4 Linear Models for Classification

Discusses linear models and *generalised linear models* (GLM). GLM means that even if the prediction functions are non-linear, the decision surfaces are linear.

4.1 Discriminant functions

4.1.1 Two Classes

Describes the geometry of a discriminant function $y(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + w_0$. That is, $\mathbf{w}$'s are orthogonal to decision surface and $|w_0|/||\mathbf{w}||$ describes dislocation from origin.

4.1.2 Multiple Classes

Discuss the limitation of one-vs-rest and one-vs-one classifiers, introduce the benfits of multi-class linear discriminant.

4.1.3 Least-squares Classification

Least squares classification has one extra limitation wrt. limitation of least squares regression: The target vector $\mathbf{t}$ are of 1-of-K type.

4.1.4 Fisher's Linear Discriminant

Perform a dimensionality reduction and then discrimination. $J(\mathbf{w})$ is a function that does this and can be minimised via (4.2.9).

4.1.5 Relation to Least-squares

By changing the target variable representation for the 2-class problem, it's possible to relate Fisher and least-squares.

4.1.6 Fisher's Discriminant for multiclass

Consider generalisation to $K > 2$ classes. The extension is similar to 2-class. Now there are multiple possible choices of (Fisher) criterion.

4.1.7 The Perceptron Algorithm

Construct GLM $y(\mathbf{x}) = f(\mathbf{w}^T\phi(\mathbf{x}))$ where $f(a) = \begin{cases} +1 & a \geq 0 \\ -1 & a < 0 \end{cases}$. Patterns in C_1 become $+1$ and for $x_n \in C_1$ we want $\mathbf{w}^T\phi(\mathbf{x}) > 0$ and for $x \in C_2$ we want it to be < 0. Both can be summarised as $t\mathbf{w}^T\phi(\mathbf{x}) > 0$.

The perceptron criterion minimises error only on misclassified patterns. The weight update algorDenemeithm operates for each sample n:

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta\nabla E_p(\mathbf{w}) = \mathbf{w}^(\tau) + \eta\phi_n t_n \tag{1}$$

where η is the *learning rate*. The update $(\tau + 1)$ happens in the *direction of misclassification* and *guarantees* the error on misclassified sample to be reduced. Of course it doesn't guarantee anything on *all* training samples.

4.2 Probabilistic Generative Models

Construct posterior $p(C_k|\mathbf{x})$ and represent via *logistic sigmoid*:

$$p(C_1|\mathbf{x}) = \frac{p(\mathbf{x}|C_1)p(C_1)}{\sum\limits_{j\in\{1,2\}} p(\mathbf{x}|C_j)p(C_j)} = \frac{1}{1+\exp(-\alpha)} = \sigma(\alpha) \tag{2}$$

where $\alpha = \ln\frac{p(\mathbf{x}|C_1)p(C_1)}{p(\mathbf{x}|C_2)p(C_2)}$ and $\sigma(\alpha)$ is the logistic sigmoid function.

We are interested in situations where $\alpha(\mathbf{x})$ is linear and therefore creates posteriors governed by GLMs.

4.2.1 Continuous Inputs

We sgtart by assuming that all classes C_k share same cov matrix Σ.

For K classes α_k becomes $\alpha_k(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + w_{k0}$ where $\mathbf{w}_k = \Sigma^{-1}\boldsymbol{\mu}_k$ and w_{k0} is as in (4.70).

That is, α_k is linear in $\mathbf{x}$. Decision boundaries (which correspond to misclassification rate) wioll be again linear in $\mathbf{x}$ so again we have GLM.

If we relax the "shared covariance matrix" assumption, then we'll have *quadratic discriminant* rather than GLM.

4.2.2 Maximum likelihood solution

Once $p(\mathbf{x}|C_k)$ defined, we can determine values of its parameters and parameters of $p(C_k)$ via *maximum likelihood*. Construct maximum function:

$$p(\mathbf{T}, \mathbf{X}|\pi, \boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \Sigma) = \prod_n [\pi\mathcal{N}(x_n|\boldsymbol{\mu}_1, \Sigma)]^{t_n}[(1-\pi)\mathcal{N}(x_n|\boldsymbol{\mu}_2, \Sigma)]^{(1-t_n)} \tag{3}$$

In the ML solution we get $\boldsymbol{\mu} = \frac{1}{N_1}\sum_n t_n\mathbf{x}_n$ and $\boldsymbol{\mu} = \frac{1}{N_2}\sum_n (1-t_n)\mathbf{x}_n$.

