24 Case Study 2: Probability Estimation

Objectives

This chapter represents the second of a series of case studies with neural networks. The previous chapter demonstrated the use of neural networks for function approximation. In this chapter we use a neural network to estimate a probability function.

Probability estimation is a special case of function approximation. In function approximation we want the neural network to map between a set of input variables and a set of response variables. However, in the case of probability estimation the response variables correspond to a set of probabilities. Since probabilities have certain special properties — they must always be positive, and they must sum to 1 — we want the neural network to enforce these conditions.

In the case study we consider in this chapter, the system in question is chemical vapor deposition of diamond. A carbon dimer (a bound pair of carbon atoms) is projected toward a diamond surface. We want to determine the probabilities for various reactions based on characteristics of the projected dimer. The input variables consist of such properties as translational energy and incidence angle, and the response variables consist of the probabilities of the potential reactions, such as chemisorption and scattering.

Theory and Examples

This chapter presents a case study in using neural networks for probability estimation. Probability estimation consists of determining the probabilities of certain events, based on a set of input variables. For example, we might want to know the probabilities associated with a patient having a certain disease, based on a set of laboratory tests. Another example would be determining the probability of a financial instrument going up in price, based on a set of market conditions.

CVD

For this probability estimation case study, we will train a neural network to estimate reaction rates in a chemical process. Chemical vapor deposition (CVD) of diamond is a process for making synthetic diamond. The idea is to cause carbon atoms in a gas to settle on a substrate in crystalline form. In order to study this process, scientists are often interested in reaction rates, which will determine how quickly the diamond can be created. In this case study, we will train a neural network to compute reaction rates as a carbon dimer (a bound pair of carbon atoms) interacts with the crystalline diamond substrate.

We will begin by describing the CVD process and how simulated data can be collected for this process. Then, we will show how a neural network can be trained to learn the reaction probabilities. The details of the procedure are described in [AgSa05].

Description of the CVD Process

During the CVD process, a carbon dimer is projected toward a diamond substrate. For the purpose of this study, we will assume that the dimer can react with the substrate in one of three ways: chemisorption (the atoms in the dimer become bound to the substrate), scattering (the atoms bounce off the substrate), or desorption (the atoms become bound to the substrate for a period of time, but are then released). There is another possible reaction that occurs with very small probability, but we will ignore it for this study. (See [AgSa05] for a full discussion.) We will train a neural network to estimate the probabilities of each of the reactions, based on various characteristics of the carbon dimer, which will be described below.

The notation we will use to define this interaction is illustrated in Figure 24.1. The black circle represents the carbon dimer, and the corresponding directed line represents the direction of the initial velocity vector. The blue star represents the location of the central carbon atom in the diamond substrate. The angle θ denotes the angle of incidence, i.e., the angle between the direction of the initial velocity vector of the carbon dimer and the perpendicular on the surface (the z direction). The impact parameter b is defined as the distance between the location of the central atom and the point of intersection of the initial velocity vector and the diamond surface (indi-

cated by the origin of the axes in Figure 24.1). The angle ϕ represents the angle between the x axis and the line from the origin to the central atom.

