10 Overfitting and generalization

10.1 Generalization

The real goal when modeling is to generalize to new data, not just perform well on the training data set that is presented during training. In general, the error on the training data will be a biased estimate of the generalization error. To be specific, it will tend to be smaller than the generalization error if we select our model such that it minimizes the training error. We can therefore not use the training error as our selection criteria. Instead, we must try to estimate the generalization error.

10.1.1 Cross validation

One way to estimate the generalization error is to do cross-validation. This means using a test data set, which is a subset of the available data (typically 25-35%) that is removed before any training is done, and which is not used again until all training is done. The performance on this test data will be an unbiased estimate of the generalization error, provided that the data has not been used in any way during the modeling process. If it has been used, e.g. for model validation when selecting hyperparameter values, then it will be a biased estimate.

If there is lots of data available then it may be sufficient to use one test set for estimating the generalization error. However, if data is scarce then it is necessary to use more data-efficient methods. One such method is the K-fold cross-validation method.

The central idea in K-fold cross-validation is to repeat the cross-validation test K times. That is, divide the available data into K subsets, here denoted by $\mathcal{D}_k$, where each subset contains a sample of the data that reflects the data distribution (i.e. you must make sure that one subset does not contain e.g. only one category in a classification task). The procedure then goes like this:

1. Repeat K times, i.e. until all data subsets have been used for testing once.

   1.1 Set aside one of the subsets, $\mathcal{D}_k$, for testing, and use the remaining data subsets $\mathcal{D}_{i \neq k}$ for training.

1.2 Train your model using the training data.

1.3 Test your model on the data subset $\mathcal{D}_k$. This gives you a test data error $E_{test,k}$.

2. The estimate of the generalization error is the mean of the K individual test errors: $E_{gen.} = \frac{1}{K}\sum_k E_{test,k}$.

One benefit with K-fold cross-validation is that you can estimate an error bar for the generalization error by computing the standard deviation of the $E_{test,k}$ values.

Note: The errors $E_{test,k}$ are often approximately log-normally distributed. At least, $\log E_{test,k}$ tends to be more normally distributed than $E_{test,k}$. It is therefore more appropriate to use the mean of the logs as an estimate for the log generalization error. That is

$$\log E_{gen.} = \frac{1}{K}\sum_{k=1}^{K} \log E_{test,k} \tag{10.1}$$

$$\Delta \log E_{gen.} = 1.96\sqrt{\frac{1}{K-1}\sum_{k=1}^{K}\left[\log E_{test,k} - \log E_{gen.}\right]^2} \tag{10.2}$$

where the lower row is a 95% confidence band for the log generalization error.

An error bar from cross-validation includes the model variation due to both different training sets and different initial conditions.

10.2 Overfitting

One of the most important issues when using nonlinear regression models, like neural networks, is how to avoid overfitting to the training data. It is very easy, when working with a flexible nonlinear model, to fit to peculiarities in the training data (e.g. noise). If this happens, then the final model will actually get worse on new data, i.e. it will have poor generalization performances.

The usual way to illustrate overfitting is the *model bias versus model variance* picture, which we covered earlier in the course. In this framework, we assume a data generating process

$$y(n) = g(\boldsymbol{x}(n)) + \varepsilon(n), \tag{10.3}$$

and use the fact that the generalization error can be separated into three terms

$$\begin{aligned}\langle E_{gen}\rangle_w &= \int \langle [f(\boldsymbol{x};\boldsymbol{w}) - \langle f(\boldsymbol{x};\boldsymbol{w})\rangle_w]^2\rangle_w p(\boldsymbol{x}) d^D x \\ &+ \int [g(\boldsymbol{x}) - \langle f(\boldsymbol{x};\boldsymbol{w})\rangle_w]^2 p(\boldsymbol{x}) d^D x + \sigma_\varepsilon^2 \\ &= \text{Variance} + \text{Bias}^2 + \text{noise variance.} \qquad (10.4)\end{aligned}$$

Thus, minimizing the generalization error is not equivalent to selecting a model family $\mathcal{F}$ where the model bias is zero, i.e. where $g \in \mathcal{F}$. This is because the model variance penalty associated with $\mathcal{F}$ may be too high. In fact, it may be beneficial to use a model family with non-zero bias if the variance penalty for all zero bias families is too high (this is a counter-intuitive result, but it is a result of the limited size of the training set). A small model variance ususally means a large model bias, and vice versa.

