 3, 5$, respectively for the first, second and third set of coefficients; and C is a constant chosen such that the coefficient of determination R^2 of the regression on the generated data is ≈ 0.75.

Experiment Details For each of $\rho \in \{0.5, 0.9\}$ and $h \in \{1, 3, 5\}$, we generated 20 repetitions of 10000-element datasets. Within each dataset, the first 60 elements are used for training, the input values for the following 4940 are used in conjunction with the ADJ algorithm described below for selecting hyper-parameters, and the remaining 5000 are used for final testing.

We compare side-by-side the performance of the following models:

- *Classical* OLS *regression.* This involves no variable selection at all.
- *Input decay*, as described above, but for linear regression. Since we don't know a priori what are good values for the hyper-parameters ϕ and η, we train the models for every combination of $\phi \in \{0.001, 0.003, 0.01, 0.03, 0.1, 0.3, 1.0, 3.0, 10.0\}$ and $\eta \in \{0.001, 0.003, 0.01, 0.03, 0.1, 0.3, 1.0\}$, in addition to the "degenerate models" $\{\phi = 0, \eta = 100\}$ and $\{\phi = 100, \eta = 10^{-5}\}$ (large and small capacity models). We then select one model using the ADJ algorithm described below.
- *Forward variable selection*. We add variables one at a time (allowing up to 30 variables), and we choose the subset of variables giving the best estimated generalization error using 12-fold cross validation on the training set.

3.2 Review of the ADJ Model Selection Algorithm

The ADJ model selection algorithm, introduced by Shurmans [10], is based on the idea of exploiting the natural geometry of the distribution of input vectors to achieve a re-ranking of competing models on the basis of those that can be "trusted" the most. The algorithm only needs access to *unlabeled* examples drawn from the input distribution, in order to estimate this distribution.

The full justification and geometric intuition behind ADJ are given in [10]; we give here an operational description. First, ADJ needs a partial order $\prec$ on hypothesis classes, ordering the classes by their complexity. For input decay, given hypothesis classes H_1 and H_2, we define the order to be:

$$H_1 \prec H_2 \iff (\phi_1 \geq \phi_2 \,\wedge\, \eta_1 \leq \eta_2) \,\wedge\, (\phi_1 \neq \phi_2 \,\vee\, \eta_1 \neq \eta_2), \quad (6)$$

where ϕ_1 and η_1 are the hyper-parameters associated with H_1, and correspondingly for H_2. Intuitively, this order corresponds to the learning capacity allowed by the hyper-parameters. (We note that a larger input decay parameter ϕ *reduces* the capacity; in contrast, a smaller threshold η increases it.)

Next, the *expected distance* between two hypotheses h_1 and h_2 is defined as $d(h_1, h_2) = \left(\frac{1}{2}\int_X (h_1(x) - h_2(x))^2\, dP_X\right)^{1/2}$, where the integral is over the input distribution space; similarly, the *empirical distance* on the training set $\{\langle x_i, y_i\rangle\}$ of length N is $\hat{d}(h_1, h_2) = \left(\frac{1}{2N}\sum_i (h_1(x_i) - h_2(x_i))^2\right)^{1/2}$ (note that the computation of this empirical distance between two hypotheses does not use the targets y_i from the training set.)

Given a set of hypotheses $\{h_j^*\}$ (obtained, as usual, by minimizing the empirical error on the training set), each having obtained an empirical RMSE on the training set of $\widehat{\text{RMSE}}_j = \left(\frac{1}{2N}\sum_i (h_j^*(x_i) - y_i)^2\right)^{1/2}$, the ADJ algorithm re-ranks the $\widehat{\text{RMSE}}_j$ as follows:

1. It finds the largest observed ratio of expected distance to empirical distance for the "smaller" hypotheses in the partial order $\prec$:

$$r_j = \max_{k,\ H_k \prec H_j} d(h_k^*, h_j^*) \,/\, \hat{d}(h_k^*, h_j^*)\,. \quad (7)$$

2. It adjusts the empirical $\widehat{\text{RMSE}}_j$ by this ratio: $\widehat{\text{RMSE}}'_j = r_j\, \widehat{\text{RMSE}}_j$.

The model ultimately selected by the algorithm is the one having the smallest adjusted $\widehat{\text{RMSE}}'$.