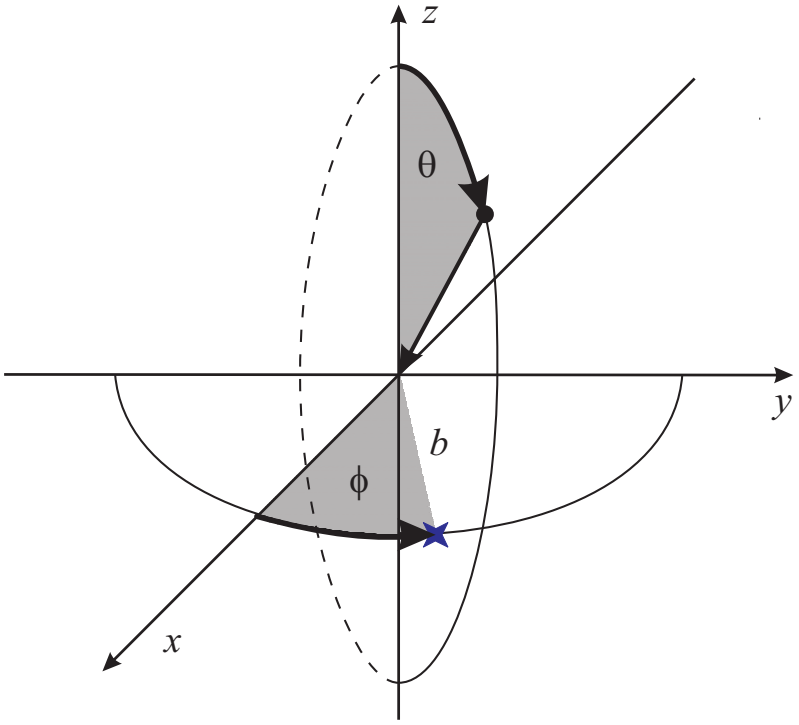

Figure 24.1 Notation for Carbon Dimer/Diamond Substrate Interaction

Data Collection and Preprocessing

Data for training the neural network are obtained by molecular dynamics (MD) simulations. In MD, the motion of atoms and molecules in a material under a given force are simulated, using known laws of physics to calculate the forces on individual atoms [RaMa05]. For this case study, we use a total of 324 atoms to model the CVD system. Out of these, 282 atoms of diamond substrate are used to model the crystalline face with 40 atoms of hydrogen on the top layer of the diamond surface, and 2 atoms in the C_2 dimer. In Figure 24.1, the (x,y) plane represents the location of the diamond substrate. Each of the carbon atoms on the top layer of the substrate, except the central atom and boundary atoms, is capped by a hydrogen atom. Any reactions will occur near the central atom.

For this study, we want to determine the dependence of probabilities for chemisorption, scattering, and desorption on b, θ, ϕ, rotational velocity (v_{rot}), and translational velocity (v_{trans}) of the C_2 dimer. The initial C_2 vibrational energy is set equal to the zero-point energy and the temperature of the lattice is maintained constant at 600 K [RaMa05].

This is an interesting function approximation problem, in that we don't have access to the true underlying reaction probabilities, which are un-

known. We will obtain estimates of these probabilities by running Monte Carlo simulation experiments. We will need some notation to help us keep track of the various probabilities that we will work with. First, we will indicate the true underlying reaction probabilities by $P_X(\mathbf{p})$, where X refers to the reaction process, and $\mathbf{p}$ is the vector that characterizes the C_2 dimer:

$$\mathbf{p} = \begin{bmatrix} \theta \\ \phi \\ b \\ v_{\text{trans}} \\ v_{\text{rot}} \end{bmatrix}. \tag{24.1}$$

The reaction process can be chemisorption ($X = C$), scattering ($X = S$), or desorption ($X = D$). The probability estimates produced by the neural network will be indicated by $P_X^{NN}(\mathbf{p})$. The probability estimates obtained from the Monte Carlo simulations will be indicated by $P_X^{MC}(\mathbf{p})$.

The Monte Carlo estimates are obtained by

$$P_X^{MC}(\mathbf{p}) = \frac{N_X}{N_T}, \tag{24.2}$$

where N_X is the number of MD trajectories that resulted in reaction X and N_T is the total number of trajectories computed. The results of a given trajectory depend upon a multitude of input variables. These include the parameters included in $\mathbf{p}$, as well as the initial orientation of the C_2 dimer, the angle defining the C_2 rotational plane, the initial C_2 vibrational energy and its phase, the temperature of the system, and all of the variables that define the vibrational phases of the diamond surface. Because we are only interested in the effect of $\mathbf{p}$ on the reaction probabilities, the other variables are randomly set for each MD simulation, except that the initial C_2 vibrational energy is set equal to the zero-point energy and the temperature of the lattice is maintained constant at 600 K. Eq. (24.2) averages over the trajectories to estimate the underlying true probabilities $P_X(\mathbf{p})$. (As a note of clarification here, we use the term Monte Carlo to refer to the set of simulations that are obtained by setting a number of the variables to random values for each trajectory. We refer to the simulation of a single trajectory as an MD simulation, since the principles of molecular dynamics are used to perform the computations.)

This is a standard method used by chemists to estimate reaction probabilities. If they want to determine the effect of ϕ, for example, on the probabilities, they must run a series of Monte Carlo simulations at each value of ϕ that is of interest. This can be extremely time consuming. The required number of Monte Carlo trials can be quite large, if an accurate reaction probability is required. Our objective in this case study is to train a neural

network to learn the true reaction probabilities as a function of the parameters in $\mathbf{p}$.