A small model family $\mathcal{F}$ means a small model variance, but a higher probability for a large model bias. A large model family $\mathcal{F}$ means a large model bias, but a small probability for a model bias.

When we have a large generalization error due to a large model bias, then we speak of "underfitting". This means that the model is not flexible enough to capture the underlying process. When we have a large generalization error due to a large model variance, then we speak of "overfitting". This means that the model family is too flexible for the limited training data set we are using. The optimal model, with respect to training data size and generalization error, is in between these two extremes.

Another way to phrase the "overfitting" problem is the ill-posed problem picture, introduced by Tikhonov (Tikhonov & Arsenin 1977). Tikhonov lists the following three requirements for a modeling problem (i.e. a "question") to be well-posed:

(i) The model (e.g. neural network) can learn the function, i.e. there *exists* a solution $f \in \mathcal{F}$ such that $f = g$.

(ii) The solution is *unique*.

(iii) The solution is *stable* under small variations in the training data set. For instance, training with two slightly different training data sets sampled from the same process must result in similar solutions (similar when evaluated on e.g. test data).

The third requirement is really important. If the model changes significantly with slight changes in the training set then it will for sure have very poor generalization properties.

The first requirement, (i), corresponds to requiring zero model bias, and the third requirement, (iii), corresponds to requiring zero model variance. Of course, achieving both simultaneously is unlikely.

Tikhonov argues that any modeling problem that does not meet the three requirements (i)-(iii) is ill-posed. An ill-posed question does not have a well defined answer. We must therefore complement the question with some additional information so that we have a well-posed question that we can expect a useful answer to.

There are several methods to control the model variance and model bias, for instance

- "Early stopping"
- Regularization
- Combining models into committees
- Pruning the model
- Growing the model

We will discuss some of these strategies here, the remaining ones are discussed in (Bishop 1995) and (Haykin 1999).

10.2.1 Early stopping

Early stopping is a technique that has developed within the neural network community. The reason being that backpropagation is such a notoriously slow algorithm that it is possible to monitor its progress and stop before it reaches the point where it has minimized the training error.

The idea is to use two data sets, one for training and one for validating the generalization performance. Typically, both the training and validation errors will decrease initially but the validation error will start to increase at some point. The early stopping algorithm is to stop the learning when the error on the validatio set starts to increase. (This is easy to describe but it is far from simple to implement in real life as shown in (Prechelt 1998).)

The reason why it works is because the learning typically starts from small random values for the weights. This corresponds to a simple, essentially linear, function. However, as training proceeds the weights grow and the function becomes more and more nonlinear, i.e. the model complexity grows. The stopping point thus controls the size of the model family $\mathcal{F}$.

10.2.2 Regularization

The term "regularization" is used for the technique where the summed square error is complemented with additional terms in order to make the modeling problem more well-posed. That is, the cost that is minimized during learning is

$$E = \frac{1}{N}\sum_{n=1}^{N}[y(n) - \hat{y}(n)]^2 + \lambda R(\mathbf{W}), \tag{10.5}$$

where the function $R(\mathbf{W})$ expresses a prior assumption about the model $\hat{y}$. The parameter λ, usually called the "regularization parameter" controls the relative importance of $R(\mathbf{W})$ versus the mean square error. Here, I use the notation $\mathbf{W}$ for a vector containing all the weights in the model.

The most common regularization example is "weight decay" where $R(\mathbf{W}) = \|\mathbf{W}\|^2$. In this case the training cost is

$$E = \frac{1}{N}\sum_{n=1}^{N}[y(n) - \hat{y}(n)]^2 + \lambda \sum_k w_k^2. \tag{10.6}$$

The regularization parameter λ is effectively a knob that controls the model bias and variance. When $\lambda \to \infty$ then the model variance goes to zero, but the bias grows, because the mean square error term becomes unimportant. When $\lambda \to 0$ then the model variance grows, but the bias shrinks, because the mean square error becomes the only important thing to consider.

