Improving Generalization

Introduction to Neural Networks : Lecture 10

1. Improving Generalization
2. Training, Validation and Testing Data Sets
3. Cross-Validation
4. Weight Restriction and Weight Sharing
5. Stopping Training Early
6. Regularization and Weight Decay
7. Adding Noise / Jittering
8. Which is the Best Approach ?

Improving Generalization

We have seen that, for our networks to generalize well, we need to avoid both under-fitting of the training data (which corresponds to high statistical bias) and over-fitting of the training data (which corresponds to high statistical variance).

There are a number of approaches for improving generalization – we can:

1. Arrange to have the optimum number of free parameters (independent connection weights) in our model.
2. Stop the gradient descent training at the appropriate point.
3. Add a regularization term to the error function to smooth out the mappings that are learnt.
4. Add noise to the training patterns to smooth out the data points.

To employ these effectively we need to be able to estimate from our training data what the generalization is likely be.

Training, Validation and Testing Data Sets

We have talked about ***training data*** sets – the data used for training our networks. The ***testing data*** set is the unseen data that is used to test the network's generalization.

We usually want to optimize our network's training procedures to result in the best generalization, but using the testing data to do this would clearly be cheating. What we can do is assume that the training data and testing data are drawn randomly from the same data set, and then any sub-set of the training data that we do not train the network on can be used to estimate what the performance on the testing set will be, i.e. what the generalization will be. The portion of the data we have available for training that is withheld from the network training is called the ***validation data*** set, and the remainder of the data is called the ***training data*** set. This approach is called the ***hold out method***.

The idea is that we split the available data into training and validation sets, train various networks using the training set, test each one on the validation set, and the network which is best is likely to provide the best generalization to the testing set.

Cross Validation

Often the availability of training data is limited, and using part of it as a validation set is not practical. An alternative is to use the procedure of ***cross-validation***.

In ***K-fold cross-validation*** we divide all the training data at random into K distinct subsets, train the network using K–1 subsets, and test the network on the remaining subset. The process of training and testing is then repeated for each of the K possible choices of the subset omitted from the training. The average performance on the K omitted subsets is then our estimate of the generalization performance.

This procedure has the advantage that is allows us to use a high proportion of the available training data (a fraction $1-{}^{1}/_{K}$) for training, while making use of all the data points in estimating the generalization error. The disadvantage is that we need to train the network K times. Typically $K \sim 10$ is considered reasonable.

If K is made equal to the full sample size, it is called ***leave-one-out cross validation***.

Weight Restriction and Weight Sharing

Perhaps the most obvious way to prevent over-fitting in our models (i.e. neural networks) is to restrict the number of free parameters they have.

The simplest way we can do this is to restrict the number of hidden units, as this will automatically reduce the number of weights. We can use some form of validation or cross-validation scheme to find the best number for each given problem.

An alternative is to have many weights in the network, but constrain certain groups of them to be equal. If there are symmetries in the problem, we can enforce ***hard weight sharing*** by building them into the network in advance. In other problems we can use ***soft weight sharing*** where sets of weights are encouraged to have similar values by the learning algorithm.

One way to do this is to add an appropriate term to the error/cost function. This method can then be seen as a particular form of ***regularization***.

Stopping Training Early

Neural networks are often set up with more than enough parameters for over-fitting to occur, and so other procedures have to be employed to prevent it.

For the iterative gradient descent based network training procedures we have considered (such as batch back-propagation and conjugate gradients), the training set error will naturally decrease with increasing numbers of epochs of training.

The error on the unseen validation and testing data sets, however, will start off decreasing as the under-fitting is reduced, but then it will eventually begin to increase again as over-fitting occurs.

The natural solution to get the best generalization, i.e. the lowest error on the test set, is to use the procedure of ***early stopping***. One simply trains the network on the training set until the error on the validation set starts rising again, and then stops. That is the point at which we expect the generalization error to start rising as well.

Practical Aspects of Stopping Early

One potential problem with the idea of stopping early is that the validation error may go up and down numerous times during training. The safest approach is generally to train to convergence (or at least until it is clear that the validation error is unlikely to fall again), saving the weights at each epoch, and then go back to weights at the epoch with the lowest validation error.

There is an approximate relationship between stopping early and a particular form of regularization which indicates that it will work best if the training starts off with very small random initial weights.