To train a neural network, we need a set of target outputs. Since we do not know the true underlying probabilities $P_X(\mathbf{p})$, we will use the estimates obtained from the Monte Carlo simulations $P_X^{MC}(\mathbf{p})$. We can think of these Monte Carlo probabilities as being noisy versions of the true probabilities. The neural network will need to interpolate these noisy values to produce an accurate estimate of $P_X(\mathbf{p})$ without overfitting. This is a good application for our generalization procedures, which were discussed in Chapter 13.

The data set consists of 2000 different $\{\mathbf{p}, P_X^{MC}(\mathbf{p})\}$ input/target pairs. Out of these 2000 data points, 1400 (70%) were randomly selected for training, 300 (15%) for validation, and 300 for testing. For each trajectory, the $\mathbf{p}$ were generated randomly, using physically-appropriate distributions for each variable [RaMa05]. A total of $N_T = 50$ different trajectories were run to obtain each $P_X^{MC}(\mathbf{p})$. This means that 2000x50 trajectories were run to create the entire data set.

The original units of the inputs are radians for ϕ and θ, angstroms for b, angstroms per picosecond for v_{trans} and radians per picosecond for v_{rot}. Before presenting the input data to the network for training, they are scaled using Eq. (22.1), so that each element of the input vector ranges from -1 to 1. The targets have values that are always in the range 0 to 1, since they represent probabilities. In the next section, we will describe a network architecture, in which the softmax transfer function of Eq. (22.3) is used in the final layer. This transfer function produces outputs that range from 0 to 1, so the original unscaled targets will work fine.

Selecting the Architecture

We will use a multilayer network for this application. We know that there will be five elements in the input vector, which is defined in Eq. (24.1). The target for the network can be a vector with three elements:

$$\mathbf{t} = \begin{bmatrix} P_C^{MC}(\mathbf{p}) \\ P_S^{MC}(\mathbf{p}) \\ P_D^{MC}(\mathbf{p}) \end{bmatrix}, \tag{24.3}$$

or we can use three different networks, each with a different $P_X^{MC}(\mathbf{p})$ as a target. We have tried both possibilities, and the results are similar.

In this case, there is an advantage to using the single network, with three elements in the output vector. The three targets represent probabilities. Therefore, they are always in the range 0 to 1, and they always sum to 1.

This is an ideal situation for using the softmax transfer function of Eq. (22.3), which is repeated here:

$$a_i = f(n_i) = \exp(n_i) \div \sum_{j=1}^{S} \exp(n_j). \tag{24.4}$$

This transfer function is different from others we have used, in that each neuron output a_i is affected by all of the net inputs n_j. (In the other transfer functions, the net input n_i affected only the neuron output a_i.) This does not cause any substantial difficulties in network training. The backpropagation algorithm of Eq. (11.44) and Eq. (11.45) can still be used to compute the gradient. However, the derivative of the transfer function is no longer a diagonal matrix. The derivative of the softmax function has the following form:

$$\dot{\mathbf{F}}^m(\mathbf{n}^m) = \begin{bmatrix} a_1^m\left(\sum_{i=1}^{S^m} a_i^m - a_1^m\right) & -a_1^m a_2^m & \cdots & -a_1^m a_{S_m}^m \\ -a_2^m a_1^m & a_2^m\left(\sum_{i=1}^{S^m} a_i^m - a_2^m\right) & \cdots & -a_2^m a_{S_m}^m \\ \vdots & \vdots & & \vdots \\ -a_{S_m}^m a_1^m & -a_{S_m}^m a_2^m & \cdots & a_{S_m}^m\left(\sum_{i=1}^{S^m} a_i^m - a_{S_m}^m\right) \end{bmatrix} \tag{24.5}$$

The complete network architecture is shown in Figure 24.2. The input vector of Eq. (24.1) has 5 elements. The output vector has 3 elements, which is consistent with the target vector of Eq. (24.3). The transfer function in the hidden layer is the hyperbolic tangent sigmoid, and the softmax transfer function is used in the output layer. The number of neurons in the hidden layer, S^1, is yet to be determined. It depends on the complexity of the function that we are trying to approximate, but we do not know at this point how complex the function is. In general, the size of the hidden layer must be determined as part of the training process. We must choose S^1 so that the network provides an accurate fit to the training data, without overfitting. We will discuss this selection in the next section.