For covariance Σ, define $\mathbf{S}_1, \mathbf{S}_2, \mathbf{S}$ as:

$$\mathbf{S}_1 = \frac{1}{N_1}\sum_{n\in C_1}(\mathbf{x}_n - \boldsymbol{\mu}_1)(\mathbf{x}_n - \boldsymbol{\mu}_1)^T \tag{4}$$

$$\mathbf{S}_2 = \frac{1}{N_2}\sum_{n\in C_2}(\mathbf{x}_n - \boldsymbol{\mu}_2)(\mathbf{x}_n - \boldsymbol{\mu}_2)^T \tag{5}$$

$$\mathbf{S} = \frac{N_1}{N}\mathbf{S}_1 + \frac{N_2}{N}\mathbf{S}_2 \tag{6}$$

Overall, process not robust to outliers because ML is not.

4.2.3 Discrete Features

4.2.4 Exponential Family

We manage to get GLMs for the above types too.

4.3 Probabilistic Discriminative Models

Advantage: There are less parameters to discover and usually leads to improved performance.

4.3.1 Fixed Basis Functions

4.3.2 Logistic Regression

Here we set M params whereas in generative modelling we set $(M+5)/2+1$ params.

Consider implementing a discriminative function directly as a via logistic sigmoid function:

$$p(C_1|\phi) = y(\phi) = \sigma(\mathbf{w}^T\phi) \tag{7}$$

and naturally $p(C_2|\phi) = 1 - p(C_1|\phi)$. We can set params via ML. We start by seeing that $\frac{d\sigma}{d\alpha} = \sigma(1-\sigma)$ (exercise 4.12). Likelihood can be written as:

$$p(\mathbf{T}|\mathbf{w}) = \prod_n y_n^{t_n}(1-y_n)^{1-t_n} \tag{8}$$

cross entropy error

where $y_n = p(C_1|\phi_n)$. Error function here is also called cross entropy error:

$$E(\mathbf{w}) = -\ln p(\mathbf{T}|\mathbf{w}) = -\sum_n [t_n \ln y_n + (1-t_n)\ln(1-y_n)] \tag{9}$$

Taking the gradient wrt $\mathbf{w}$:

$$\nabla E(\mathbf{w}) = \sum_n (y_n - t_n)\phi_n \tag{10}$$

4.3.3 Iterative Reweighted Least Squares

We no longer have closed-form solution (as we did for regression). Fortunately the error function is still convex there is the (iterative) Newton-Raphson or iterative reweighted least squares algorithm:

Newton-Raphson or iterative reweighted least squares

$$\mathbf{w}^{(new)} = \mathbf{w}^{(old)} = \mathbf{H}^{-1}\nabla E(\mathbf{w}) \tag{11}$$

where $\mathbf{H}$ is the hassian matrix whose elements comprise the second derivs of $E(\mathbf{w})$ wrt components of $\mathbf{w}$.

$$\nabla E(\mathbf{w}) = \sum_n (y_n - t_n)\phi_n = \mathbf{\Phi}^T(\mathbf{Y} - \mathbf{T}) \tag{12}$$

$$\mathbf{H} = \nabla\nabla E(\mathbf{w}) = \sum_n y_n(1 - y_n)\phi_n\phi_n^T = \mathbf{\Phi}^T\mathbf{R}\mathbf{\Phi} \tag{13}$$

design matrix

where $\mathbf{\Phi}$ is the $N \times M$ design matrix whose nth row is given by ϕ_n^T and $\mathbf{R}$ is the $N \times N$ diagonal matrix with elements $\mathbf{R}_{nn} = y_n(1 - y_n)$.

4.3.4 Multiclass logistic regression

The formalism is similar to 2-class logistic regression. Instead of sigmoid we use the *softmax* function. Again we have *cross-entropy* function as error function. The multiclass version of cross-entropy is:

$$E(\mathbf{w}_1, \ldots, \mathbf{w}_K) = -lnp(\mathbf{T}|\mathbf{w}_1, \ldots, \mathbf{w}_K) = -\sum_n \sum_k t_{nk} - \ln y_{nk}. \tag{14}$$

Again we can use *iterative reweighted least squares* (#210).

