Regularization Using Jittered Training Data

Russell Reed, Seho Oh, Robert J. Marks II
Dept. of Electrical Engineering
University of Washington, Seattle, WA, 98195

Abstract

In this paper we study the effect of generating additional training data by adding noise to the input data and show that it introduces convolutional smoothing of the target function. Training using such jittered data is shown, under a small variance assumption, to be equivalent to Lagrangian regularization with a derivative regularizer.

1 Introduction

Jittered data is data corrupted by additive noise. We consider the training of a layered perceptron with jittered input data. We show that use of a small amount of jitter convolutionally smooths the target and imposes a smoothing regularization on the error function.

Many studies have observed that training a multilayer perceptron with jittered input data often helps generalization (e.g., [12,10,7]). Holmström and Koistinen [4,6] study the problem of generating additional training data by adding noise to the data and show that the method is consistent, i.e., that under appropriate conditions, the resulting error function approaches the true error function as the number of training samples increases and the degree of jitter decreases. They also suggest ways of choosing the noise distribution and demonstrate smoothing of the computed function. Oh, Marks, and El-Sharkawi [8] have studied the effect of data jitter applied to query based learning in a multilayer perceptron.

In this paper we investigate the training of a layered perceptron with jittered data. The effect of jitter is a convolutional smoothing of the target function by the jitter's probability density [5]. With jitter, the resulting equivalent target function is defined over continuous ranges of the input plane while the original target function is defined only at the specific training points. The similarity of the effect of jitter and Lagrangian regularization is shown. The variance of the jitter, in certain cases, plays a role that is identical to the Lagrangian multiplier. Training with jitter thus allows regularization within the conventional layered perceptron architecture.

2 Convolution Property

A convolutional smoothing of the target function occurs in the presence of input jitter. Consider the problem of learning the mapping $(x, t(x))$ for all x. Assume x is a vector and t is a scalar. When we jitter the data, the network sees the input $\tilde{x} = x + n$ and the target value is $t(x) = t(\tilde{x} - n)$ where n is a random variable with probability density $p_n(n)$. Using the MSE error function, the optimum output for the network, given the input $\tilde{x}$, is

$$\begin{aligned} y^*(\tilde{x}) &= \langle t(\tilde{x} - n) \rangle && (1) \\ &= \int_n t(\tilde{x} - n) p_n(n)\, dn && (2) \\ &= t(\tilde{x}) * p_n(\tilde{x}) && (3) \end{aligned}$$

where '$*$' denotes convolution and $\langle \cdot \rangle$ denotes expectation. Thus, the optimum output for the network, given the input it sees, is the convolution of the actual target with the noise function. When the noise distribution has appropriate properties, this will smooth the target function and remove small features. This convolutional property and other effects of noisy sampling are studied in more detail in [5].

3 Sampled Data: Effect on the Target Function

The previous section assumes that training data for *all* x are available. When only limited training data are available, the target function is a sampled version of the underlying function. In this case, however, the distribution of $\tilde{x}_i = x_i + n$ is not uniform and the optimum output function is modified.

During training, we randomly select one of the training pairs, add noise to the input, and apply it to the network. Given a set of training points $\{(x_i, t_i) \mid i = 1 \dots M\}$, for any point $\tilde{x}$, let $P(i \mid \tilde{x})$ denote the probability $P[\tilde{x} = x_i + n \mid i \in 1 \dots M]$, i.e., the probability that $\tilde{x}$ is a noise corrupted version of the ith point given that it must be a version of one of the M points. In terms of the noise distribution $p_n(n)$, this is

$$P(i \mid \tilde{x}) = \frac{p_n(\tilde{x} - x_i)}{\sum_k p_n(\tilde{x} - x_k)} \tag{4}$$

and the expected value of the target, given $\tilde{x}$, is

$$\begin{aligned} \langle t(\tilde{x} - n) \mid \tilde{x} \rangle &= \sum_i t_i P(i \mid \tilde{x}) && (5) \\ &= \sum_i \frac{t_i p_n(\tilde{x} - x_i)}{\sum_k p_n(\tilde{x} - x_k)}. && (6) \end{aligned}$$

The denominator in (6) is $p(\tilde{x})$, the probability that input $\tilde{x}$ will be selected for training. In the case described in section 2, $p(\tilde{x}) = 1$ for all $\tilde{x}$ and (6) reduces to a convolution.

