Bias and Variance, Under-Fitting and Over-Fitting

Introduction to Neural Networks : Lecture 9

1. The Computational Power of MLPs
2. Learning and Generalization Revisited
3. A Statistical View of Network Training
4. Bias and Variance
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

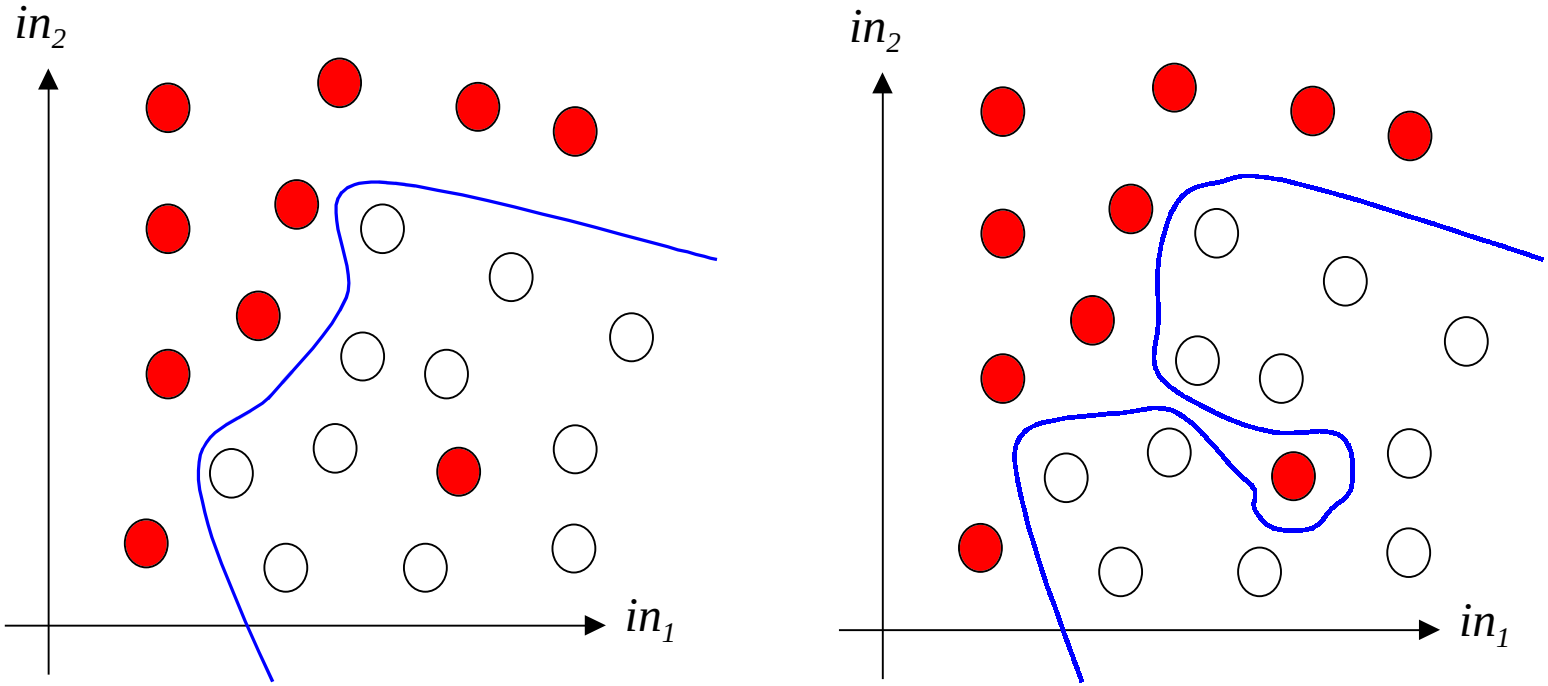
$$\left| F(x_1, \ldots, x_m) - f(x_1, \ldots, x_m) \right| < \varepsilon$$

