Bias and Variance, Under-Fitting and Over-Fitting

Neural Computation : Lecture 9

1. The Computational Power of MLPs
2. Learning and Generalization Revisited
3. Statistical View of Neural Network Learning
4. Bias and Variance Decomposition
5. Under-fitting, Over-fitting and the Bias/Variance Trade-off
6. Preventing Under-fitting and Over-fitting

Computational Power of MLPs

The ***Universal Approximation Theorem*** can be stated as:

Let $\varphi(\cdot)$ be a non-constant, bounded, and monotone-increasing continuous function. Then for any continuous function $f(\mathbf{x})$ with $\mathbf{x} = \{x_i \in [0,1] : i = 1, \ldots, m\}$ and $\varepsilon > 0$, there exists an integer M and real constants $\{\alpha_j, b_j, w_{jk} : j = 1, \ldots, M, k = 1, \ldots, m\}$ such that

$$F(x_1, \ldots, x_m) = \sum_{j=1}^{M} \alpha_j \varphi\left(\sum_{k=1}^{m} w_{jk} x_k - b_j \right)$$

is an approximate realisation of $f(\cdot)$, that is

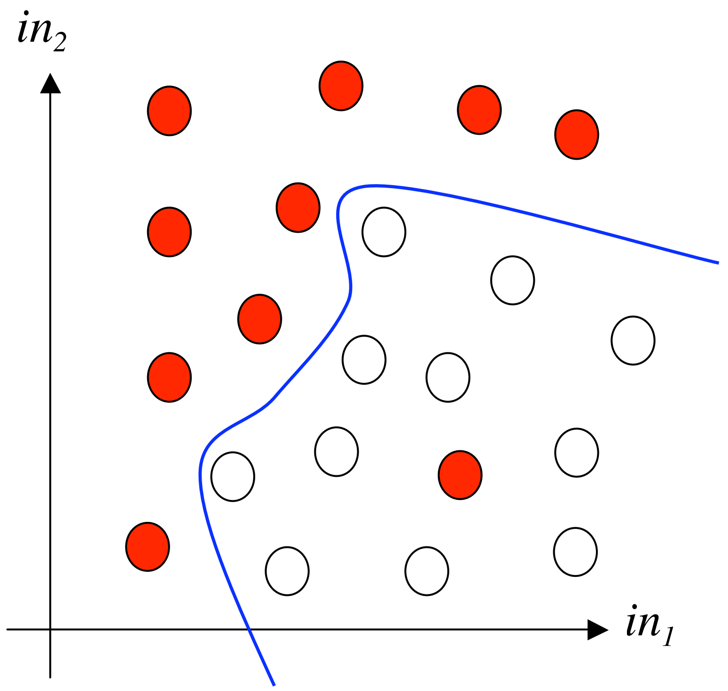
$$\left| F(x_1, \ldots, x_m) - f(x_1, \ldots, x_m) \right| < \varepsilon$$

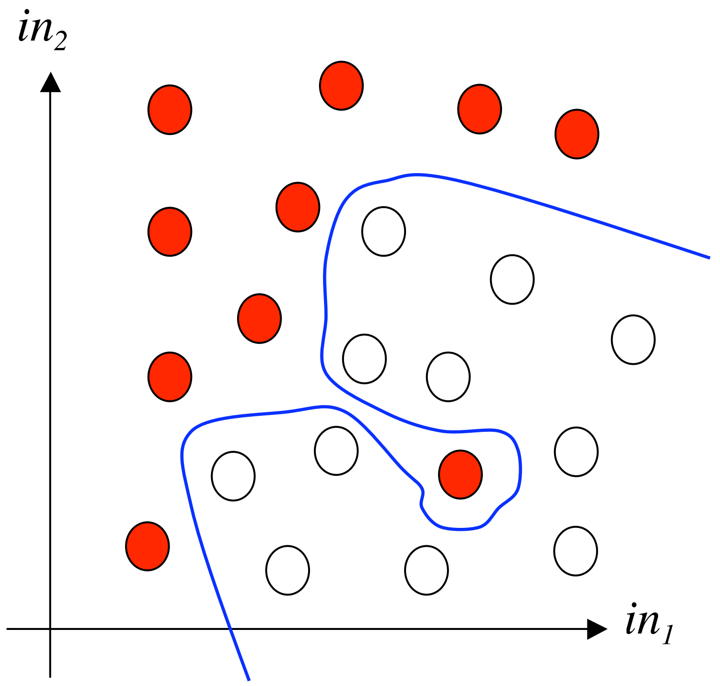
for all $\mathbf{x}$ that lie in the input space.