The expected value of the error at $\tilde{x}$ is

$$\tilde{\mathcal{E}} = \sum_{i=1}^{M} (t_i - y(\tilde{x}))^2 P(i \mid \tilde{x}). \tag{7}$$

Abbreviate $P(i \mid \tilde{x})$ with P_i and $y(\tilde{x})$ with y. After expanding the square, the value of $y(\tilde{x})$ computed by the network is independent of i. Thus

$$\begin{aligned} \tilde{\mathcal{E}} &= (\Sigma t_i^2 P_i) - 2y(\Sigma t_i P_i) + y^2 \Sigma P_i \\ &= (\Sigma t_i^2 P_i) - (\Sigma t_i P_i)^2 + \left[(\Sigma t_i P_i)^2 - 2y(\Sigma t_i P_i) + y^2 \Sigma P_i\right] \\ &= (\Sigma t_i^2 P_i) - (\Sigma t_i P_i)^2 + \left[(\Sigma_i t_i P_i) - y\right]^2, \end{aligned}$$

and

$$\frac{\partial \tilde{\mathcal{E}}}{\partial w} = -2\left[(\Sigma_i t_i P_i) - y\right] \frac{\partial y}{\partial w}. \tag{8}$$

In other words, under gradient descent, the system acts as if the target function is $\Sigma_i t_i P_i$, the expected value of the target in (6) given the conditions stated for P_i. Likewise, the effective error function is

$$E = \left[(\Sigma_i t_i P_i) - y\right]^2. \tag{9}$$

In contrast to conventional training with discrete data, the effective target function for jittered data is defined for all x and not just at the given training points. The following example illustrates the point.

Example

Fig. 1(a) is the Voronoi map of a set of points in two dimensions, the basis for a nearest neighbor classifier. Fig. 1(b) is expression (6) and Fig. 1(c) is convolution of the sampled target function with a Gaussian function. It can be seen that the convolution property smooths the decision surface and removes small features.

Assume the output neuron is thresholded at zero for a ± 1 decision. The optimum decision boundary is then the surface where the probabilities that $\tilde{x}$ is in the class $t_i = 1$ or the class $t_i = -1$ are equal. When t_i can only take values of ± 1, at the boundary where $\langle t(\tilde{x} - n) \mid \tilde{x} \rangle = 0$ the normalizing factor (denominator) has no effect and (6) coincides with the simple convolution.

4 Regularization Effect of Jitter on the Error Function

Even when the training data is noiseless, it is often sparse and the network is underconstrained. When the number of examples is less than the number of degrees of freedom of the network, some assumptions must be made about the function in order to choose an approximation. Regularization methods assume that the function is smooth —that small changes in the input don't cause large changes in the output. Poggio and Girosi [9], for example, suggest the cost function

$$\sum_i (y_i - t_i)^2 + \lambda \|Py\|^2 \tag{10}$$

where 'P is usually a differential operator' and λ balances the trade–off between smoothing and minimizing the error.

Jittering the inputs while keeping the target fixed embodies this smoothness assumption and results in a similar cost function. That is, we add small amounts of noise to the input data, assume that the target function doesn't change much, and minimize

$$\begin{aligned} \mathcal{E} &= \{\langle [t(x) - y(x+n)]^2 \rangle\} && (11) \\ &= \{t(x)^2 - 2t(x)\langle y(x+n)\rangle + \langle y(x+n)^2\rangle\} && (12) \end{aligned}$$

where $\{u\}$ indicates the expected value of u over the training patterns and $\langle u\rangle$ indicates the expected value of u over the noise n.