The name "weight decay" comes from the fact that gradient descent on (10.6) leads to a weight update of the form

$$w_k(t+1) = (1 - \eta\lambda)w_k(t) - \eta\frac{\partial \mathrm{MSE}}{\partial w_k}. \tag{10.7}$$

That is, the weight w_k slowly decays away if the gradient of the mean square error is small.

Another observation is that weight decay corresponds to the "ridge regression" method discussed in context with linear regression earlier. This means that the Bayesian interpretation of weight decay is the same, i.e. that we impose a gaussian prior on the weights

$$p(\mathbf{W}) \sim \exp\left(\frac{-\|\mathbf{W}\|^2}{2\sigma^2}\right). \tag{10.8}$$

This technique is of course easily extended to other forms of prior assumptions. One is the assumption that the prior for $\mathbf{W}$ should be the product of several gaussians whose means and variances are unknown (but fitted from data). This leads to the "soft weight sharing" algorithm discussed in (Bishop 1995).

Another type of prior belief is to constrain the curvature of the function, which can also be expressed in terms of weight penalties. This is discussed both in (Bishop 1995) and (Haykin 1999). One method for doing this is e.g. to have

$$R(\mathbf{W}) = \sum_{j=1}^{M} v_j^2 \left(\sum_{k=1}^{D} w_{jk}^2\right)^m \tag{10.9}$$

where m is the order of the smoothness.

10.2.3 Committees

A committee machine is a common name for combinations of models. It is not unusual that a modeling project will result in several models that are "equal" (i.e. whose errors are not significantly different). These models can be either the result of training with different initial conditions, in which case the models have the same set of inputs, or they can have different inputs and also be of different type (e.g. polynomial, MLP models, RBF models).

Haykin devotes a whole chapter to committee machines.

Averaging committe Suppose we have a set of K models that are equal, i.e. we can not make a statistical distinction between them. Each model produces an output $\hat{y}_k$. Instead of selecting one of these models at random (it would be at random since we can not decide which one is the best) we could average their outputs

$$\hat{y}_{com}(n) = \frac{1}{K}\sum_{k=1}^{K}\hat{y}_k(n). \tag{10.10}$$

This averaging committe will have a sum square test error (or mean square test error) that is never worse than the randomly selected model.

The test MSE from model k is

$$\mathrm{MSE}_k = \frac{1}{N}\sum_{n=1}^{N}\left[y(n) - \hat{y}_k(n)\right]^2, \tag{10.11}$$

where the sum runs over the test data. The expected MSE if we select any of the K models at random is

$$\langle \mathrm{MSE}\rangle_k = \frac{1}{K}\sum_{k=1}^{K}\mathrm{MSE}_k, \tag{10.12}$$

and the MSE from the committee is

$$\mathrm{MSE}_{com} = \frac{1}{N}\sum_{n=1}^{N}\left[y(n) - \hat{y}_{com}(n)\right]^2. \tag{10.13}$$

It is simple to show that (10.13) is always less or equal to (10.12) by using the Cauchy-Schwartz inequality for two vectors $\mathbf{a}$ and $\mathbf{b}$

$$(\mathbf{a}^T\mathbf{b})^2 \le \|\mathbf{a}\|^2\|\mathbf{b}\|^2. \tag{10.14}$$

If we use the K-dimensional vectors

$$\mathbf{a} = \begin{pmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{pmatrix} \text{ and } \mathbf{b} = \begin{pmatrix} \hat{y}_1 \\ \hat{y}_2 \\ \vdots \\ \hat{y}_K \end{pmatrix} \tag{10.15}$$

then the Cauchy-Schwartz inequality tells us that

$$\left(\sum_{k=1}^{K}\hat{y}_k\right)^2 = (\mathbf{a}^T\mathbf{b})^2 \le \|\mathbf{a}\|^2\|\mathbf{b}\|^2 = K\sum_{k=1}^{K}\hat{y}_k^2. \tag{10.16}$$