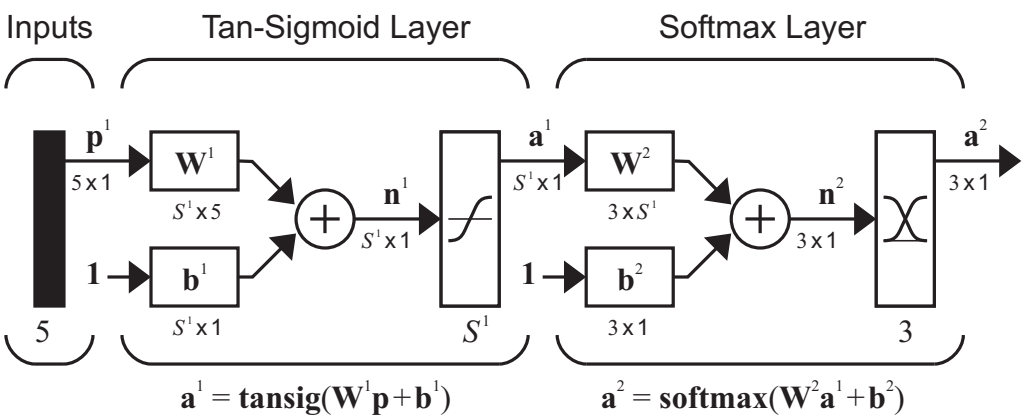

Figure 24.2 Network Architecture

Training the Network

We trained the network using the scaled conjugate gradient algorithm of [Mill93]. Many other conjugate gradient or Levenberg-Marquardt algorithms, such as those discussed in Chapter 12, would have also worked well. The targets for this problem have a significant amount of noise, so we are not expecting extreme accuracy in the final fit. We used early stopping, as described in Chapter 13, to prevent overfitting. We stopped the training if the error on the validation set failed to improve over 25 iterations. A typical training session is illustrated in Figure 24.3, which shows training and validation MSE. The minimum of the validation performance was reached at iteration 69. The algorithm continued for 25 more iterations, until iteration 94. Since the validation error was not reduced during those 25 iterations, the weights from iteration 69 were saved as the final trained values.

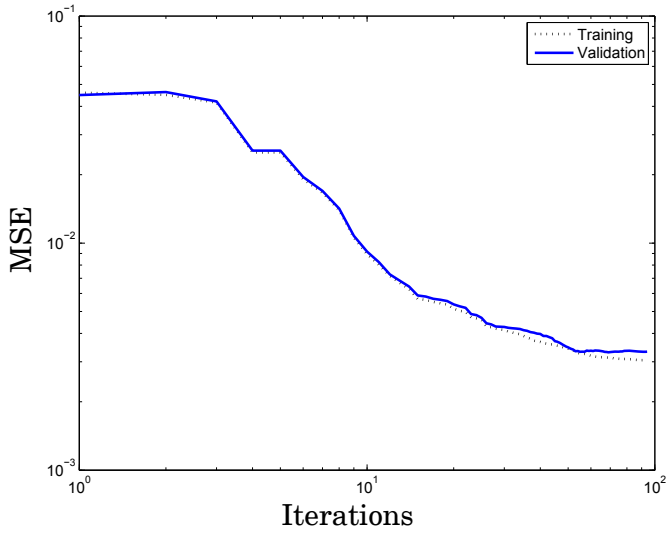

Figure 24.3 Training and Validation Mean Square Error ($S^1 = 10$)

The results shown in Figure 24.3 represent a network with 10 neurons in the hidden layer ($S^1 = 10$). We need to verify that this is a reasonable number. One indicator is a comparison of training and validation performance. Table 24.1. shows the training and validation root mean square error (RMSE) for the trained network. We can see that training and validation errors are roughly the same. The validation data was randomly selected and was selected independently of the training set. Because the errors were approximately the same on both sets, it appears that the network fit is consistent throughout the relevant input range, and no overfitting occurs.

	Training RMSE	Validation RMSE
$P_C(\mathbf{p})$	0.0496	0.0439
$P_S(\mathbf{p})$	0.0634	0.0659
$P_D(\mathbf{p})$	0.0586	0.0604

Table 24.1. Comparison of Training and Validation RMSE for $S^1 = 10$

It is also important to determine if the errors are as small as possible and if the fit is adequate. We will have more to say about that in the next section, but at this point we can try fitting networks with different numbers of hidden neurons. Table 24.2. shows the results of fitting a network with two hidden neurons. Again, the training and validation errors are consistent, which indicates lack of overfitting, but the errors are higher than those for $S^1 = 10$.

