DESIGN OF ROBUST NEURAL NETWORK CLASSIFIERS

Jan Larsen, Lars Nonboe Andersen, Mads Hintz-Madsen and Lars Kai Hansen

CONNECT, Department of Mathematical Modelling, Building 321
Technical University of Denmark, DK-2800 Lyngby, Denmark
emails: jl,lna,mhm,lkhansen@imm.dtu.dk
www: http://eivind.imm.dtu.dk

ABSTRACT

This paper addresses a new framework for designing robust neural network classifiers. The network is optimized using the maximum a posteriori technique, i.e., the cost function is the sum of the log-likelihood and a regularization term (prior). In order to perform robust classification, we present a modified likelihood function which incorporate the potential risk of outliers in the data. This leads to introduction of a new parameter, the outlier probability. Designing the neural classifier involves optimization of network weights as well as outlier probability and regularization parameters. We suggest to adapt the outlier probability and regularization parameters by minimizing the error on a validation set, and a simple gradient descent scheme is derived. In addition, the framework allows for constructing a simple outlier detector. Experiments with artificial data demonstrates the potential of the suggested framework.

1. INTRODUCTION

Neural networks are flexible tools for pattern recognition due to the universal approximation theorems [6]. We consider a neural classifier architecture based on a feed-forward net with a modified SoftMax [3] normalization as presented in [1] (see also, [2], [7]). The outputs of the network estimate the class conditional posterior probabilities and the network is trained using a maximum a posteriori (MAP) framework. Robustness is incorporated via a probabilistic definition of outliers. Thus a given example is considered as an outlier if its class label is changed with a certain probability ε; the *outlier probability*.

The associated risk of overfitting on noisy data is of major concern in neural network design [5]. The objective of network design is to obtain a reliable and minimal generalization error which can be done by constraining the model flexibility and adapting the outlier probability. Model constraints are imposed directly via pruning techniques (see e.g., [1], [9], [10]) or indirectly using regularization. We will merely consider regularization in this presentation.

Based on earlier work [1], [9], [10], we will present an iterative scheme for simultaneously adapting the amount of regularization and outlier probability by minimizing the validation error calculated from a single validation set. Here we take the validation error as an estimate of the true generalization error.

This research was supported by the Danish Natural Science and Technical Research Councils through the Computational Neural Network Center (CONNECT). JL furthermore acknowledge the Radio Parts Foundation for financial support.

2. NETWORK ARCHITECTURE

Suppose that the input (feature) vector is denoted by $\boldsymbol{x}$ with $\dim(\boldsymbol{x}) = n_I$ and $\mathcal{C}_i$ denotes the i'th of the mutually exclusive classes, $i = 1, 2, \cdots, c$. The aim is to model the posterior probabilities of the class given the input. Aiming at robustness against an outlier[1], defined as a class label which erroneously is changed to one of the other classes, we introduce the probability of an outlier, $0 \le \varepsilon \le 1$. It is assumed that the occurrence of an outlier is independent of class label and input location. Thus the outlier process acts as an extra noise source independent of input location, as opposed to the error due to overlap in class posterior probabilities. This leads to the definition of posterior probabilities, $p(\mathcal{C}_i|\boldsymbol{x})$, $1 \le i \le c$,

$$p(\mathcal{C}_i|\boldsymbol{x}) = p_0(\mathcal{C}_i|\boldsymbol{x}) \cdot (1-\varepsilon) + \frac{\varepsilon}{c-1} \sum_{j=1, j\neq i}^{c} p_0(\mathcal{C}_j|\boldsymbol{x}). \quad (1)$$

where $p_0(\mathcal{C}_i|\boldsymbol{x})$ is the posterior probability in the case of zero outlier probability. The first term is the posterior probability for $\mathcal{C}_i$ times the probability that an outlier does not occur while the second term is the sum of posterior probabilities for $\mathcal{C}_j \neq \mathcal{C}_i$ times the scaled outlier probability $\beta \equiv \varepsilon/(c-1) \in [0; 1/(c-1)]$ that $\mathcal{C}_i$ has changed specifically to $\mathcal{C}_j$. Under a simple loss function the Bayes optimal[2] classifier assigns class label $\mathcal{C}_i$ to $\boldsymbol{x}$ if $i = \arg\max_j p(\mathcal{C}_j|\boldsymbol{x})$. Note that Eq. (1) can be rewritten as

$$p(\mathcal{C}_i|\boldsymbol{x}) = p_0(\mathcal{C}_i|\boldsymbol{x})(1-\beta c) + \beta. \quad (2)$$