4.3.5 Probit regression

The inverse probit function (or the similar erf function) are similar to sigmoid in shape but have more plausible analytical properties. Will be discussed in Sec. 4.5.

4.3.6 Canonical link function

This is one of the most frequently-referred sections of the book. The choices of sigmoid/softmax in earlier sections were not arbitrary — they were chosen to convert the error function to a simple form that involves $y_n - t_n$. This is a general result of assuming a conditional distribution for the activation function known as the canonical link function.

canonical link function

A GLM is a model for which y is a nonlinear function of a linear combination of input variables:

$$y = f(\mathbf{w}^T\phi) \tag{15}$$

where $f(\cdot)$ is the *activation function* and $f^{-1}(\cdot)$ is known as the *link function*.

Let the conditional distro be $p(\mathbf{T}|\eta, s)$. We formulate its derivative in the following form:

$$\nabla_{\mathbf{w}} \ln p(\mathbf{T}|\eta, s) = \ldots (\text{see } \#213) = \sum_n \frac{1}{s}\psi'(y_n)f'(y_n)\phi_n \tag{16}$$

. The canonical link function chosen as $f^{-1}(y) = \psi(y)$ provides a great simplification:

$$\nabla E(\mathbf{w}) = \frac{1}{s}\sum_n (y_n - t_n)\phi_n \tag{17}$$

4.4 The Laplace Approximation

To perform closed-form analysis for Bayesian logistic regression, we'll need to do approximation. The Laplace approx. is used for this purpose. Approximation is performed by matching the *mode* of the target distribution with the mode of a Gaussian via Taylor expansion (where the first-order term disappears as expansion is made around a local maximum). Let $\mathbf{z}_0$ be the mode of the target distribution. The 2^{nd} order Taylor expansion around $\mathbf{z}_0$ is:

$$f(\mathbf{z}) \approx \ln f(\mathbf{z}_0) - \frac{1}{2}(\mathbf{z} - \mathbf{z}_0)^T\mathbf{A}(\mathbf{z} - \mathbf{z}_0). \tag{18}$$

This will enable us to compute the approximated distribtion $q(\mathbf{z})$ directly as $q(\mathbf{z}) = \mathcal{N}(\mathbf{z}|\mathbf{z}_0, \mathbf{A})$.

Better methods will be explored in Chapter 10

Better methods will be explored in Chapter 10.

4.4.1 Model comparison and BIC

We can use the approximation above for model comparison, which will lead to Bayesian Information Criterion (BIC). Start with the normalisation term:

$$Z \approx f(\mathbf{z}_0) \int \exp\left[-\frac{1}{2}(\mathbf{z}-\mathbf{z}_0)^T \mathbf{A}(\mathbf{z}-\mathbf{z}_0)\right] d\mathbf{z} = f(\mathbf{z}_0)\frac{(2\pi)^{M/2}}{|\mathbf{A}|^{1/2}}. \tag{19}$$

Consider data set $\mathcal{D}$ and models $\{\mathcal{M}_i\}$ with parameters $\{\boldsymbol{\theta}_i\}$. For each model we define a likelihood function $p(\mathcal{D}|\boldsymbol{\theta}_i, \mathcal{M}_i)$ — or shortly, $p(\mathcal{D}|\boldsymbol{\theta}_i)$.

Defining $f(\boldsymbol{\theta}) = p(\mathcal{D}|\boldsymbol{\theta})p(\theta)$ and identifying that $Z = p(\mathcal{D})$, we can apply the result above to get:

$$\ln p(\mathcal{D}) \approx \ln p(\mathcal{D}|\boldsymbol{\theta}_{\text{MAP}}) + \overbrace{\ln p(\boldsymbol{\theta}_{\text{MAP}}) + \frac{M}{2}\ln(2\pi) - \frac{1}{2}\ln|\mathbf{A}|}^{\text{Occam factor}}. \tag{20}$$

With further simplifications via (not necessarily realistic) assumptions (see #217) we get the BIC:

$$\ln p(\mathcal{D}) \approx \ln p(\mathcal{D}|\boldsymbol{\theta}_{\text{MAP}}) - \frac{1}{2}M \ln N \tag{21}$$

Essentially this is an information criterion that penalizes model complexity

4.5 Bayesian Logistic Regression

Again, exact inference for logistic regression is intractable, due to normalisation (which involves likelihood computation, which is a product of sigmoids (one for each data point)). We apply Laplace approximation for tractability.