	Training RMSE	Validation RMSE
$P_C(\mathbf{p})$	0.0634	0.0627
$P_S(\mathbf{p})$	0.0669	0.0704
$P_D(\mathbf{p})$	0.0617	0.0618

Table 24.2. Comparison of Training and Validation RMSE for $S^1 = 2$

Table 24.3. shows the results for $S^1 = 20$. The validation error is slightly higher than the training error, which might indicate some overfitting. The main point is that neither training nor validation errors are significantly smaller for $S^1 = 20$ than for $S^1 = 10$. This indicates that ten hidden neurons are sufficient for this problem. We will investigate this further in the next section.

	Training RMSE	Validation RMSE
$P_C(\mathbf{p})$	0.0432	0.0444
$P_S(\mathbf{p})$	0.0603	0.0643
$P_D(\mathbf{p})$	0.0569	0.0595

Table 24.3. Comparison of Training and Validation RMSE for $S^1 = 20$

There is one further step that we want to make as part of the training process. We want to ensure that we have not fallen into a local minimum. For this reason, we want to retrain the network several times, using different initial weights and biases. (We use the Nguyen-Widrow initialization method described in Chapter 22.) Table 24.4. shows the final validation MSE for each of five different training runs. We can see that all of the errors are similar, so we have reached a global minimum at each run. If one error was significantly lower than the others, then we would use the weights that obtained the lowest error.

3.074e-003	2.953e-003	3.031e-003	3.105e-003	3.050e-003

Table 24.4. Final Validation MSE for Five Different Initial Conditions

We have determined that a neural network with ten neurons in the hidden layer produces a reasonable response without overfitting. The next step is to analyze the performance of the network. Depending on the results of that analysis, we might adjust the network architecture or training data and retrain the network.

Validation

An important tool for network validation is a scatter plot of network outputs versus targets, as shown in Figure 24.4. Here we can see that there is a strong linear relationship between the targets and the network outputs, but there appears to be quite a bit of variation. We might expect that for a well trained network the points in the scatter plot would fall exactly on the outputs=target line. Why do we have so much variation in this plot? The reason is that the targets of the network are not the true reaction probabilities, $P_X(\mathbf{p})$, but the Monte Carlo estimates $P_X^{MC}(\mathbf{p})$. There is noise in the targets.

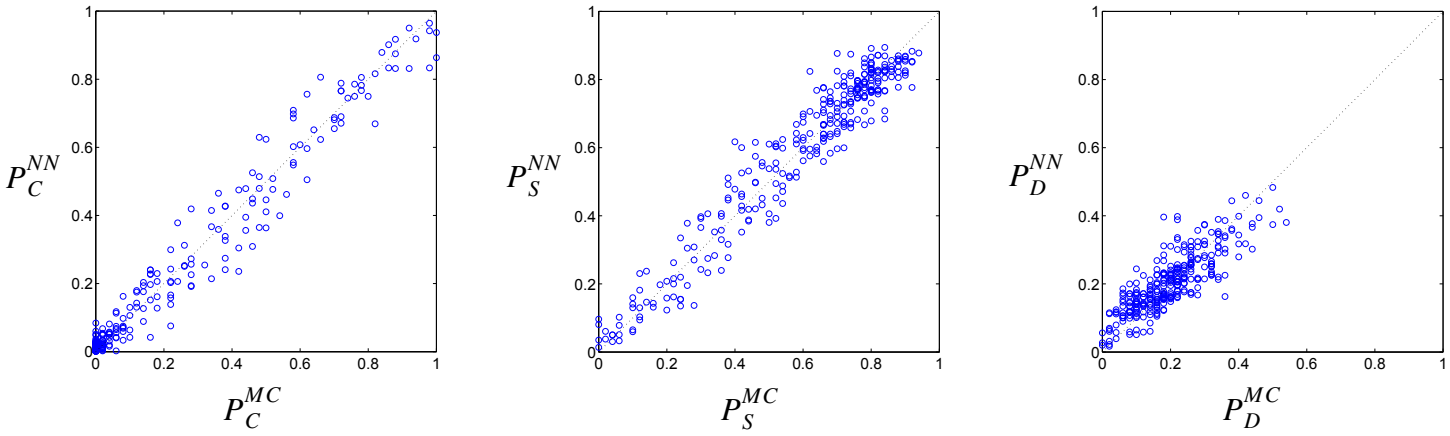

Figure 24.4 Scatter Plot of Network Outputs vs. Targets ($N_T = 50$)

The relationship between P_X^{MC} and P_X is such that ~95% of the time we expect to have

$$P_X - 2\Delta \le P_X^{MC} \le P_X + 2\Delta\,, \tag{24.6}$$

where

$$\Delta = \sqrt{\frac{P_X\{1 - P_X\}}{N_T}}\,. \tag{24.7}$$

This relationship is illustrated in Figure 24.5 for $N_T = 50$.