Since $0 \le p_0(\mathcal{C}_i|\boldsymbol{x}) \le 1$ and $\sum_{i=1}^{c} p_0(\mathcal{C}_i|\boldsymbol{x}) = 1$ due to mutual exclusive classes, then $\beta \le p(\mathcal{C}_i|\boldsymbol{x}) \le 1-\varepsilon$ and further $\sum_{i=1}^{c} p(\mathcal{C}_i|\boldsymbol{x}) = 1$.

Define $\widehat{y}_i$ as *estimates* of the posterior probabilities given by $\widehat{y}_i = \widehat{z}_i(1-\beta c) + \beta$ where $\widehat{z}_i$ are estimates of the $\varepsilon = 0$ posterior probabilities, $p_0(\mathcal{C}_i|\boldsymbol{x})$. Following [1], [7] (see also [2]), $\widehat{z}_i$, are taken as outputs of a neural network. Since the posterior probabilities sums to 1, also $\sum_{i=1}^{c} \widehat{z}_i = 1$, i.e., we merely estimate $c-1$ posterior probabilities, say $\widehat{z}_i$, $1 \le i \le c-1$, and calculate the last as $\widehat{z}_c = 1 - \sum_{i=1}^{c-1} \widehat{z}_i$.

Define a 2-layer feed-forward network with n_I inputs, n_H hidden neurons and $c-1$ outputs by:

$$h_j(\boldsymbol{x}) \quad = \quad \tanh\left(\sum_{\ell=1}^{n_I} w_{j\ell}^I x_\ell + w_{j0}^I\right), \quad (3)$$

[1]See [8] for various approaches on robust statistics.

[2]That is, each misclassification is equally weighted corresponding to minimal probability of misclassification.

$$\phi_i(\boldsymbol{x}) \quad = \quad \sum_{j=1}^{n_H} w_{ij}^H h_j(\boldsymbol{x}) + w_{i0}^H \tag{4}$$

where $w_{j\ell}^I$, w_{ij}^H are the input-to-hidden and hidden-to-output weights, respectively. All weights are assembled in the weight vector $\boldsymbol{w} = \{w_{j\ell}^I, w_{ij}^H\}$. In order to interpret the network outputs as probabilities we use a *modified* normalized exponential transformation [1] similar to SoftMax [3],

$$\widehat{z}_i = \frac{\exp(\phi_i)}{\sum_{j=1}^{c-1}\exp(\phi_j)+1}, \; 1 \le i \le c-1, \tag{5}$$

and $\widehat{z}_c = 1 - \sum_{i=1}^{c-1}\widehat{z}_i$.

3. TRAINING AND REGULARIZATION

Assume that we have a training set $\mathcal{T}$ of N_t related input-output pairs $\mathcal{T} = \{(\boldsymbol{x}(k), \boldsymbol{y}(k))\}_{k=1}^{N_t}$ where

$$y_i(k) = \begin{cases} 1 & \text{if } \boldsymbol{x}(k) \in \mathcal{C}_i \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