4.5.1 Laplace Approximation

Because Laplace involves Gaussian approx, we start with Gaussian prior $p(\mathbf{w}) = \mathcal{N}(\mathbf{w}|\mathbf{m}_0, \mathbf{S}_0)$. Posterior is $p(\mathbf{w}|\mathbf{T}) \propto p(\mathbf{w})p(\mathbf{T}|\mathbf{w})$. To compute the approx of this posterior, $q(\mathbf{w})$, we first express it in closed form (4.142) and then find the MAP solution, $\mathbf{w}_{\text{MAP}}$. Then we compute the approximation around $\mathbf{w}_{\text{MAP}}$; that is, we find the cov. matrix S_N by using (4.132) — the mean is already $\mathbf{w}_{\text{MAP}}$ – and obtain approx as: $p(\mathbf{w}|\mathbf{T}) \approx q(\mathbf{w}) = \mathcal{N}(\mathbf{w}|\mathbf{w}_{\text{MAP}}, \mathbf{S}_N)$.

Now we'll use this for prediction by computing the *predictive distribution*.

4.5.2 Predictive Distribution

Recall that for 2-class NN the prob of a sample being C_1 is the output of the NN, which is $p(C_1|\boldsymbol{\phi}, \mathbf{w}) = \sigma(\mathbf{w}^T\phi)q(\mathbf{w})$. Predictive distribution involves the following marginalization over $\mathbf{w}$:

$$p(C_1|\boldsymbol{\phi}, \mathbf{T}) = \int p(C_1|\boldsymbol{\phi}, \mathbf{w})p(\mathbf{w}|\mathbf{T}) \approx \int \sigma(\mathbf{w}^T\boldsymbol{\phi})q(\mathbf{w})d\mathbf{w}. \tag{22}$$

The problem here is that $\sigma(\cdot)$ is non-linear and again not tractable. Putting this aside for a moment, the section first computes the marginalization of $q(\mathbf{w})$, $\int q(\mathbf{w})d\mathbf{w}$. Then, it computes the overall integral in (22) by approximating the sigma function with the inverse probit function, $\Phi(\cdot)$ — see #219 for details.

5 Neural Networks

This is a critical chapter because many commonly-used techniques are motivated and described in detail. These include the *gradient-descent optimization* and also the *backpropagation* technique which is used for many purposes including *Hessian computation* and *Jacobian computation*.

5.1 Feed-forward Neural Networks

5.2 Network Training

Again we'll minimize an error function. We can directly minimize a sum-of-squares error such as $E(\mathbf{w}) = \frac{1}{2}\sum ||\mathbf{y}(\mathbf{x}_n - \mathbf{t}_n)||^2$. But we'll give rise to a probabilistic interpretation by considering likelihood maximization. This will be beneficial for many purposes.

The rest of the section derives the energy function and cross-entropy error function in the context of NNs.

5.2.1 Parameter optimisation

Clearly, there is no hope to find an analytical solution to optimum $\mathbf{w}$. We'll therefore resort to iterative algorithms of the following form:

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} + \Delta\mathbf{w}^{(\tau)}. \tag{23}$$

There are many algos of this form, and they differ by their choice of $\Delta\mathbf{w}^{(\tau)}$. Most use *gradient information* due to reasons discussed in following section.