The computation of expected distance $d(h_k^*, h_j^*)$ between two hypotheses requires a model of the input distribution P_X. This model can be estimated by having only access to unlabeled data drawn from the input distribution; either kernel estimators or Monte Carlo methods can be used for this purpose. In our experiments, we used 4940 (unlabeled) vectors (separate from either the training or the test set) drawn from the input distribution to form a Monte Carlo estimate of the expected distance.

3.3 Results

The results of the experiments are summarized in tables 1 and 2. The first table gives the mean-squared errors obtained by each method, averaged over the 20 generated training and test sets. Standard errors under a Student t_{19} distribution are also given. From this table, we note that input decay model selected by ADJ performs much better than either a standard OLS regression using all the variables or a regression after stepwise variable selection. This is true for both moderately ($\rho = 0.5$) and highly correlated ($\rho = 0.9$) input variables, and for all coefficient vectors ($h = 1, 3, 5$).

Table 2 formally confirms these observations by tabulating the p-values obtained under the t_{19} distribution for the MSE differences between input decay and the other two methods. All p-values are highly statistically significant. In addition, the column '# significant' lists the number of times, out of 20 repetitions, that input decay was found to be significantly better than the other method, using paired t-tests on the test set results.

We conclude from these results that linear regression with input decay, given a reasonable model of the input distribution and a good model selection algorithm such as the ADJ algorithm, performs significantly better than either the benchmark OLS regression or stepwise variable selection.

4 Experiments with an Asset-Allocation Problem

We also applied input decay to a real-world asset-allocation problem. Our experiments consisted in allocating among the 14 sectors (sub-indices) of the Toronto Stock Exchange TSE 300 index. Our input variables consisted in technical indicators related to each asset, including moving averages (at several depths) of asset returns and estimated asset volatilities; a total of 75 input variables were used.

We used standard MLPs to make the asset allocation decisions. They were trained to make forecasts of future asset performance, those forecasts serving as input to a fixed decision system. In all cases, both ordinary weight decay and the input decay regularizer were incorporated into the cost functions used for training the MLPs. The complete experimental setup, including details on our investment framework that allocates assets according to a value-at-risk (VaR) constraint, is explained in [11, 12].

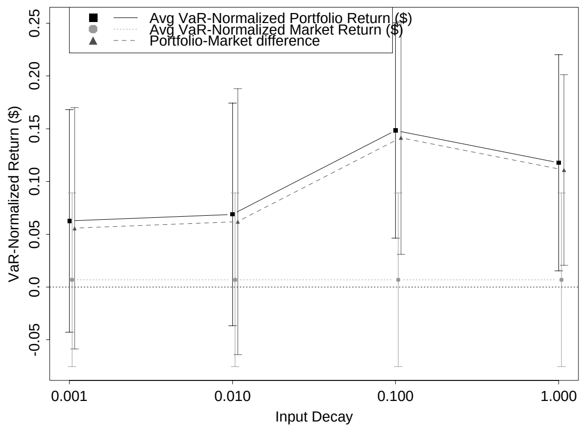

FIGURE 3. *Effect of Input Decay on the financial performance (average portfolio return normalized by the ex ante Value at Risk) obtained by an* MLP *in an asset-allocation task (solid). The (constant) benchmark market performance is given (dotted), along with the* MLP*–market difference (dashed). The error bars represent 95% confidence intervals.*

We compared the performance of various MLP topologies, varying the weight decay level, the input decay level (ϕ in eq. (3)) and the number of hidden units. In all cases, the input decay threshold η was kept fixed at 1.0. The performance criterion was a financial measure (the average return per period, normalized by the value-at-risk incurred) rather than a more conventional mean-squared error criterion.

Extensive statistical analysis of the results is presented elsewhere [11, 12]. We performed analyses of variance (ANOVA) to single out the effects of specific factors on the overall performance. In summary, we obtained the following results (we reserve the term "significant" to denote statistical significance at least at the 0.05 level):

- The effect of input decay is always significant (an example appears in figure 3).
- Weight decay is never significant.
- No higher-order interaction between the above factors is significant (as assessed by the ANOVA).