The likelihood of the network parameters is given by (see e.g., [2], [7]),

$$p(\mathcal{T}|\boldsymbol{w}) = \prod_{k=1}^{N_t} p(\boldsymbol{y}(k)|\boldsymbol{x}(k),\boldsymbol{w}) = \prod_{k=1}^{N_t}\prod_{i=1}^{c}\left(\widehat{y}_i(k)\right)^{y_i(k)} \tag{7}$$

where $\widehat{\boldsymbol{y}}(k) = \widehat{\boldsymbol{y}}(\boldsymbol{x}(k),\boldsymbol{w})$ is a function of the input and weight vectors. The training error is the normalized negative log-likelihood

$$S_{\mathcal{T}}(\boldsymbol{w}) = -\frac{1}{N_t}\log p(\mathcal{T}|\boldsymbol{w}) \equiv \frac{1}{N_t}\sum_{k=1}^{N_t}\ell\left(\boldsymbol{y}(k),\widehat{\boldsymbol{y}}(k);\boldsymbol{w}\right) \tag{8}$$

with $\ell(\cdot)$ denoting the loss given by

$$\ell\left(\boldsymbol{y}(k),\widehat{\boldsymbol{y}}(k);\boldsymbol{w}\right) = \sum_{i=1}^{c} -y_i(k)\log(\widehat{y}_i(k)) \tag{9}$$

Making an comparison with M-estimates considered in robust statistics [8], we note that the loss for a specific example is $\ell = -\log(\widehat{y}_i) = \psi(\ell_0)$ where $\ell_0 = -\log(\widehat{z}_i)$ is the non-robust loss ($\varepsilon = 0$) and $\psi(\cdot)$ is a function which downweights extreme losses[3].

The objective of training is minimization of the regularized cost function[4]

$$C(\boldsymbol{w}) = S_{\mathcal{T}}(\boldsymbol{w}) + R(\boldsymbol{w},\boldsymbol{\kappa}) \tag{10}$$

where the regularization term $R(\boldsymbol{w},\boldsymbol{\kappa})$ is parameterized by a set of regularization parameters $\boldsymbol{\kappa}$. Training provides the estimated weight vector $\widehat{\boldsymbol{w}} = \arg\min_{\boldsymbol{w}} C(\boldsymbol{w})$ and is done using a Gauss-Newton scheme (see e.g., [11]),

$$\boldsymbol{w}^{\text{new}} = \boldsymbol{w}^{\text{old}} - \eta\cdot\boldsymbol{J}^{-1}(\boldsymbol{w}^{\text{old}})\boldsymbol{\nabla}(\boldsymbol{w}^{\text{old}}) \tag{11}$$

where η is the step-size (line search parameter). For that purpose we require the gradient, $\boldsymbol{\nabla}(\boldsymbol{w}) = \partial C/\partial\boldsymbol{w}$, and the Gauss-Newton approximation[5], $\boldsymbol{J}(\boldsymbol{w})$, of the Hessian $\partial^2 C/\boldsymbol{w}\boldsymbol{w}^\top$ which can be written as

$$\frac{\partial C}{\partial\boldsymbol{w}}(\boldsymbol{w}) = \frac{\beta c - 1}{N_t}\sum_{k=1}^{N_t}\sum_{i=1}^{c}\frac{\partial\widehat{z}_i}{\partial\boldsymbol{w}}\frac{y_i(k)}{\widehat{y}_i} + \frac{\partial R(\boldsymbol{w},\boldsymbol{\kappa})}{\partial\boldsymbol{w}}, \tag{12}$$

$$\boldsymbol{J}(\boldsymbol{w}) = \frac{(1-\beta c)^2}{N_t}\sum_{k=1}^{N_t}\sum_{i=1}^{c}\frac{1}{\widehat{y}_i}\frac{\partial\widehat{z}_i}{\partial\boldsymbol{w}}\frac{\partial\widehat{z}_i}{\partial\boldsymbol{w}^\top} + \frac{\partial^2 R(\boldsymbol{w},\boldsymbol{\kappa})}{\partial\boldsymbol{w}\partial\boldsymbol{w}^\top}. \tag{13}$$

where

$$\frac{\partial\widehat{z}_i}{\partial\boldsymbol{w}} \quad = \quad \widehat{z}_i\sum_{j=1}^{c-1}(\delta_{ij}-\widehat{z}_j)\frac{\partial\phi_j}{\partial\boldsymbol{w}}, \; 1\le i\le c-1, \tag{14}$$

$$\frac{\partial\widehat{z}_c}{\partial\boldsymbol{w}} \quad = \quad -\widehat{z}_c\sum_{j=1}^{c-1}\widehat{z}_j\frac{\partial\phi_j}{\partial\boldsymbol{w}}. \tag{15}$$