If we expand the squares (10.12) and (10.13) then we get

$$\langle \mathrm{MSE}\rangle_k = \frac{1}{N}\sum_n y^2(n) - \frac{2}{NK}\sum_n\sum_k y(n)\hat{y}_k(n)$$

$$
\begin{aligned}
&+ \frac{1}{NK}\sum_n \sum_k \hat{y}_k^2(n) && (10.17)\\
\mathrm{MSE}_{com} &= \frac{1}{N}\sum_n y^2(n) - \frac{2}{NK}\sum_n\sum_k y(n)\hat{y}_k(n) \\
&+ \frac{1}{NK^2}\sum_n \left(\sum_k \hat{y}_k(n)\right)^2 && (10.18)
\end{aligned}
$$

and we see immediately, using (10.16), that $\mathrm{MSE}_{com} \leq \langle \mathrm{MSE}\rangle_k$.

Weighted committees Sometimes we end up with different models that are significantly different in terms of MSE but which use very different inputs. In this case it is better to use a weighted committee, where we give more weight to the better models than to the worse models. There are many ways to do this weighting and Bishop (Bishop 1995) discusses it in the general case of having correlated models. However, if the models are uncorrelated then one way to do the weighting is

$$
\begin{aligned}
\hat{y}_{wcom} &= \sum_{k=1}^{K} \alpha_k \hat{y}_k(n) && (10.19)\\
\text{with } \alpha_k &= \frac{\mathrm{MSE}_k^{-1}}{\sum_{j=1}^{K}\mathrm{MSE}_j^{-1}}. && (10.20)
\end{aligned}
$$

This weighted committe will, if the errors are uncorrelated and the MSE values are correctly estimated, produce an MSE that is never worse than the best individual model!

The weighted committee MSE can be expanded into

$$
\begin{aligned}
\mathrm{MSE}_{wcom} &= \frac{1}{N}\sum_{n=1}^{N}\left[y(n) - \hat{y}_{wcom}\right]^2 = \frac{1}{N}\sum_n \left[y(n) - \sum_k \alpha_k \hat{y}_k(n)\right]^2 \\
&= \frac{1}{N}\sum_n \left[\sum_k \alpha_k e_k(n)\right]^2 \\
&= \sum_k \alpha_k^2 \left[\frac{1}{N}\sum_n e_k^2(n)\right] + \sum_k \sum_{j\neq k} \alpha_j \alpha_k \left[\frac{1}{N}\sum_n e_k(n) e_j(n)\right] \\
&= \sum_k \alpha_k^2 \mathrm{MSE}_k + \sum_k \sum_{j\neq k} \alpha_j \alpha_k \left[\frac{1}{N}\sum_n e_k(n) e_j(n)\right], && (10.21)
\end{aligned}
$$

where we have used the fact that $\sum_{k=1}^{K}\alpha_k = 1$, and $e_k(n) = y(n) - \hat{y}_k(n)$.

If the models have uncorrelated errors then $\sum_{j\neq k} \frac{1}{N} \sum_n e_k(n)e_j(n) \approx 0$ and we get

$$\mathrm{MSE}_{wcom} \approx \sum_{k=1}^{K} \alpha_k^2 \mathrm{MSE}_k = \frac{1}{\sum_{k=1}^{K} \mathrm{MSE}_k^{-1}} \leq \min_k[\mathrm{MSE}_k]. \qquad (10.22)$$

10.2.4 Pruning

"Pruning" has its origin in the artificial intelligence method decision trees (hence the name). The idea is to cut away those connections (branches) in the tree that are insignificant, or less significant. This method also applies to neural networks: We can remove connections (weights) that are insignificant. The significance of a weight is measured by its "saliency", which is a measure of how the error (test or training) changes if the weight is removed. Different pruning methods compute the saliency differently.

Weight decay The simplest neural network pruning method is to use weight decay, which is discussed above under regularization, and remove all weights that have a magnitude less than some cut off threshold. That is, the saliency of the weight w_i is $|w_i|$. This was common about ten years ago. However, this is a very *ad hoc* method – there is little reason to expect small weights to be unneccessary. What really matters is how the output is influenced by removing the weight.