Clearly this applies to an MLP with M hidden units, since $\varphi(\cdot)$ can be a sigmoid, w_{jk}, b_j can be hidden layer weights and biases, and α_j can be output weights. It follows that, given enough hidden units, a two layer MLP can approximate any continuous function.

Learning and Generalization Revisited

Recall the idea of getting a neural network to learn a classification decision boundary:

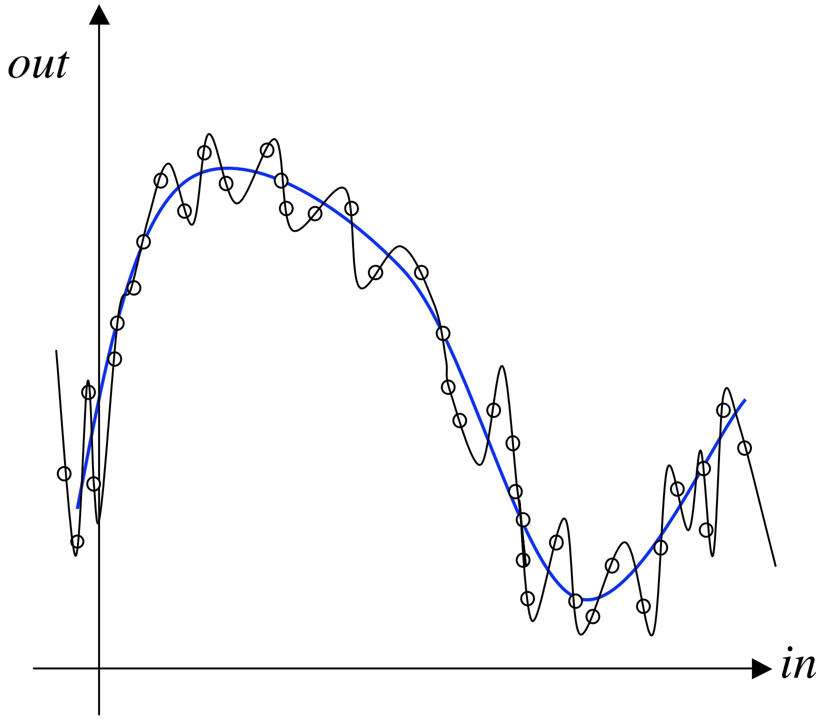




The aim is to get the network to generalize to classify new inputs appropriately. If the training data is known to contain noise, we don't necessarily want the training data to be classified totally accurately, because that is likely to reduce the generalization ability.

Generalization in Function Approximation

Similarly if our network is required to recover an underlying function from noisy data:



There are good reasons to expect the network to give a more accurate generalization to new inputs if its output curve does not pass through all the data points. Again, allowing a larger error on the training data is likely to lead to better generalization.

Generalization in General

It is helpful to think about what we are asking our neural networks to cope with when they generalize to deal with unseen input data. There are a few recurring factors:

1. Empirically determined data points will usually contain a certain level of noise, e.g. incorrect class labels or measured values that are inaccurate.

2. In most cases, the underlying "correct" decision boundary or function will be smoother than that indicated by a given set of noisy training data.

3. If we had an infinite number of data points, the errors and inaccuracies would be easier to spot and tend to cancel out of averages.

4. Different sets of training data, i.e. different sub-sets of the infinite set of all possible training data, will lead to different network weights and outputs.

The key question is: how can we recover the best smooth underlying function or decision boundary from a given set of noisy training data?

A Statistical View of the Training Data

To understand generalization from a theoretical point of view, it needs to be considered using a rigorous statistical approach.

In the same way that the frequentist approach to probability takes a finite set of measurements from an infinite set of possible measurements to estimate a probability, we take a finite set of data points to train our neural networks.

Suppose we have a ***training data set*** *D* for our neural network:

$$D = \left\{ \; x_i^p ,\, y^p : i = 1 \;\ldots\; ninputs,\; p = 1 \;\ldots\; npatterns \; \right\}$$

consisting of an output y^p for each input pattern x_i^p. To keep the notation simple, we assume that we only have one output unit – the extension to many outputs is obvious.

What we need to do is find a way to understand statistically the generalization obtained using such a data set, and then optimize the generalization process.