By convenience, the dependency of $\widehat{z}_i$, $\widehat{y}_i$ and ϕ_i on $\boldsymbol{x}(k)$ and $\boldsymbol{w}$ is omitted, and δ_{ij} denotes the Kronecker delta.

4. ADAPTING REGULARIZATION PARAMETERS AND OUTLIER PROBABILITY

The available data set, $\mathcal{D}$, of N examples is split into two disjoint sets: a validation set, $\mathcal{V}$, with $N_v = \lceil\gamma N\rceil$ examples for estimation of regularization and outlier probability, and a training set, $\mathcal{T}$, with $N_t = N - N_v$ examples for estimation of network parameters. γ is referred to as the split-ratio. The validation error of the trained network is given by

$$S_{\mathcal{V}}(\widehat{\boldsymbol{w}}) = \frac{1}{N_v}\sum_{k=1}^{N_v}\ell\left(\boldsymbol{y}(k),\widehat{\boldsymbol{y}}(k);\widehat{\boldsymbol{w}}\right) \tag{16}$$

where the sum runs over the N_v validation examples. $S_{\mathcal{V}}(\widehat{\boldsymbol{w}})$ is thus an estimate of the generalization error defined as the expected loss: $G(\widehat{\boldsymbol{w}}) = E_{\boldsymbol{x},\boldsymbol{y}}\{\ell(\boldsymbol{y},\widehat{\boldsymbol{y}};\widehat{\boldsymbol{w}})\}$, where $E_{\boldsymbol{x},\boldsymbol{y}}\{\cdot\}$ denotes the expectation w.r.t. the joint input-output distribution.

Let $\boldsymbol{\theta} = [\boldsymbol{\kappa},\beta]$ be the vector of all regularization parameters and the scaled outlier probability. Aiming at adapting $\boldsymbol{\theta}$ so as to minimize the validation error we apply the iterative scheme suggested in [1], [9], [10]:

$$\boldsymbol{\theta}^{\text{new}} = \boldsymbol{\theta}^{\text{old}} - \mu\frac{\partial S_{\mathcal{V}}}{\partial\boldsymbol{\theta}}(\widehat{\boldsymbol{w}}(\boldsymbol{\theta}^{\text{old}})) \tag{17}$$

where μ is a step-size and $\widehat{\boldsymbol{w}}(\boldsymbol{\theta}^{\text{old}})$ is the estimated weight vector using $\boldsymbol{\theta}^{\text{old}}$. Suppose that

$$R(\boldsymbol{w},\boldsymbol{\kappa}) = \boldsymbol{\kappa}^\top\boldsymbol{r}(\boldsymbol{w}) = \sum_{i=1}^{q}\kappa_i r_i(\boldsymbol{w}) \tag{18}$$

where κ_i are the regularization parameters and $r_i(\boldsymbol{w})$ the associated regularization functions. The gradient of the validation error then equals [9], [10]:

$$\frac{\partial S_{\mathcal{V}}}{\partial\boldsymbol{\kappa}}(\widehat{\boldsymbol{w}}) = -\frac{\partial\boldsymbol{r}}{\partial\boldsymbol{w}^\top}(\widehat{\boldsymbol{w}})\cdot\boldsymbol{J}^{-1}(\widehat{\boldsymbol{w}})\cdot\frac{\partial S_{\mathcal{V}}}{\partial\boldsymbol{w}}(\widehat{\boldsymbol{w}}), \tag{19}$$

[3] $\psi(\ell) = -\log(e^{-\ell}(1-\beta c)+\beta)$.

[4] This might be viewed as a maximum a posteriori (MAP) technique.