5.2.2 Local quadratic approximation

Consider second-order Taylor approx of $E(\mathbf{w})$ around some $\widehat{\mathbf{w}}$:

$$E(\mathbf{w}) \approx E(\widehat{\mathbf{w}}) + (\mathbf{w} - \widehat{\mathbf{w}})^T\mathbf{b} + \frac{1}{2}(\mathbf{w} - \widehat{\mathbf{w}})^T\mathbf{H}(\mathbf{w} - \widehat{\mathbf{w}}) \tag{24}$$

where $\mathbf{b} = \nabla E_{\mathbf{w}}(\mathbf{w})|_{\mathbf{w}=\widehat{\mathbf{w}}}$ and $\mathbf{H}$ is the Hessian matrix with elements $(\mathbf{H})_{ij} = \frac{\partial^2 E}{\partial w_i \partial w_j}|_{\mathbf{w}=\widehat{\mathbf{w}}}$. If we pick $\widehat{\mathbf{w}}$ to be a minimum, say $\mathbf{w}^\star$ the second term above vanishes.

$$E(\mathbf{w}) \approx E(\mathbf{w}^\star) + \frac{1}{2}(\mathbf{w} - \mathbf{w}^\star)^T\mathbf{H}(\mathbf{w} - \mathbf{w}^\star) \tag{25}$$

We analyse $\mathbf{H}$ by considering its eigenvectors, and we see that constant-error contours $E(\mathbf{w}) = C$ are ellipses whose axes are aligned with the eigenvectors of $\mathbf{H}$, $\mathbf{u}_i$, and the length of these axes are inverse proportional to the corresponding eigenvectors, λ_i (see #238-239 and Exercise 5.10).

5.2.3 Use of gradient information

Gradient allows us to compute evaluate E in $O(W^2)$ instead of $O(W^3)$.

5.2.4 Gradient descent optimization

Standard (*i.e.* batch) gradient descent ($\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta\nabla E(\mathbf{w})$.) leads to poor performance. However we can perform gradient-descent for each sample in dataset separately, which is known as on-line gradient descent (or stochastic GD or sequential GD), which proved to be much better.

But for batch optimization there are much more efficient methods such as *conjugate gradients* and *quasi-Newton* methods.

5.3 Error Backpropagation

The main goal is to find efficient methods to compute the gradient of $E(\mathbf{w})$, $\nabla E(\mathbf{w})$. The importance of backpropagation lies in that it can be used beyond the scope of NNs and gradient descent, such as other derivatives and graphical models etc.

There are 2 steps at each iteration of a backpropt technique:

1. Evaluate $\nabla E(\mathbf{w})$ (this is where backpropagation comes to play)
2. Update $\mathbf{w}^\star$ based on step 1.

5.4 Evaluation of error-function derivatives

We will analyse the error of a single sample, E_n, and the total error is simply the sum over N. Let δ_j be defined as

$$\delta_j = \frac{\partial E_n}{\partial a_j}, \tag{26}$$

where δ's are typically referred to as errors. In the last layer, due to our choice of activation function, we have (see 5.18):

$$\delta_k = \frac{\partial E_n}{\partial a_k} = y_k - t_k \tag{27}$$

The ultimate goal is to compute derivatives $\frac{\partial E_n}{\partial w_{ji}}$ and the δ's will be the messages that are propagated backwards.

We can express the desired derivative by means of the output and label as:

$$\frac{\partial E_n}{\partial w_{ji}} = (y_{nj} - t_{nj}) x_{ni} \tag{28}$$

Note that by feed-forward NN definition we have $a_j = \sum_i w_{ji} z_i$ and $z_j = h(a_j)$. We can therefore express the derivative above through the chain rule as:

$$\frac{\partial E_n}{\partial w_{ji}} = \frac{\partial E_n}{\partial a_j} \frac{\partial a_j}{\partial w_{ji}} = \delta_j \frac{\partial a_j}{\partial w_{ji}} \tag{29}$$

Note that the first derivative on the rhs corresponds to our d_j definition.

Now we can start from the last (output) layer and propagate backwards:

$$\delta_j = \frac{\partial E_n}{\partial a_j} = \sum_k \frac{\partial E_n}{\partial a_k} \frac{\partial a_k}{\partial a_j} = \sum_k \delta_k w_{kj} \tag{30}$$

where the units k are those to which j sends connections. Here $\frac{\partial a_k}{\partial a_j} = w_{kj}$ holds because the kth layers are output layers, and we are considering the regression problem where the activation function ($h(\cdot)$) is simply the unity function and therefore:

$$a_k = \sum_i w_{ki} a_i. \tag{31}$$

Now let us consider a node j that lies in one layer before where the activation function is not unity and therefore:

$$a_j = \sum_i w_{ji} h(a_i) \tag{32}$$

and therefore we have

$$\delta_j = h'(a_j) \sum_k w_{kj} \delta_k. \tag{33}$$

We can summarise backpropagation as follows:

1. Apply an input vector $\mathbf{x}_n$ to the network and propagate forwards
2. Evaluate δ_k for output units through (5.54) (or (30 above)
3. Evaluate δ_j for all hidden units in between input and output layers (or (33) above).
4. Finally obtain the derivative $\frac{\partial E_n}{\partial w_{ji}}$ through (5.53), which is $\frac{\partial E_n}{\partial w_{ji}} = \delta_j z_i$.

5.4.1 Efficiency of backpropagation

Backpropagation's main advantage is efficiency. Derivatives can be computed with central differences method as in (5.69), however, that is too costly. But the central differences method can be used to check the correctness of a backpropagation method.

5.4.2 The Jacobian Matrix

In addition to the derivatives of the error function, backprop can also be used for the Jacobian:

$$J_{ki} = \frac{\partial y_k}{\partial x_i} \tag{34}$$

We similarly will try to derive a message passing algorithm. Jacobian can be written out as:

$$J_{ki} = \frac{\partial y_k}{\partial x_i} = \sum_j \frac{\partial y_k}{\partial a_j}\frac{\partial a_j}{\partial x_i} = \sum_j w_{ji}\frac{\partial y_k}{\partial a_j} \tag{35}$$

where j's represent the nodes to which node i sends connection. To compute the derivative above, we compute the second term in the rhs as:

$$\frac{\partial y_k}{\partial a_j} = \sum_l \frac{\partial y_k}{\partial a_l}\frac{\partial a_l}{\partial a_j} = h'(a_j)\sum_l w_{lj}\frac{\partial y_k}{\partial a_l} \tag{36}$$

Again we start at output units, see #248 for further details.

5.5 The Hessian Matrix

This section is critical as it discusses in detail various types of Hessian computation, including Hessian computation via backprop. The different methods for Hessian computations are due to the different assumptions which lead to different approximations. Exact Hessian computation (*i.e.* no assumptions) is also possible via backprop. Among other things, Hessian is useful for:

1. Finding non-significant weights
2. Computing Laplace approximation for Bayesian Networks
3. Non-linear optimization techniques
4. Efficient method for re-training network

5.5.1 Diagonal Approximation

The first way is to discard non-diagonal elements in which case we have a quite simple computation for Hessian. However Hessian can be strongly non-diagonal therefore these assumptions must be treated with care.

5.5.2 Outer product approximation (Levenberg-Marquardt approximation)

For sum-of-squares Error function, the Hessian can be written out as:

$$\mathbf{H} = \nabla\nabla E = \sum_{n=1}^{N} \nabla y_n (\nabla y_n)^T + \sum_{(} y_n - t_n)\nabla\nabla y_n. \tag{37}$$

Levenberg-Marquard approximation

The second term is likely to be very small and can be neglected, which leads to *outer product* or Levenberg-Marquard approximation:

$$\mathbf{H} \approx \sum_{n=1}^{N} \nabla a_n (\nabla a_n)^T = \sum_{n=1}^{N} \mathbf{b}_n \mathbf{b}_n^T. \tag{38}$$

Similarly, for cross-entropy error function we obtain:

$$\mathbf{H} \approx \sum_{n=1}^{N} y_n(1-y_n)\mathbf{b}_n \mathbf{b}_n^T \tag{39}$$

5.5.3 Inverse Hessian

The outer product approx above leads also to efficient methods for computing the inverse of Hessian. The Hessian can be updated sequentially for each point as:

$$\mathbf{H}_{L+1} = \mathbf{H}_L + \mathbf{b}_{L+1}\mathbf{b}_{L+1}^T \tag{40}$$

and using the Woodbury identity (C.7) the inverse can be computed as 5.89.

5.5.4 Finite differences

The correctness of the Hessian and its inverse can again be checked by comparing it to the output of finite differences method.

5.5.5 Exact evaluation of the Hessian

The backprop procedure can be used for Hessian as well. Again we derive a message passing scheme by defining some δ's — see #253-254.