A Regressive Model of the Data

Generally, the training data will be generated by some actual function $g(x_i)$ plus random noise ε^p (which may, for example, be due to data gathering errors), so

$$y^p = g(x_i^p) + \varepsilon^p$$

We call this a ***regressive model*** of the data. We can define a statistical expectation operator $\mathcal{E}$ that averages over all possible training patterns, so the random errors cancel out leaving just the regression function

$$g(x_i) = \mathcal{E}[y \mid x_i]$$

which is the *conditional mean of the model output y given the inputs* x_i.

We will only have one particular set D of training patterns, and the network output derived from that is likely to deviate from $g(x_i)$. What we want to do is find the best possible approximation to $g(x_i)$. That will depend on the set D we have, so we want an approach that gives the best possible approximation averaged over all possible sets D.

A Statistical View of Network Training

The neural network training problem is to construct an output function $net(x_i, W, D)$ of a set of network weights $W = \{w_{ij}^{(n)}\}$, based on the data D, that best approximates $g(x_i)$.

We have seen how to train a network by minimising the sum-squared error cost function:

$$E(W) = \tfrac{1}{2} \sum_{p \in D} \left(y^p - net(x_i^p, W, D) \right)^2$$

with respect to the network weights $W = \{w_{ij}^{(n)}\}$. However, we have also observed that, to get good generalization, we do not necessarily want to achieve that minimum. What we really want to do is minimise the difference between the network's outputs $net(x_i, W, D)$ and the underlying function $g(x_i) = \mathcal{E}[y \mid x_i]$.

The natural sum-squared error function, i.e. $\left(\mathcal{E}[y \mid x_i] - net(x_i, W, D) \right)^2$, depends on the specific training set D, and so we really want our network training regime to produce the best results when averaged over all possible noisy training sets D.

Computing the Expected Generalization Error

If we define the expectation or average operator $\mathcal{E}_D$ which takes the ***ensemble average*** over all possible training sets D, then the expected generalization error is

$$\mathcal{E}_D\left[\left(\mathcal{E}[y \mid x_i] - net(x_i, W, D)\right)^2\right]$$

$$= \mathcal{E}_D\left[\left(\left(\mathcal{E}[y \mid x_i] - \mathcal{E}_D\left(net(x_i, W, D)\right)\right) + \left(\mathcal{E}_D\left(net(x_i, W, D)\right) - net(x_i, W, D)\right)\right)^2\right]$$

$$= \mathcal{E}_D\left[\begin{array}{l}\left(\mathcal{E}[y \mid x_i] - \mathcal{E}_D\left(net(x_i, W, D)\right)\right)^2 + \left(\mathcal{E}_D\left(net(x_i, W, D)\right) - net(x_i, W, D)\right)^2 \\ +2\left(\mathcal{E}[y \mid x_i] - \mathcal{E}_D\left(net(x_i, W, D)\right)\right)\left(\mathcal{E}_D\left(net(x_i, W, D)\right) - net(x_i, W, D)\right)\end{array}\right]$$

$$= \left(\mathcal{E}[y \mid x_i] - \mathcal{E}_D\left(net(x_i, W, D)\right)\right)^2 + \mathcal{E}_D\left(\mathcal{E}_D\left(net(x_i, W, D)\right) - net(x_i, W, D)\right)^2$$
$$+ 2\left(\mathcal{E}[y \mid x_i] - \mathcal{E}_D\left(net(x_i, W, D)\right)\right)\left(\mathcal{E}_D\left(net(x_i, W, D)\right) - \mathcal{E}_D\left(net(x_i, W, D)\right)\right)$$

Bias and Variance

Thus the expected generalization error can be written as:

$$\mathcal{E}_D\left[\left(\mathcal{E}[y \mid x_i] - net(x_i, W, D)\right)^2\right]$$

$$= \left(\mathcal{E}_D\left[net(x_i, W, D)\right] - \mathcal{E}[y \mid x_i]\right)^2 + \mathcal{E}_D\left[\left(net(x_i, W, D) - \mathcal{E}_D\left[net(x_i, W, D)\right]\right)^2\right]$$

$$= \qquad (\text{bias})^2 \qquad + \qquad (\text{variance})$$

which consists of two positive components:

(bias)² the squared difference between the average network output $\mathcal{E}_D[net(x_i, W, D)]$ and the regression function $g(x_i) = \mathcal{E}[y \mid x_i]$. This can be viewed as a measure of the average network ***approximation error*** over all possible training data sets *D*.