Optimal brain surgeon A more well-founded pruning method than cutting away small weights is the "optimal brain surgeon" (OBS), which is discussed e.g. in (Bishop 1995) (OBS is actually equal to the classical statistical Wold test). It is based on a second order Taylor expansion to estimate the effect of removing a weight. Suppose the network is in a local minima, i.e. a point $\mathbf{w}$ where $\nabla_w E(\mathbf{w}) = 0$. The second order expansion around this location is

$$\begin{aligned} E(\mathbf{w}+\delta\mathbf{w}) &= E(\mathbf{w}) + \delta\mathbf{w}^T \nabla_w E(\mathbf{w}) + \frac{1}{2}\delta\mathbf{w}^T H(\mathbf{w})\delta\mathbf{w} + \mathcal{O}\left(\|\delta\mathbf{w}\|^3\right) \\ &= E(\mathbf{w}) + \frac{1}{2}\delta\mathbf{w}^T H(\mathbf{w})\delta\mathbf{w} + \mathcal{O}\left(\|\delta\mathbf{w}\|^3\right), \qquad (10.23) \end{aligned}$$

where $H(\mathbf{w})$ is the Hessian evaluated in location $\mathbf{w}$.

To compute the effect of removing weight w_i we require that $(\delta\mathbf{w})_i = \delta w_i = -w_i$. This means that $(\mathbf{w}+\delta\mathbf{w})_i = 0$. It is more convenient to use a vector/matrix notation, so we introduce the vector

$$\mathbf{1}_i = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}, \tag{10.24}$$

i.e. the vector $\mathbf{1}_i$ has zeros everywhere except in position i where there is a one. This allows us to write our requirement as

$$\delta\mathbf{w}^T\mathbf{1}_i = -w_i. \tag{10.25}$$

The saliency for weight w_i is

$$\begin{aligned} S_i = \Delta E_i = E(\mathbf{w}+\delta\mathbf{w}) - E(\mathbf{w}) &= \frac{1}{2}\delta\mathbf{w}^T H(\mathbf{w})\delta\mathbf{w} && (10.26) \\ \text{where } \delta\mathbf{w}^T\mathbf{1}_i &= -w_i, && (10.27) \end{aligned}$$

which can be computed using the method of Lagrange multipliers. We simply add the constraint (10.27), weighted by a Lagrange multiplier λ, to expression (10.26). (Note that λ here has nothing to do with the regularization parameter discussed erlier.)

This gives us the following cost to minimze

$$C(\delta\mathbf{w}) = \frac{1}{2}\delta\mathbf{w}^T H(\mathbf{w})\delta\mathbf{w} + \lambda\left(\delta\mathbf{w}^T\mathbf{1}_i + w_i\right). \tag{10.28}$$

The extremum conditions are

$$\begin{aligned} \frac{\partial C(\delta\mathbf{w})}{\partial\lambda} = 0 \quad &\Rightarrow \quad \delta\mathbf{w}^T\mathbf{1}_i + w_i = 0, && (10.29) \\ \nabla_{\delta w} C(\delta\mathbf{w}) = 0 \quad &\Rightarrow \quad H(\mathbf{w})\delta\mathbf{w} + \lambda\mathbf{1}_i = 0. && (10.30) \end{aligned}$$

Expression (10.29) is just our constraint. The second equation, (10.30), gives us

$$\delta\mathbf{w} = -\lambda H^{-1}(\mathbf{w})\mathbf{1}_i \tag{10.31}$$

as a solution for $\delta\mathbf{w}$. All that is left is to determine λ, which we do by multiplying (10.31) from the left with $\mathbf{1}_i^T$ and using the constraint (10.29). This gives

$$\left.\begin{array}{l} \mathbf{1}_i^T\delta\mathbf{w} = -\lambda\mathbf{1}_i^T H^{-1}(\mathbf{w})\mathbf{1}_i \\ \mathbf{1}_i^T\delta\mathbf{w} = -w_i \end{array}\right\} \Rightarrow \lambda = \frac{w_i}{\mathbf{1}_i^T H^{-1}(\mathbf{w})\mathbf{1}_i} = \frac{w_i}{[H^{-1}(\mathbf{w})]_{ii}}. \tag{10.32}$$