There are general practical problems concerning how to best split the available training data into distinct training and validation data sets. For example: What fraction of the patterns should be in the validation set? Should the data be split randomly, or by some systematic algorithm? As so often, these issues are very problem dependent and there are no simple general answers.

Regularization

The general technique of ***regularization*** encourages smoother network mappings by adding a penalty term Ω to the standard (e.g. sum squared error) cost function

$$E_{reg} = E_{sse} + \lambda\Omega$$

where the regularization parameter λ controls the trade-off between reducing the error E_{sse} and increasing the smoothing. This modifies the gradient descent weight updates so

$$\Delta w_{kl}^{(m)} = -\eta \frac{\partial E_{sse}(w_{ij}^{(n)})}{\partial w_{kl}^{(m)}} - \eta\lambda \frac{\partial \Omega(w_{ij}^{(n)})}{\partial w_{kl}^{(m)}}$$

For example, for the case of soft weight sharing we can use the regularization function

$$\Omega = -\sum_{k,l,m} \ln\left(\sum_{j=1}^{M} \frac{\alpha_j}{\sqrt{2\pi\sigma_j^2}} \exp\left\{ -\frac{(w_{kl}^{(m)} - \mu_j)^2}{2\sigma_j^2} \right\} \right)$$

in which the $3M$ parameters α_j, μ_j, σ_j are optimised along with the weights.

Weight Decay

One of the simplest forms of regularization has a regularization function which is just the sum of the squares of the network weights (not including the thresholds):

$$\Omega = -\tfrac{1}{2} \sum_{k,l,m} (w_{kl}^{(m)})^2$$

In conventional curve fitting this regularizer is known as ***ridge regression***. We can see why it is called ***weight decay*** when we observe the extra term in the weight updates:

$$-\eta\lambda \frac{\partial\Omega(w_{ij}^{(n)})}{\partial w_{kl}^{(m)}} = -\eta\lambda w_{kl}^{(m)}$$

In each epoch the weights decay in proportion to their size, i.e. exponentially.

Empirically, this leads to significant improvements in generalization. This is because producing over-fitted mappings requires high curvature and hence large weights. Weight decay keeps the weights small and hence the mappings are smooth.

Adding Noise / Jittering

Adding ***noise*** or ***jitter*** to the inputs during training is also found empirically to improve network generalization. This is because the noise will 'smear out' each data point and make it difficult for the network to fit the individual data points precisely, and consequently reduce over-fitting.

Actually, if the added noise is ξ with variance λ, the error function can be written:

$$E_{sse}(\xi_i) = \tfrac{1}{2}\sum_p \sum_\xi \sum_j \left(y_j^p - net_j(x_i^p + \xi_i)\right)^2$$

and after some tricky mathematics one can show that

$$\lambda\Omega = E_{sse}(\xi_i) - E_{sse}(0) = \tfrac{1}{2}\lambda \sum_p \sum_j \left(\frac{\partial net_j(x_k^p)}{\partial x_i^p} \right)^2$$

which is a standard ***Tikhonov regularizer*** minimising curvature. The gradient descent weight updates can then be performed with an extended back-propagation algorithm.

Which is the Best Approach ?

To get an optimal balance between bias and variance, we need to find a way of optimising the effective complexity of the model, which for neural networks means optimising the effective number of adaptable parameters.

One way to optimise that number is to use the appropriate number of connection weights, which can easily be adjusted by changing the number of hidden units. The other approaches we have looked at can all be seen to be effectively some form of regularization, which involves adding a penalty term to the standard gradient descent error function. The degree of regularization, and hence the effective complexity of the model, is controlled by adjusting the regularization scale λ.

In practice, we find the best number of hidden units, or degree of regularization, using a validation data set or the procedure of cross-validation. The approaches all work well, and which we choose will ultimately depend on which is most convenient for the particular problem in hand. Unfortunately, there is no overall best approach!

Overview and Reading

1. We began by recalling the aim of good generalization.
2. Then the ideas of validation and cross-validation were introduced as convenient methods for estimating generalization using only the available training data.
3. We then went through the main approaches for improving generalization: limiting the number of weights, weight sharing, stopping training early, regularization, weight decay, and adding noise to the inputs.
4. As usual, we concluded that there was no generally optimal approach.

Reading

1. Bishop: Sections 9.2, 9.3, 9.4, 9.8
2. Haykin: Sections 4.12, 4.14
3. Gurney: Section 6.10