(variance) the variance of the approximating function $net(x_i, W, D)$ over all the training sets *D*. It represents the ***sensitivity*** of the results on the particular choice of data *D*.

In practice, there will always be a trade-off between these two error components.

Extreme Case of Bias and Variance – Under-fitting

A good way to understand the concepts of ***bias*** and ***variance*** is by considering the two extreme cases of what a neural network might learn.

Suppose the neural network is lazy and just produces the same constant output whatever training data we give it, i.e. $net(x_i, W, D) = c$. Then we have generalization error:

$$\mathcal{E}_D\left[\left(\mathcal{E}[y \mid x_i] - net(x_i, W, D)\right)^2\right]$$

$$= \left(\mathcal{E}_D[c] - \mathcal{E}[y \mid x_i]\right)^2 + \mathcal{E}_D\left[\left(c - \mathcal{E}_D[c]\right)^2\right]$$

$$= \quad (c - g(x_i))^2 \quad + \quad 0$$

$$= \quad (\text{bias})^2 \quad + \quad (\text{variance})$$

In this case the variance term will be zero, but the bias will be large, because the network has made no attempt to fit the data. We say we have extreme ***under-fitting***.

Extreme Case of Bias and Variance – Over-fitting

On the other hand, suppose the neural network is very hard working and makes sure that it exactly fits every data point. The average network output is then:

$$\mathcal{E}_D\left[net(x_i,W,D)\right] = \mathcal{E}_D\left[y(x_i)\right] = \mathcal{E}_D\left[g(x_i)+\varepsilon\right] = g(x_i) = \mathcal{E}[y\,|\,x_i]$$

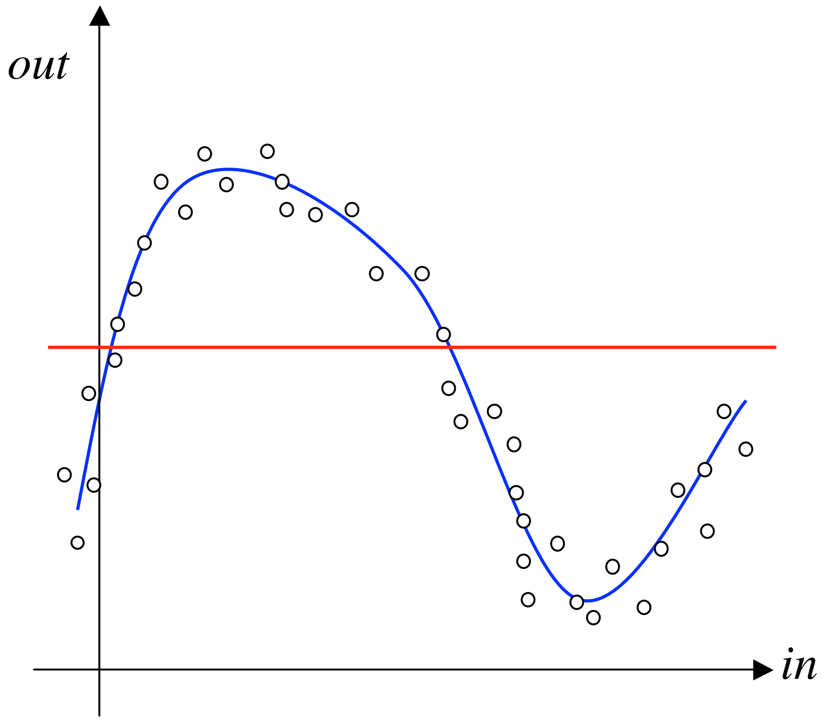
i.e. the regression function. Thus we have the generalization error:

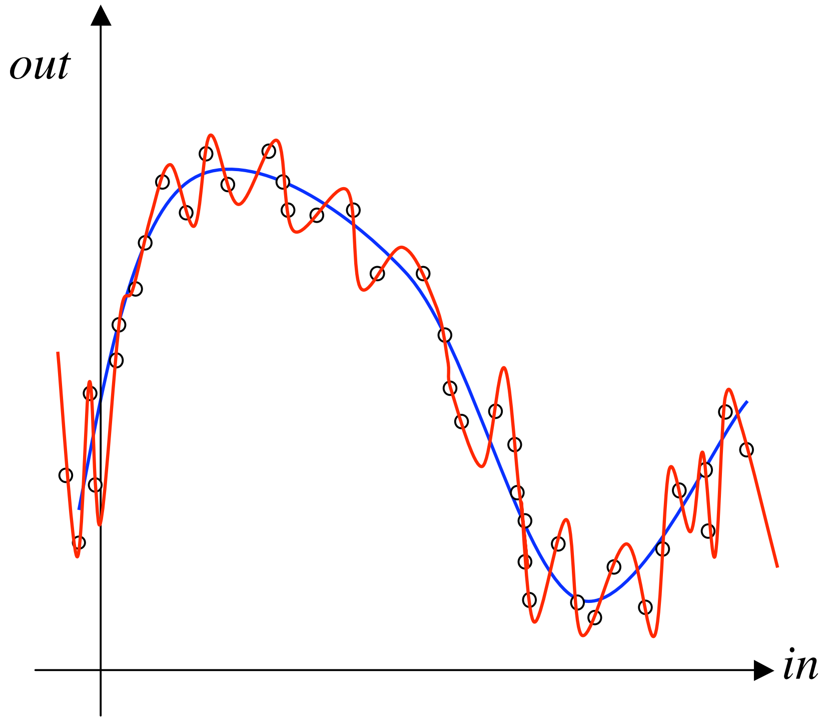
$$\begin{aligned}
&\mathcal{E}_D\left[\left(\mathcal{E}[y\,|\,x_i] - net(x_i,W,D)\right)^2\right] \\
&\quad = \left(\mathcal{E}[y\,|\,x_i] - \mathcal{E}[y\,|\,x_i]\right)^2 + \mathcal{E}_D\left[\left(\mathcal{E}[y\,|\,x_i] + \varepsilon - \mathcal{E}[y\,|\,x_i]\right)^2\right] \\
&\quad = \qquad 0 \qquad + \qquad \mathcal{E}_D\left[\varepsilon^2\right] \\
&\quad = \qquad (\text{bias})^2 \qquad + \qquad (\text{variance})
\end{aligned}$$

so the bias is zero, but the variance is the square of the noise on the data, which could be substantial. In this case we say we have extreme ***over-fitting***.

Examples of the Two Extreme Cases

The lazy and hard-working networks approach the function approximation as follows:





Ignore the data ⇒
  Big approximation error (high bias)
  No variation between data sets (no variance)

Fit every data point ⇒
  No approximation error (zero bias)
  Variation between data sets (high variance)

The Bias/Variance Trade-off

If a network is to generalize well to new data, it obviously needs to generate a good approximation to the underlying function $g(x_i) = \mathcal{E}[y \mid x_i]$, and to do that it must minimise the sum of the bias and variance contributions to the generalization error. There will be a ***trade-off*** between minimising the bias and minimising the variance.

Networks which are too closely fitted to the data will tend to have a large variance and hence give rise to a large expected generalization error. Consequently, we then say that ***over-fitting*** of the training data has occurred.

We can easily decrease the variance by smoothing the network outputs, but if this is taken too far, then the bias becomes large, and the expected generalization error is large again. We then say that ***under-fitting*** of the training data has occurred.

This trade-off between bias and variance plays a crucial role in the application of neural network techniques to practical applications.

Preventing Under-fitting and Over-fitting

To ***prevent under-fitting*** we need to make sure that:

1. The network has enough hidden units to represent the required mappings.
2. The network is trained for long enough that the error/cost function (e.g., SSE or Cross Entropy) is sufficiently minimised.

To ***prevent over-fitting*** we have several options:

1. Restrict the number of adjustable parameters the network has – e.g. by reducing the number of hidden units, or by forcing connections to share the same weight values.
2. Stop the training early – before it has had time to learn the training data too well.
3. Add some form of *regularization* term to the error/cost function to encourage smoother network mappings.
4. Add noise to the training patterns to smear out the data points.

Next lecture we will look in detail at all these approaches to improving generalization.

Overview and Reading

1. We began by looking at the computational power of MLPs.
2. Then we saw why the generalization is often better if we don’t train the network all the way to the minimum of its error function.
3. A statistical treatment of learning showed that there was a trade-off between bias and variance.
4. Both under-fitting (resulting in high bias) and over-fitting (resulting in high variance) will result in poor generalization.
5. There are many ways we can try to improve generalization.

Reading

1. Bishop: Sections 6.1, 9.1
2. Haykin-2009: Sections 2.7, 4.11, 4.12
3. Gurney: Sections 6.8, 6.9