[5] This is obtained using Fisher's property: $E[\partial^2 L/\partial\boldsymbol{w}\partial\boldsymbol{w}^\top] = E[\partial L/\partial\boldsymbol{w}\,\partial L/\partial\boldsymbol{w}^\top]$ where $L = -\log p(\mathcal{T}|\boldsymbol{w})$.

$$\frac{\partial S_{\mathcal{V}}}{\partial \beta}(\widehat{\boldsymbol{w}}) = -\frac{1}{N_v}\sum_{k=1}^{N_v}\sum_{i=1}^{c}\frac{1-c\widehat{z}_i(\widehat{\boldsymbol{w}})}{\widehat{y}_i(\widehat{\boldsymbol{w}})}y_i(k)$$
$$-\frac{\partial S_{\mathcal{V}}}{\partial \boldsymbol{w}^\top}(\widehat{\boldsymbol{w}})\cdot \boldsymbol{J}^{-1}(\widehat{\boldsymbol{w}})\cdot\frac{1}{N_t}\sum_{k=1}^{N_t}\sum_{i=1}^{c}\frac{\partial \widehat{z}_i}{\partial \boldsymbol{w}}(\widehat{\boldsymbol{w}})\frac{y_i(k)}{\widehat{y}_i^2(\widehat{\boldsymbol{w}})}. \quad (20)$$

In order to ensure that $\kappa_i \geq 0$ and $0 \leq \beta \leq 1/(c-1)$ we perform a reparameterization,

$$\kappa_i = \begin{cases} \exp(\lambda_i), \lambda_i < 0 \\ \lambda_i + 1 \ \ , \lambda_i \geq 0 \end{cases} \qquad \beta = \frac{1+\tanh(\gamma)}{2(c-1)}, \gamma \in \mathbb{R} \quad (21)$$

and carry out the minimization w.r.t. the new parameters $\boldsymbol{\lambda}$ and γ assembled in the vector $\boldsymbol{\xi} = [\boldsymbol{\lambda}, \gamma]$. Note that

$$\frac{\partial S_{\mathcal{V}}}{\partial \xi_i} = \frac{\partial S_{\mathcal{V}}}{\partial \theta_i}\cdot\frac{\partial \theta_i}{\partial \xi_i}. \quad (22)$$

In summary the algorithm for adapting regularization parameters and outlier probability is:

1. Select the split ratio γ and initialize $\boldsymbol{\kappa}$, β and the weights of the network.
2. Train the network with fixed $\boldsymbol{\xi}$ to achieve $\widehat{\boldsymbol{w}}(\boldsymbol{\xi})$. Calculate the validation error $S_{\mathcal{V}}$.
3. Calculate the gradient $\partial S_{\mathcal{V}}/\partial\boldsymbol{\xi}$ cf. Eq. (19), (20). Initialize the step-size μ.
4. Update $\boldsymbol{\xi}$ using Eq. (17), (21), train the network from the previous weights and calculate $S_{\mathcal{V}}$.
5. If $S_{\mathcal{V}}$ decreases repeat: double μ, update $\boldsymbol{\xi}$, retrain weights and recalculate $S_{\mathcal{V}}$ until no decrease is noticed then goto step 7.
6. Repeat: perform bisection of μ, update $\boldsymbol{\xi}$, retrain weights and recalculate $S_{\mathcal{V}}$ until a decrease is noticed, then continue.
7. Repeat steps 3–6 until the relative change in validation error is below a small percentage or, e.g., $\|\partial S_{\mathcal{V}}/\partial\boldsymbol{\kappa}\|$ is below a small number.