5 Conclusion

We introduced a new penalty-based method for soft variable selection that is very well-suited to multi-layer neural networks and classical linear regression settings. We

TABLE 1. *Comparative test* MSE *on the linear regression task described in Section 3.1, obtained by regression with input decay, standard* OLS *regression, and stepwise variable selection. In this experimental setting, ρ determines the correlation coefficients between the input variables, and $h = 1, 3, 5$ sets the number of non-zero coefficients in the data generating process (see eq. 5). The results are averaged over 20 different training and test sets. We observe that* ***the test* MSE *with Input Decay is always smaller.***

		Input Decay		OLS		Stepwise Selection	
ρ	h	Avg. MSE	Std Err.	Avg. MSE	Std Err.	Avg. MSE	Std Err.
0.5	1	0.866	(0.027)	1.046	(0.050)	1.085	(0.087)
0.5	3	0.813	(0.027)	1.046	(0.050)	1.354	(0.101)
0.5	5	0.740	(0.019)	1.046	(0.050)	1.104	(0.040)
0.9	1	0.730	(0.028)	1.045	(0.050)	0.848	(0.029)
0.9	3	0.693	(0.023)	1.045	(0.050)	0.810	(0.026)
0.9	5	0.700	(0.020)	1.045	(0.050)	0.840	(0.041)

TABLE 2. MSE *differences between regression with input decay and the other two methods, averaged over 20 generated datasets. The p-values result from t-tests over sequences of 20 differences. The '# Significant' columns list the number of experiments in which input decay was found significantly better than the competing method (computed with a paired t-test on the test sets for the 20 datasets). The improvement brought forth by Input Decay is nearly always statistically significant.*

		Input Decay vs. OLS			Input Decay vs. Stepwise Selection		
ρ	h	MSE diff.	p-value	# Significant	MSE diff.	p-value	# Significant
0.5	1	-0.180	$< \mathbf{0.001}\star$	19/20	-0.219	$\mathbf{0.030}\star$	12/20
0.5	3	-0.233	$< \mathbf{0.001}\star$	18/20	-0.541	$< \mathbf{0.001}\star$	15/20
0.5	5	-0.306	$< \mathbf{0.001}\star$	19/20	-0.364	$< \mathbf{0.001}\star$	19/20
0.9	1	-0.315	$< \mathbf{0.001}\star$	20/20	-0.117	$\mathbf{0.008}\star$	15/20
0.9	3	-0.352	$< \mathbf{0.001}\star$	20/20	-0.118	$\mathbf{0.001}\star$	16/20
0.9	5	-0.348	$< \mathbf{0.001}\star$	20/20	-0.143	$\mathbf{0.004}\star$	14/20

showed this method to be successful on difficult simulated regression tasks and a real-world financial application. Moreover, it is exceedingly easy to implement, and, compared with combinatorial variable selection, is quite cheap computationally.

References

[1] N.R. Draper and H. Smith, *Applied Regression Analysis*, John Wiley and Sons, 1981.

[2] G.E. Hinton, "Learning translation invariant in massively parallel networks," in *Proc. of PARLE Conference on Parallel Architectures and Languages Europe*, J.W. de Bakker et al., Ed., Berlin, 1987, pp. 1–13, Springer.

[3] Y. Grandvalet, "Least absolute shrinkage is equivalent to quadratic penalization," in *ICANN'98*, L. Niklasson, M. Boden, and T. Ziemske, Eds. 1998, vol. 1 of *Perspectives in Neural Computing*, pp. 201–206, Springer.

[4] R.J. Tibshirani, "Regression shrinkage and selection via the lasso," *Journal of the Royal Statistical Society B*, vol. 58, pp. 267–288, 1995.

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

[6] R.M. Neal, "Assessing relevance determination methods using delve," in *Neural Networks and Machine Learning*, C.M. Bishop, Ed., pp. 97–129. Springer-Verlag, 1998.

[7] Y. Bengio, "Gradient-based optimization of hyper-parameters," *Neural Computation*, vol. 12, no. 8, pp. 1889–1900, 2000.

[8] A. S. Weigend, D. E. Rumelhart, and B. A. Huberman, "Backpropagation, weight-elimination and time series prediction," in *Connectionist Models: Proceedings of the 1990 Summer School*, D.S Touretzky, J. L. Elman, T. J. Sejnowski, and G. E. Hinton, Eds., San Mateo, CA, 1991, Morgan Kaufmann.

[9] Leo Breiman, "Bagging predictors," *Machine Learning*, vol. 24, no. 2, pp. 123–140, 1994.

[10] D. Shuurmans, "A new metric-basic approach to model selection," in *Proc. of the 14th National Conference on Artificial Intelligence*, Providence, RI, July 1997.

[11] N. Chapados, "Optimization criteria for learning algorithms in portfolio selection," M.S. thesis, Université de Montréal, Montréal, Canada, January 2000.

[12] N. Chapados and Y. Bengio, "Cost functions and model combination for var–based asset allocation using neural networks," *IEEE Transactions on Neural Networks*, 2001, To appear.