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

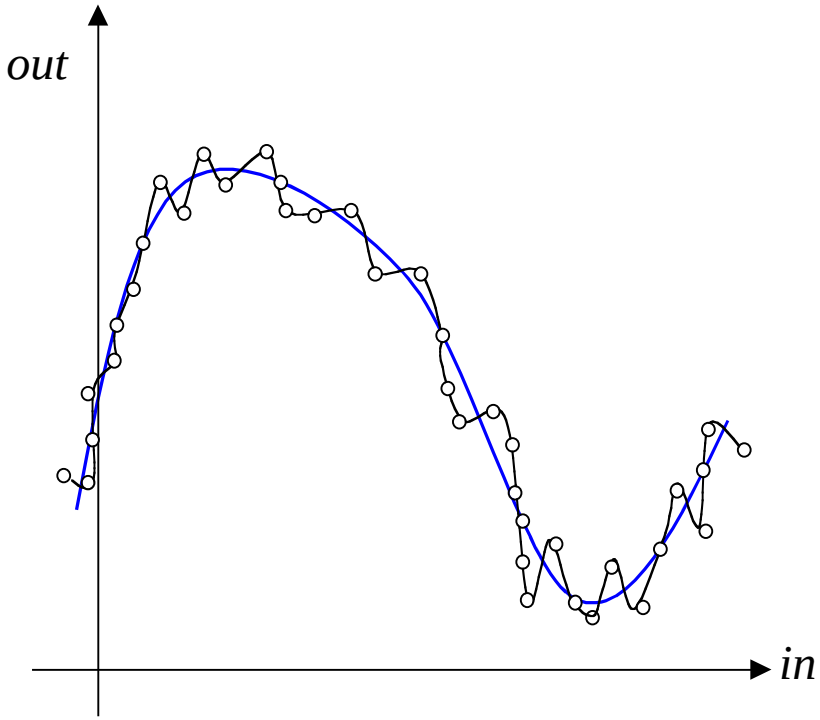
Recall the idea of getting a neural network to learn a classification decision boundary:



Our aim is for the network to generalize to classify new inputs appropriately. If we know that the training data contains noise, we don't necessarily want the training data to be classified totally accurately as that is likely to reduce the generalisation ability.

Generalization in Function Approximation

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



We can 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.

A Statistical View of the Training Data

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

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

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

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

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

We say that the regression function $g(x_i)$ is the *conditional mean of the model output y given the inputs* x_i.

A Statistical View of Network Training

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

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 generalisation, 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 we really want our network training regime to produce good results averaged over all possible noisy training sets.

Bias and Variance

If we define the expectation or average operator $\mathcal{E}_D$ which takes the ***ensemble average*** over all possible training sets D, then some rather messy algebra allows us to show that:

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

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

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

This error function consists of two positive components:

(bias)^2 the 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 the ***approximation error***.

(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.

The Extreme Cases of Bias and Variance

We can best understand the concepts of ***bias*** and ***variance*** by considering the two extreme cases of what the network might learn.

Suppose our network is lazy and just generates the same constant output whatever training data we give it, i.e. $net(x_i, W, D) = c$. 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.

Suppose our network is very hard working and makes sure that it fits every data point:

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

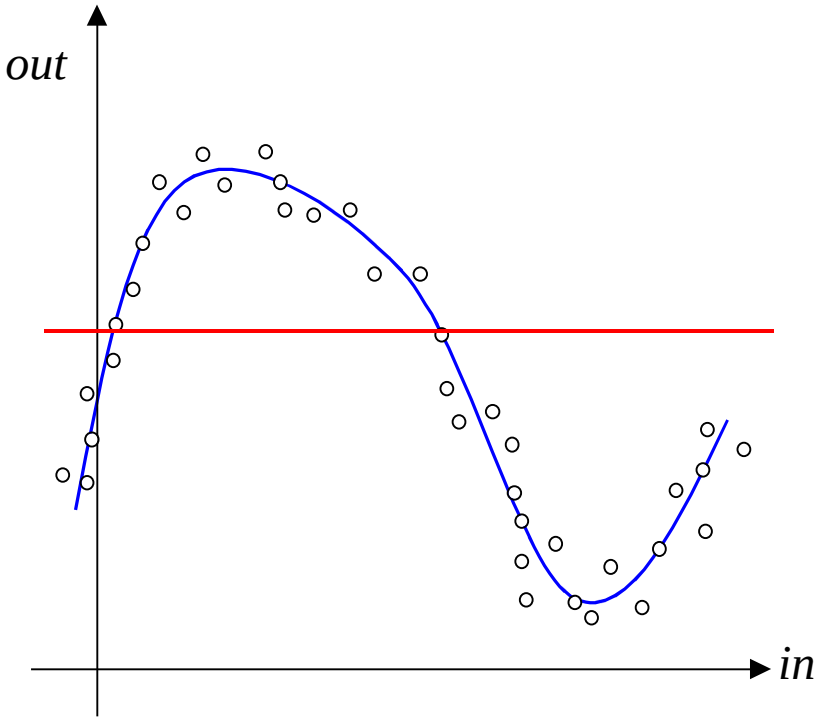
so the bias is zero, but the variance is:

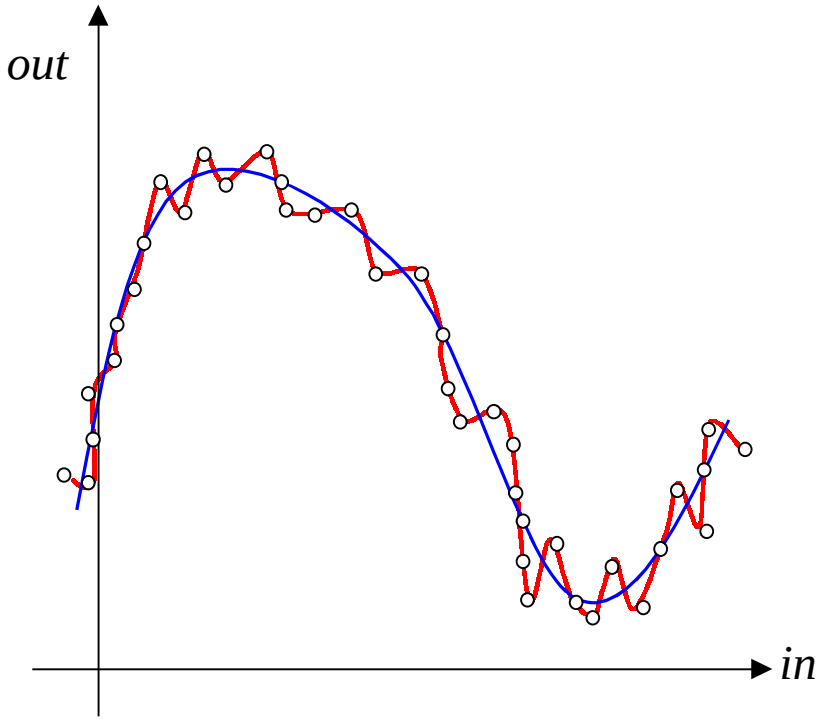
$$\mathcal{E}_D\left[\left(net(x_i, W, D) - \mathcal{E}_D\left[net(x_i, W, D)\right]\right)^2\right] = \mathcal{E}_D\left[\left(g(x_i) + \varepsilon - \mathcal{E}_D\left[g(x_i) + \varepsilon\right]\right)^2\right] = \mathcal{E}_D\left[(\varepsilon)^2\right]$$

i.e. the variance of the noise on the data, which could be substantial.

Examples of the Two Extreme Cases

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





Ignore the data ⇒

- Big approximation errors (high bias)
- No variation between data sets (no variance)

Get every data point ⇒

- No approximation errors (zero bias)
- Variation between data sets (high variance)

Under-fitting, Over-fitting and the Bias/Variance Trade-off

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

A network which is too closely fitted to the data will tend to have a large variance and hence give a large expected generalization error. 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 to required mappings.
2. We train the network for long enough so that the sum squared error cost function is sufficiently minimised.

To ***prevent over-fitting*** we can:

1. Stop the training early – before it has had time to learn the training data too well.
2. 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.
3. Add some form of *regularization* term to the error function to encourage smoother network mappings.
4. Add noise to the training patterns to smear out the data points.

Next lecture will be dedicated to looking at 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 (giving high bias) and over-fitting (giving high variance) will result in poor generalization.
5. The are many ways we can try to improve generalization.

Reading

1. Bishop: Sections 6.1, 9.1, 9.2
2. Gurney: Sections 6.8, 6.9
3. Haykin: Sections 2.13, 4.13