5. OUTLIER DETECTION

Once the network is designed, i.e., we have estimates of the weights, regularization parameters and outlier probability[6], it is possible to devise a method for outlier detection. Suppose we want to decide whether an example $\boldsymbol{x}$ with label $\mathcal{C}_i$ is an outlier or not. Define the binary variable O which is 1 if the example is an outlier, and zero otherwise. The probability that the example is an outlier is given as $p_{\text{outlier}} = p(O=1|\boldsymbol{x},\mathcal{C}_i)$. Using Bayes rule, $p_{\text{outlier}} = p(O=1|\boldsymbol{x},\mathcal{C}_i) = p(O=1 \wedge \mathcal{C}_i|\boldsymbol{x})/p(\mathcal{C}_i|\boldsymbol{x})$. The denominator is given by Eq. (2) and the numerator is the posterior probability for $\mathcal{C}_i$ in the case of outliers which is equal to the last addend of Eq. (1). Thus[7],

$$p_{\text{outlier}} = \frac{\beta(1-p_0(\mathcal{C}_i|\boldsymbol{x}))}{p_0(\mathcal{C}_i|\boldsymbol{x})(1-\beta c)+\beta}. \quad (23)$$

The estimated probability that the example is an outlier is consequently, $\widehat{p}_{\text{outlier}} = \widehat{\beta}(1-\widehat{z}_i)/\widehat{y}_i$.

[6] In this contribution we do not include network pruning as an element in the design phase; however, this is easily done, see further [1], [9], [10].

[7] Note $p_{\text{outlier}} = 0$ for $\varepsilon = 0$ and $p_{\text{outlier}} = 1$ for $\varepsilon = 1$.

6. EXPERIMENTS

We first test the performance of the algorithm on an artificial example with $c = 3$ classes in a 2D input space. The class conditional probabilities are $p(\boldsymbol{x}|\mathcal{C}_i) = [\mathcal{N}(\boldsymbol{\mu}_i, \boldsymbol{I}) + \mathcal{N}(-\boldsymbol{\mu}_i, \boldsymbol{I})]/2$ where $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{I})$ is a 2D Gaussian distribution with mean vector $\boldsymbol{\mu}$ and identity covariance matrix. The mean vectors are given by $\boldsymbol{\mu}_i = 3\cdot[\cos(\pi(2i-1)/6), \sin(\pi(2i-1)/6)]$, $i = 1,2,3$. The prior class probabilities are $p(\mathcal{C}_i) = 1/3$. We generated $N = 300$ data and used $N_t = 150$ for training and $N_v = 150$ for validation. In addition we generated a test set of $N_{\text{test}} = 600$. Further, we introduced outliers by changing class labels with probability $\varepsilon = 0.08$. Suppose that the network weights are given by $\boldsymbol{w} = [\boldsymbol{w}^I, \boldsymbol{w}^I_{\text{bias}}, \boldsymbol{w}^H, \boldsymbol{w}^H_{\text{bias}}]$ where $\boldsymbol{w}^I$, $\boldsymbol{w}^H$ are input-to-hidden and hidden-to-output weights, respectively, and the bias weights are assembled in $\boldsymbol{w}^I_{\text{bias}}$ and $\boldsymbol{w}^H_{\text{bias}}$. In this example, we use the following weight decay regularization term:

$$R(\boldsymbol{w},\boldsymbol{\kappa}) = \kappa^I|\boldsymbol{w}^I|^2 + \kappa^I_{\text{bias}}|\boldsymbol{w}^I_{\text{bias}}|^2 + \kappa^H|\boldsymbol{w}^H|^2 + \kappa^H_{\text{bias}}|\boldsymbol{w}^H_{\text{bias}}|^2 \quad (24)$$

where $\boldsymbol{\kappa} = [\kappa^I, \kappa^I_{\text{bias}}, \kappa^H, \kappa^H_{\text{bias}}]$. The simulation set-up was:

- Network: 2 inputs, 5 hidden neurons, 2 outputs.
- Weights were initialized uniformly over $[-0.5, 0.5]$, regularization parameters were initialized at zero. 30 steps in a gradient descent training algorithm (see e.g., [11]) was performed and the weight decays, $\boldsymbol{\kappa}$, were re-initialized at $\lambda_{\max}/10^4$, where $\lambda_{\max}$ is the max. eigenvalue of the Hessian matrix of the cost function. This prevents initial numerical stability problems. ε is initialized at 0.01.
- Training is now done using a Gauss-Newton algorithm (see e.g., [11]). The Hessian is inverted using the Moore-Penrose pseudo inverse (see e.g., [11]) ensuring that the eigenvalue spread[8] is less than 10^8.
- The step-size μ in Eq. (17) is initialized at 1 and $\boldsymbol{\xi}$ is adapted until the validation error has reached a minimum.
- Finally, weights are retrained on the combined set of training and validation data using the optimized weight decay parameters and outlier probability.