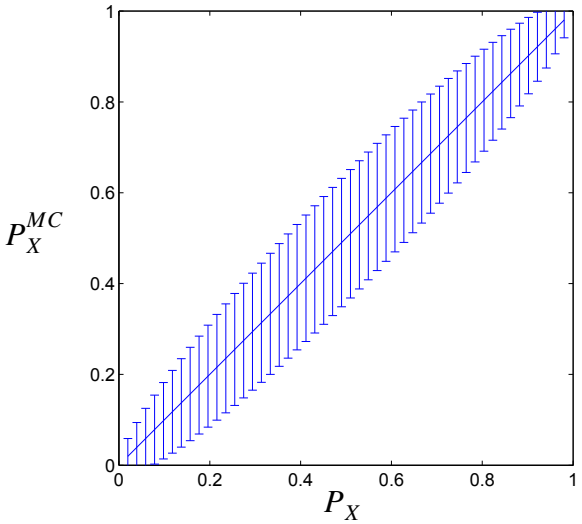

Figure 24.5 Expected Statistical Spread of P_X^{MC} for $N_T = 50$

By comparing Figure 24.4 with Figure 24.5, we can see that the spread in the data is explained by the statistical variations in P_X^{MC}. To further verify this observation, we generated additional testing data, in which 500 Monte Carlo trials were run to obtain each $P_X^{MC}(\mathbf{p})$ (i.e., $N_T = 500$). We applied this testing data to the network that was trained on the original data set with $N_T = 50$. The resulting scatter plots are shown in Figure 24.6. Here we can see that the spread has decreased dramatically from Figure 24.4, even though the network has not changed. This means that the neural network is fitting the true probabilities $P_X(\mathbf{p})$ and not the statistical fluctuations in $P_X^{MC}(\mathbf{p})$.

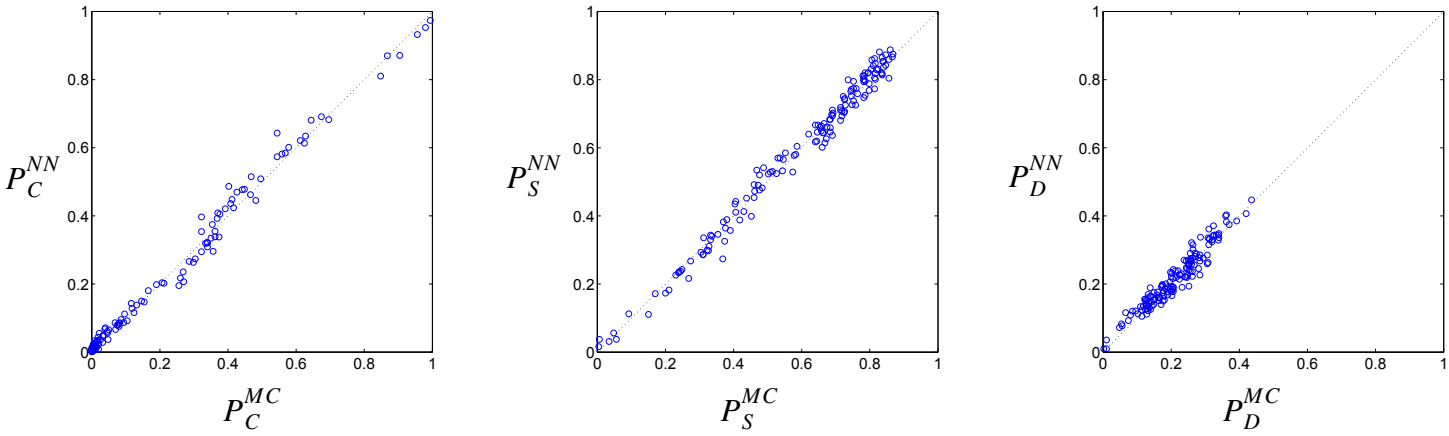

Figure 24.6 Scatter Plot of Network Outputs vs. Targets ($N_T = 500$)