For small n, the network output can be approximated by

$$y(x+n) \approx y(x) + \left(\frac{\partial y}{\partial x}\right)^T n. \tag{13}$$

Substitution into (12) and dropping the independent variable for brevity gives

$$\mathcal{E} \approx \left\{ t^2 - 2ty - 2t\left(\frac{\partial y}{\partial x}\right)^T \langle n\rangle + y^2 + 2y\left(\frac{\partial y}{\partial x}\right)^T \langle n\rangle + \left(\frac{\partial y}{\partial x}\right)^T \langle n\, n^T\rangle \left(\frac{\partial y}{\partial x}\right)\right\}. \tag{14}$$

For zero–mean uncorrelated noise with equal variances, $\langle n\rangle = 0$ and $\langle n\, n^T\rangle = \sigma^2 I$ so

$$\begin{aligned} \mathcal{E} &\approx \left\{ t^2 - 2ty + y^2 + \sigma^2 \left(\frac{\partial y}{\partial x}\right)^T \left(\frac{\partial y}{\partial x}\right)\right\} && (15) \\ &\approx \{(t-y)^2\} + \sigma^2 \left\{ \left\| \frac{\partial y}{\partial x} \right\|^2 \right\}. && (16) \end{aligned}$$

In (16), the term $\{(t-y)^2\}$ is the normal back–propagation error function and the term $\left\{ \left\| \frac{\partial y}{\partial x} \right\|^2 \right\}$ is the average squared magnitude of the gradient of the output function on the training points. Thus training with jitter introduces a term in the error function that encourages smooth solutions [11,1]. The system minimizes sensitivity to noisy inputs by driving the gradient of the transfer function to zero at the training points. (A method which explicitly calculates and back–propagates similar terms in a multilayer perceptron is described in [3].)

Comparison with (10) shows that σ^2 plays a role similar to λ in the regularization equation, balancing smoothness and error minimization. They differ in that training with jitter minimizes the gradient term at the training points whereas regularization seeks to minimize it for all x.

4.1 Effect on E: 2nd Order Approximation

Eq. (13) is a linear approximation for the network output. A second-order approximation is

$$y(x+n) \approx y(x) + \left(\frac{\partial y}{\partial x}\right)^T n + \frac{1}{2} n^T H n \tag{17}$$

where H is the Hessian matrix $\partial^2 y/(\partial x_i \partial x_j)$. Assuming an even noise distribution, so that $\langle n^k \rangle = 0$ for k odd, one can write

$$\begin{aligned} \mathcal{E} \approx \quad & \{(t-y)^2\} + \sigma^2 \left\{ \left\| \frac{\partial y}{\partial x} \right\|^2 \right\} \\ + \quad & \left\{ \sigma^2 (y-t) Tr(H) + \frac{\sigma^4}{4} Tr(H)^2 + \frac{\sigma^4}{2} Tr(H^2) + \frac{m_4 - 3\sigma^4}{4} (\Sigma_i h_{ii}^2) \right\} \end{aligned} \tag{18}$$

where m_4 is the fourth moment $\langle n^4 \rangle$ and h_{ii} are the diagonal components of the Hessian. Since H is symmetric, $Tr(H^2)$ may also be written as $||H||^2$. When H is assumed to be zero, (18) reduces to (16). For Gaussian noise, $m_4 = 3\sigma^2$, and (18) can be written

$$\mathcal{E} \approx \{(\hat{y}-t)^2\} + \sigma^2 \left\{ \left\| \frac{\partial y}{\partial x} \right\|^2 \right\} + \frac{\sigma^4}{2} \{Tr(H^2)\} \tag{19}$$

where the equivalent output

$$\hat{y} = y + \frac{\sigma^2}{2} Tr(H) \tag{20}$$

is biased proportional to $Tr(H) = \nabla^2 y$. The Laplacian, $\nabla^2 y$, can be interpretted roughly as a measure of the difference between the average surrounding values and the precise value of the field at a point [2]. $\hat{y}$ can thus be thought of as the average value of y in the some neighborhood around the training point. The second term in (19) is the first order regularization term in (16). The third term is proportional to $Tr(H^2) = ||H||^2$, the sum of the squared eigenvalues of H.

In contrast to training with non-jittered data, which simply minimizes the error at the training points and puts no constraints on the function at other points, training with jitter also requires the approximating function to have small derivatives and a local average that approaches the target in the vicinity of each training point.