Plugging this into (10.31) gives the weight step

$$\delta \mathbf{w} = \frac{-w_i H^{-1}(\mathbf{w}) \mathbf{1}_i}{[H^{-1}(\mathbf{w})]_{ii}} \tag{10.33}$$

that fulfills the constraint. This yields the saliency

$$S_i = \frac{w_i^2}{2[H^{-1}(\mathbf{w})]_{ii}}. \tag{10.34}$$

It is important to note that $[H(\mathbf{w})]_{ii}^{-1} \neq [H^{-1}(\mathbf{w})]_{ii}$. The former is the diagonal element ii of the Hessian, to the power of -1, whereas the latter is the diagonal element ii of the inverse Hessian.

Equation (10.34) is the core of OBS. The OBS algorithm is

1. Train to a minimum, i.e. a location $\mathbf{w}$ where $\nabla E(\mathbf{w}) = \mathbf{0}$.
2. Repeat until the validation MSE starts to increase:
   - 2.1 Remove the weight with the smallest saliency (10.34), and adjust the weights according to the corresponding $\delta \mathbf{w}$.
   - 2.2 Retrain to a minimum.

The main problem with the OBS technique is the overhead cost with computing the Hessian. The Hessian and its inverse is therefore approximately updated iteratively to reduce the computational cost. This is desribed in e.g. Haykin's book (Haykin 1999).

Sensitivity based pruning for variable selection Another pruning method, specifically designed for pruning input units, is the "sensitivity based pruning" (SBP), e.g. (Moody, Utans, Rehfuss & Siegelmann 1995). The idea behind SBP is to estimate the effect of an input on the MSE. This is done in (Moody *et al.* 1995) by comparing the validation MSE from the network when all inputs are included to the MSE from the network when one of the inputs is replaced by its mean value. The saliency of the input is the difference between these two MSE values, i.e.

$$S_k = \text{MSE}_k - \text{MSE} \tag{10.35}$$

where MSE_k denotes the MSE when input k is replaced by its mean value. If the saliency is low, or even negative, then the input k is removed and the network retrained with the fewer variables.

10.2.5 Growing algorithms

Pruning algorithms start out with a large model and then shrink it. This is a costly way of doing things since large models take longer time to train. A growing algorithm, on the other hand, starts out with a small model and increases it, which is usually more time efficient.

There have been many automatic algorithms suggested for growing – Bishop (Bishop 1995) lists a few (including Cascade Correlation). However, my experience and impression is that none of the automatic algorithms really work satisfactorily (they overfit too much). The only method that really works, and which should always be performed, is the iterative increase of hidden units with repeated evaluation of the validation error.

The procedure is really simple:

1. Start out with two hidden units (one hidden unit is equivalent to linear regression, or logistic regression, depending on how the transfer function is chosen).
2. Repeat until the validation error starts to increase:
    - 2.1 Train the network (this could be with or without early stopping).
    - 2.2 Evaluate the MSE over the validation data.
    - 2.3 Increase the number of hidden units.

This can also be done with slightly more advanced search than just increasing the number of hidden units every time.

References

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

Haykin, S. (1999), *Neural Networks – A Comprehensive Foundation*, second edn, Prentice Hall Inc., Upper Saddle River, New Jersey.

Moody, J., Utans, J., Rehfuss, S. & Siegelmann, H. (1995), Input variable selection for neural networks: Application to predicting the U.S. business cycle, *in* 'Proc. of the IEEE/IAFE Conference on Computational Intelligence for Financial Engineering, (CIFer)', New York.

Prechelt, L. (1998), Early stopping – but when?, *in* 'Neural Networks: Tricks of the Trade', Springer-Verlag, Berlin, pp. 55–70.

Tikhonov, A. N. & Arsenin, V. Y. (1977), *Solutions of Ill-Posed problems*, V. H. Winston & Sons, Washington D.C.