5.5.6 Fast multiplication by the Hessian

Many times we are interested in computing a multiplication of the Hessian with a vector $\mathbf{Hv}$ rather than computing the Hessian $\mathbf{H}$ itself. This is also possible via the backprop algo — see #254-256.

5.6 Regularization in Neural Networks

5.6.1 Consistent Gaussian Priors

In Neural nets, different sets of priors can be equivalent to each other due to the bias parameter. A well-defined regularizer should be aware of this and not favour one equivalent solution to the other. We can achieve this via the consistent Gaussian priors discussed here. The section will be used for automatic relevance determination in RVMs.

automatic relevance determination in RVMs

5.6.2 Early Stopping

An alternative is to prevent overfitting is early stopping – stop the training as soon as the validation error starts getting larger.

5.6.3 Invariances

There are four methods to implement invariance for certain transformations:

1. Augment training set with replicas
2. Tangent propagation (manifold learning)
3. Invariance during feature extraction
4. Shared weights (for limited types of invariances)

5.6.4 Tangent propagation

Provided that a transformation is continuous, then the transformed pattern will sweep out a manifold $\mathcal{M}$ in a D-dimensional space. Assume that this transformation acting on a vector $\mathbf{x}_n$ is given via the function $\mathbf{s}(\mathbf{x}_n, \xi)$ where ξ controls the amount of transformation.

Under a transformation, the output of the network will usually change — but we don't want it to. So we punish changes on the output.

tangent distance

A related technique is tangent distance where invariance is built into the distance metric (Simard *et al.* , 1993).

5.6.5 Training with transformed data

Training with transformed data is closely related to tangent propagation. The section discusses this analytically.

5.6.6 Convolutional networks

Discuss the weight-sharing of CNNs.

5.6.7 Soft weight sharing

CNNs share weights by making them identical. This is a softer version of sharing weights. This section model soft weights as GMM.

5.6.8 Mixture Density Networks

Potentially useful

We can create a very flexible output by having a Gaussian mixture of NNs. Potentially useful technique for many purposes — it can model arbitrary distributions. The conditional mode of a distribution is likely to be an important property (see end of section and Fig. 5.21d).

5.7 Bayesian Neural Networks

So far we focused on ML for finding the weight parameters. Now we consider Bayesian methods. We'll ultimately be interested in making predictions. That is, as far as Bayesian is concerned, computing the predictive distribution.

The conditional distro for one sample is:

$$p(t|\mathbf{x}, \mathbf{w}, \beta) = \mathcal{N}(t|y(\mathbf{x}, \mathbf{w}, \beta^{-1})). \tag{41}$$

Then we have the prior:

$$p(\mathbf{w}|\alpha) = \mathcal{N}(\mathbf{w}|\mathbf{0}, \alpha^{-1}\mathbf{I}) \tag{42}$$

We compute likelihood for dataset $\mathcal{D}$ as $p(\mathcal{D}|\mathbf{w}, \beta)$. Then posterior is

$$p(\mathbf{w}|\mathcal{D}, \alpha, \beta) \propto p(\mathcal{D}|\mathbf{w}, \beta)p(\mathbf{w}, \alpha). \tag{43}$$

We'll ultimately be interested in computing the predictive distro: $p(t|\mathbf{x}, \mathcal{D}, \alpha, \beta)$. We'll also be optimizing α, β and evantually will have $p(t|\mathbf{x}, \mathcal{D})$.

We work with Laplace approximation due to intractability of exact analytical solutions.

First we make Laplace approx for the posterior $p(\mathbf{w}|\mathcal{D})$ — the approx is denoted with $q(\mathbf{w}|\mathcal{D})$. But even with this approx the following integral for predictive distro is intractable (see #279):

$$p(t|\mathbf{w}, D) = \int p(t|\mathbf{x}, \mathbf{w})q(\mathbf{w}|\mathcal{D})d\mathbf{w}. \tag{44}$$

To make progress we'll approximate the output function $\mathbf{y}(\mathbf{x}, \mathbf{w})$

Miscellaneous

Model Comparison The more rigorous section is Sec. 3.4 (and 3.5) with a proper treatment of a theoretically plausible model selection approach. AIC (see 1.73) and BIC (Sec 4.4.1, #217) offer simpler model comparison criteria.