5 Simulation: BP Training with Jittered Data

Fig. 2(a) shows the decision boundary formed by an intentionally overtrained 2/50/10/1 feed-forward network. With 671 weights, but only 31 training points, the network is very underconstrained and chooses a very nonlinear boundary.

Training with jittered data discourages sharp changes in the network response near the training points and so discourages the network from forming overly complex boundaries. Fig. 2(b) shows the same network trained for the same amount of time from the same initial conditions with Gaussian input noise ($\sigma = 0.1$). Despite very long training times, the network shows no effects of overtraining. For reference, Fig. 2(c) shows the expression in (6).

6 Discussion

Training with small amounts of jitter approaches the generalization problem directly by assuming that slightly different inputs give approximately the same output. If the noise distribution is smooth, the network will interpolate among training points in proportion to a smooth function of the distance to each training point.

A more well-founded justification for smoothing the target function can be based on sampling theory. When training data for all x is available, the effective target function is the convolution of the given target and the noise distribution. It is well known that the Fourier transform of the convolution of two functions is the product of their transforms. Thus, if the noise distribution has appropriate characteristics, the effective target will be a low-pass filtered version of the original target. (Another choice of distributions could, of course, result in band-pass or high-pass filtering and would not smooth the target.) If the target is represented by a small number of samples, one can expect the samples to capture the large features (low-frequency components) of the function, but not the small features (high-frequency components). By acting like a low-pass filter, training with jitter removes high-frequency elements from the target which can't be reliably represented by the small number of samples. If, for example, $p_n(x) = W\text{sinc}^2(Wx)$, where $W > 0$ and $\text{sinc}(x) = \sin(\pi x)/(\pi x)$, then $y^*(x)$ in (3) is bandlimited and therefore uniquely determined by its samples located on a rectangular grid of linear dimension $1/(2W)$ [5].

In the case of large networks trained on a small number of samples, jitter helps to prevent overfitting by providing additional constraints. The effective target is defined for all x so the network is no longer underconstrained and the extra degrees of freedom are used to approximate the smoothed target function rather than forming an arbitrarily complex boundary that just happens to fit the original training data. Even though the network may be large, it models a simpler system.

Training with jitter is a simple and effective means of regularization. Significantly, training can be performed within the multilayer sigmoid perceptron architecture in the spirit of classical back-propagation. This is not possible with most forms of Lagrangian regularization.

We have not addressed the problem of how to choose an appropriate noise distribution. Holmström and Koistinen [4,6] suggest some criteria for doing so. The similarity of training with jitter and regularization suggests that the regularization research may also be helpful.

References

[1] C. Bishop. Improving the generalization properties of radial basis function neural networks. *Neural Computation*, 3(4):579–588, 1991.

[2] H. F. Davis and A. D. Snider. *Introduction to Vector Analysis*. Allyn and Bacon, Inc., Boston, fourth edition, 1979.

[3] H. Drucker and Y. Le Cun. Double backpropagation and increasing generalization performance. In *Proceedings of the International Joint Conference on Neural Networks*, pages II–145, 1991. (Seattle).

[4] L. Holmström and P. Koistinen. Using additive noise in back-propagation training. *IEEE Transactions on Neural Networks*, 3(1):24–38, Jan. 1992.

[5] R. J. Marks II. *Introduction to Shannon Sampling and Interpolation Theory*. Springer-Verlag, New York, 1991.

[6] P. Koistinen and L. Holmström. Kernel regression and backpropagation training with noise. In *Proceedings of the International Joint Conference on Neural Networks*, pages 367–372, 1991. (Singapore).

[7] A. Linden and J. Kindermann. Inversion of multilayer nets. In *Proceedings of the International Joint Conference on Neural Networks*, pages II–425, 1989.

[8] S. Oh, R. J. Marks II, and M. A. El-Sharkawi. Query based learning in a multilayered perceptron in the presence of data jitter. In M. A. El-Sharkawi and R. J. Marks II, editors, *Applications of Neural Networks to Power Systems*, pages 72–75, 1991. (Seattle, WA).