After the neural network has been trained, it becomes a simple matter to investigate the effect of the input parameters on the reaction probabilities. In Figure 24.7, we see the effect of the impact parameter b on the reaction probabilities, as determined by the neural network. As the impact parameter is increased, the probability of chemisorption decreases, while the probabilities of scattering and desorption increase. (For this study, we have set θ to 5.4 radians, ϕ to 0.3 radians, v_{rot} to 0.004 radians per femtosecond, and v_{trans} to 0.004 angstroms per femtosecond.)

With standard methods, a study such as that shown in Figure 24.7 would take thousands of simulations. The trained neural network has fully captured the relationships between the parameters in $\mathbf{p}$ and the reaction probabilities. Therefore, we can perform arbitrary studies by simply computing the network responses at a varying set of input points. Note that the network interpolated smoothly through a noisy set of data points to capture the true underlying function. By using the early stopping technique, we prevented the network from overfitting the noise in the data.

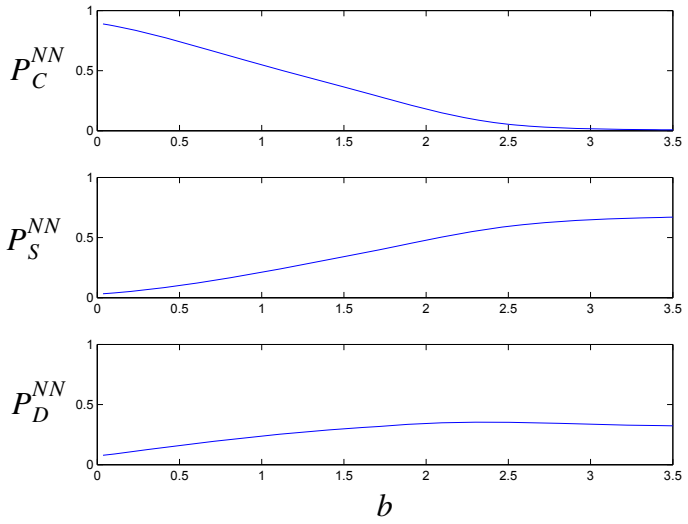

Figure 24.7 Reaction Probabilities vs. Impact Parameter

Data Sets

There are four data files associated with this case study:

- cvd_p.txt — contains the input vectors in the original data set
- cvd_t.txt — contains the target vectors in the original data set
- cvd_p500.txt — contains the input vectors in the $N_T = 500$ test set
- cvd_t500.txt — contains the target vectors in the $N_T = 500$ test set

They can be found with the demonstration software, which is described in Appendix C.

Epilogue

This chapter has illustrated the use of neural networks for probability estimation on a chemical vapor deposition problem. Monte Carlo simulations were used to provide estimates of the reaction probabilities. These estimates were used as targets for the neural network. The network was able to capture the true underlying probability function without overfitting to the errors in the Monte Carlo estimates. This was accomplished by using the early stopping procedure, which stops network training if the error on an independent validation set increases.

In the next chapter, we apply neural networks to a pattern recognition problem. We will also use multilayer neural networks for that application.

Further Reading

[AgSa05] P.M. Agrawal, A.N.A. Samadh, L.M. Raff, M. Hagan, S. T. Bukkapatnam, and R. Komanduri, "Prediction of molecular-dynamics simulation results using feedforward neural networks: Reaction of a C2 dimer with an activated diamond (100) surface," *The Journal of Chemical Physics* 123, 224711, 2005.

This paper describes the details of training a neural network to predict the reaction probabilities for chemical vapor deposition of diamond.

[Mill93] M.F. Miller, "A scaled conjugate gradient algorithm for fast supervised learning," Neural Networks, vol. 6, pp. 525-533, 1993.

The scaled conjugate gradient algorithm is a fast batch training algorithm for neural networks that requires a minimum of memory and computation at each iteration.

[RaMa05] L.M. Raff, M. Malshe, M. Hagan, D.I. Doughan, M.G. Rockley, and R. Komanduri, "*Ab initio* potential-energy surfaces for complex, multi-channel systems using modified novelty sampling and feedforward neural networks," The Journal of Chemical Physics, 122, 084104, 2005.

This paper explains how neural networks can be used for molecular dynamics simulations.