Table 1 reports the average and standard deviations of the probability of misclassification (*pmc*) over 10 runs using the optimal $\boldsymbol{\kappa}$ and β. Note that retraining on the full data set decreases the test *pmc* slightly on average; improvement was found in 5 out of 10 runs.

In Fig. 1 a typical run of the $\boldsymbol{\theta}$ adaptation algorithm is shown. We tested the possibility to detect whether specific examples in the data set, e.g., the combined training/validation set, are outliers and the result is summarized in Table 2. This technique can e.g., be applied to manual inspection of examples which are likely to be outliers. This might lead to relabeling or discovery of new interesting features of the problem.

7. CONCLUSIONS

This paper presented a new framework for design of robust neural classifiers by invoking a probabilitistc model for outliers. We devised an iterative scheme for simultanenous adaptation of regularization parameters and the outlier probabilty. Moreover, we discussed the possibility of detecting outliers. Numerical examples demonstrated the potential of the framework.

[8] Eigenvalue spread should not be larger than the square root of the machine precision [4].

	Initial Neural Net	Optimal Neural Net	Optimal Bayes Decisions
Train.	0.141 ± 0.010	0.145 ± 0.011	0.160
Val.	0.266 ± 0.029	0.241 ± 0.012	0.240
Test	0.278 ± 0.013	0.250 ± 0.009	0.222
Test after retrain.	0.268 ± 0.008	0.249 ± 0.012	0.222

Table 1: Probability of misclassification, when outlier probability is $\varepsilon = 0.08$. For the neural network the averages and standard deviations over 10 runs are reported. Inital and optimal neural net refers to using initial and optimized setting of $\boldsymbol{\kappa}$ and ε. Optimal Bayes decisions boundaries are calculated from the detailed knowledge of the true posterior probabilities. The minimal Bayes error on an infinite set is 0.213. The outlier probability was estimated as $\widehat{\varepsilon} = 0.097 \pm 0.018$.

	True	
Estimated	*not outlier*	*outlier*
not outlier	$a = 0.911 \pm 0.017$	$b = 0.142 \pm 0.022$
outlier	$c = 0.089 \pm 0.017$	$d = 0.858 \pm 0.022$

Table 2: Confusion matrix for outlier detection (over 10 runs) on the combined training/validation set. An example is considered not to be an outlier if $1 - \widehat{p}_{\text{outlier}} > 0.9$. The aim is, e.g., to set the threshold so that the false positive rate $b/(b+d)$ is acceptable small.