[9] T. Poggio and F. Girosi. Networks for approximation and learning. *Proceedings of the IEEE*, 78(9):1481–1497, Sept. 1990.

[10] J. Sietsma and R. J. F. Dow. Creating artificial neural networks that generalize. *Neural Networks*, 4(1):67–69, 1991.

[11] A. R. Webb. Functional approximation by feed-forward networks: A least-squares approach to generalization. R.S.R.E. Memorandum 4453, 1991.

[12] A. S. Weigend, D. E. Rumelhart, and B. A. Huberman. Generalization by weight-elimination applied to currency exchange rate prediction. In *Proceedings of the International Joint Conference on Neural Networks*, pages I-837, 1991.

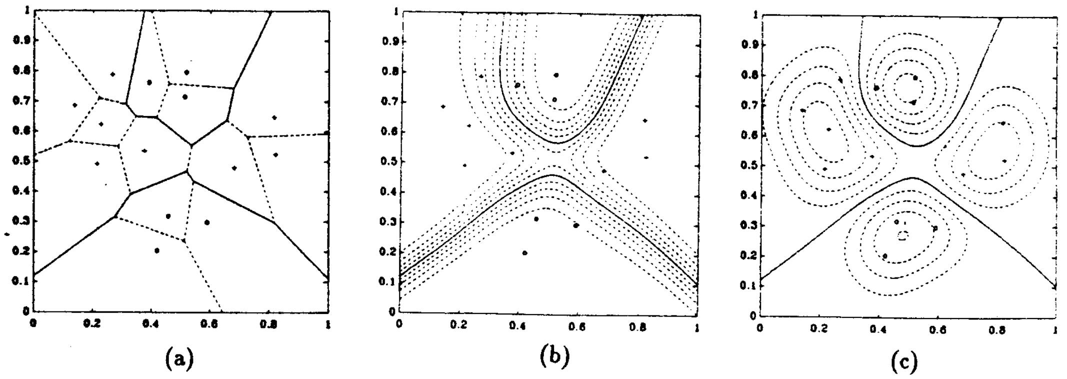

Figure 1: A 14-point nearest neighbor classification problem. (a) The Voronoi map for the 14 points. (b) The expected value for the classification of x calculated in (6) for a Gaussian noise distribution ($\sigma = 0.1$). (c) The convolution of the training set with the same Gaussian noise. The zero contours of (b) and (c) coincide.

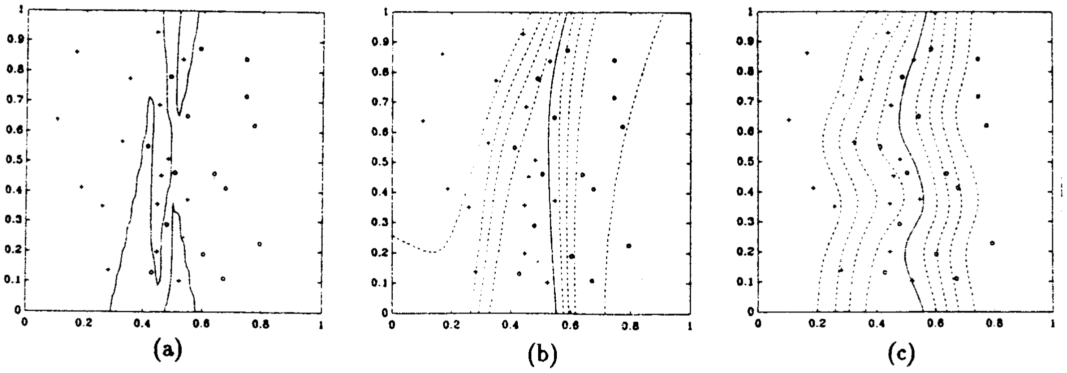

Figure 2: (a) An intentionally overtrained 2/50/10/1 feed-forward network chooses an overly complex boundary and can be expected to generalize poorly. (b) The same network trained with Gaussian ($\sigma = 0.1$) input noise forms a much smoother boundary and better generalization can be expected. (c) The expression in Eq. (6) for the expected value of the target function at an arbitrary point x.