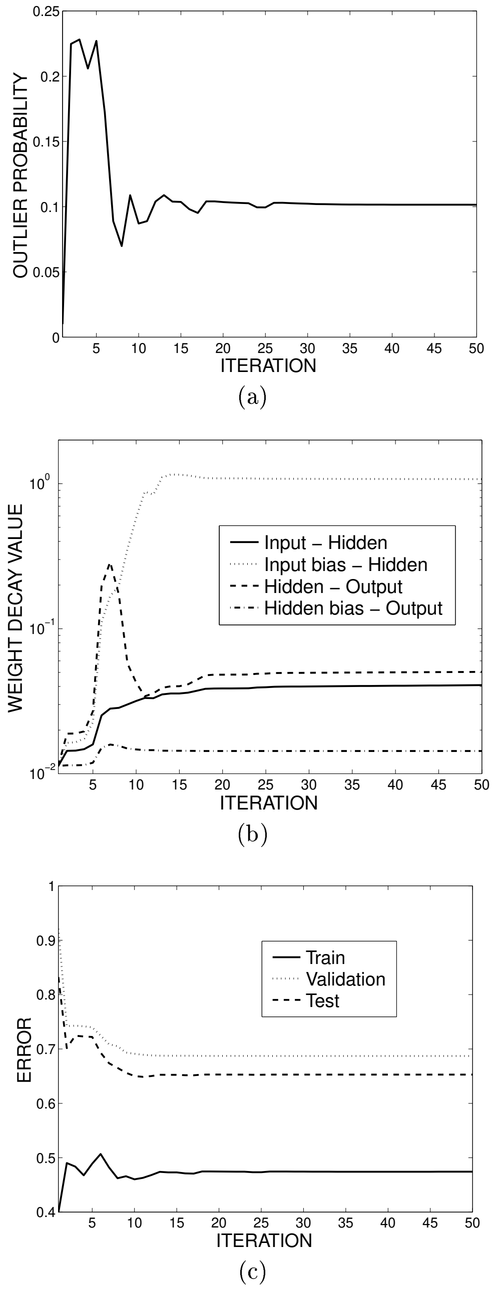

Figure 1: Typical run of the $\boldsymbol{\theta}$ adaptation algorithm. In panel (a) the evolution of the outlier probability $\varepsilon = \beta(c-1)$ is shown. Panel (b) shows the adaptation of the normalized weight decays $\alpha = \kappa \cdot N_t$, and in panel (c) the training, validation and test errors are depicted.

8. REFERENCES

[1] L.N. Andersen, J. Larsen, L.K. Hansen & M Hintz-Madsen: "Adaptive Regularization of Neural Classifiers," in J. Principe *et al.* (eds.) *Proc. IEEE Workshop on Neural Networks for Signal Processing VII*, Piscataway, NJ: IEEE, pp. 24–33, 1997.

[2] C.M. Bishop: *Neural Networks for Pattern Recognition*, Oxford, UK: Oxford University Press, 1995.

[3] J.S. Bridle: "Probabilistic Interpretation of Feedforward Classification Network Outputs with Relationships to Statistical Pattern Recognition," *Neurocomp. - Algor. Arch. and Appl.*, Berlin, Germany: Springer-Verlag, vol. 6, pp. 227–236, 1990.

[4] J.E. Dennis & R.B. Schnabel: *Numerical Methods for Unconstrained Optimization and Non-linear Equations*, Englewood Cliffs, NJ: Prentice-Hall, 1983.

[5] S. Geman, E. Bienenstock & R. Doursat, "Neural Networks and the Bias/Variance Dilemma," *Neural Computation*, vol. 4, pp. 1–58, 1992.

[6] K. Hornik: "Approximation Capabilities of Multilayer Feedforward Networks," *Neural Networks*, vol. 4, pp. 251–257, 1991.

[7] M. Hintz-Madsen, M. With Pedersen, L.K. Hansen, & J. Larsen: "Design and Evaluation of Neural Classifiers," in S. Usui *et al.* (eds.), *Proc. IEEE Workshop on Neural Networks for Signal Processing VI*, Piscataway, NJ: IEEE, 1996, pp. 223–232.

[8] P.J. Huber: *Robust Statistics*, New York, NY: John Wiley & Sons, 1981.

[9] J. Larsen, L.K. Hansen, C. Svarer & M. Ohlsson: "Design and Regularization of Neural Networks: The Optimal Use of a Validation Set," in S. Usui *et al.* (eds.), *Proc. IEEE Workshop on Neural Networks for Signal Processing VI*, Piscataway, NJ: IEEE, 1996, pp. 62–71.

[10] J. Larsen, C. Svarer, L.N. Andersen & L.K. Hansen: "Adaptive Regularization in Neural Network Modeling," appears in G.B. Orr *et al.* (eds.) *"The Book of Tricks"*, Germany: Springer-Verlag, 1997. Generalization

[11] G.A.F. Seber & C.J. Wild: *Nonlinear Regression*, New York, New York: John Wiley & Sons, 1989.