Artificial Neural Networks as Knowledge Aided Design Modules

Independent Study

Reto Zingg

December 15th 2000

University of Colorado at Boulder

Table of Contents

1 Artificial Neural Network

For all applications in this report we used MLP3 ANNs. The basic structure is shown in Figure 1. MLP stands for Multi Layer Perceptron. The three stands for three layers. These three layers are input layer, the hidden layer and the output layer. Each layer is made up by a certain number of neurons. The number of neurons in the hidden layer is free to choose by the user. The number of neurons in the input and output layers correspond to the number of values fed into the ANN respectively delivered by the ANN. Each arrow in Figure 1 has a certain weight. The operation at a neuron can vary. We use Back-propagation as training algorithm (see [1] for details on ANNs).

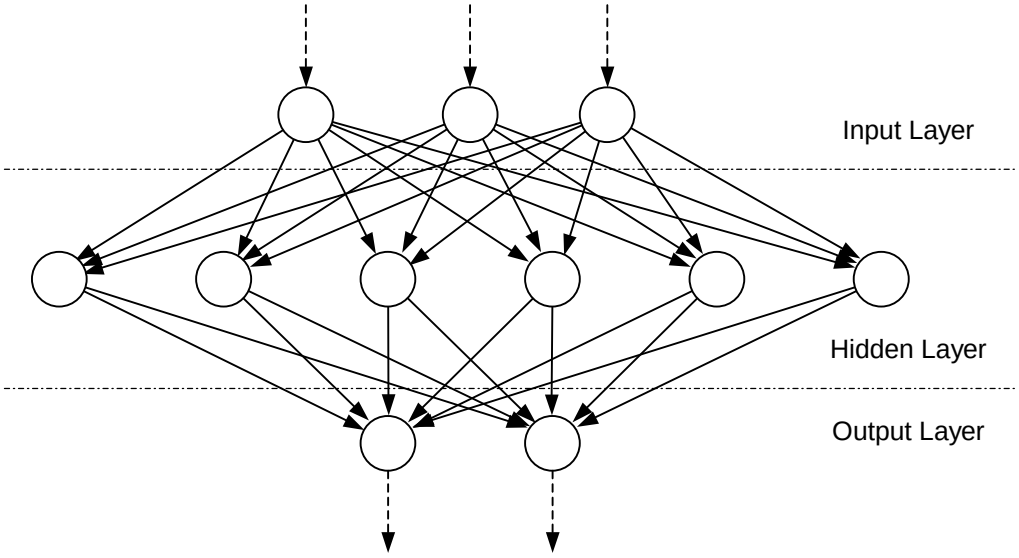

Figure 1 Basic setup of an MLP3 ANN

As an example for using ANNs as KAD modules we have a look at the reflection type phase shifter and the loaded line type phase shifter. Results are shown for these two circuits. Also we have a look at amplifier design and matching networks, where we explore some of the problems and limitations.

Figure 2 shows the naming conventions used in this report to refer to the input respectively output parameters.

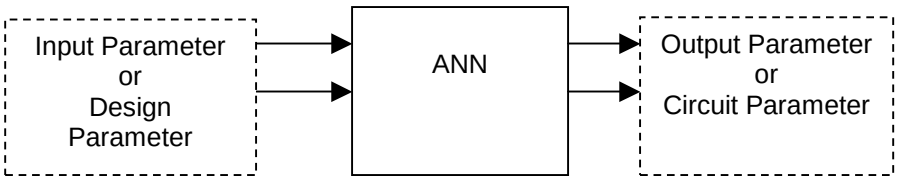

Figure 2 ANN for the reflection type phase shifter

2 PIN diodes

As we will use PIN diodes as switching devices for our examples (phase shifters) we first have a look at them. We use the PIN diodes in two states. A forward bias state and a reverse bias state. We can represent the diode in these two states with the equivalent circuits shown in Figure 3.

Figure 3 (a) Forward and (b) reverse bias equivalent circuit of the PIN diode

R_S represents a parasitic loss resistance. We assume it to be constant. L_f originates from package and bond wire. C_r is due to the junction capacitance of the PIN diode in reverse bias. From these values and at a certain operating frequency we get Z_f and Z_r, the impedances in the two bias states.

Table 1 shows a short survey of Agilent PIN diodes and their equivalent circuit parameters.

Table 1 Short PIN diode survey

PIN Diode Type	Series Resistance [Ohm]		Capacitance [pF]			X_r [jΩ] at	
	Typ.	Max.	Min.	Typ.	Max.	1GHz	5GHz
HPND-4005	4.7	6.5		0.017	0.02	-7957.9	-1591.6
HPND-4018	4.6			0.025		-6366.3	-1273.3
HPND-4028	2.3	3.0	0.025		0.045	-3536.8	-707.4
HPND-4038	1.5	2.0	0.045		0.065	-2448.6	-489.7
HSMP-3800		2.0			0.37	-430.1	-86.0
1N5767		2.0			0.37	-430.1	-86.0
5082-314x		0.08			0.14	-1136.8	-227.4
DiodeChips		0.8			0.16	-994.7	-198.9
HPND-40x8		4.6			0.03	-5305.3	-1061.1
HSMP-382x		0.6			0.8	-198.9	-39.8
HSMP-386x		4.5			0.25	-636.6	-127.3
HSMP-388x		6.5			0.4	-397.8	-79.6
HSMP-389x		2.5			0.3	-530.5	-106.1
HSMP-482x		0.6			0.8	-198.9	-39.8
HSMP-489x		2.5			0.3	-530.5	-106.1

In an application note of Agilent the total inductance of a SOT-23 package is noted to be ≅2nH. At 1 GHz this is 12.6 jΩ and 63 jΩ at 5 GHz.

3 Reflection Type Phase Shifter

In this chapter we will develop an ANN as KAD module for the design of reflection type phase shifters.

3.1 The Circuit

Lets first have a look at the design shown in Figure 4.

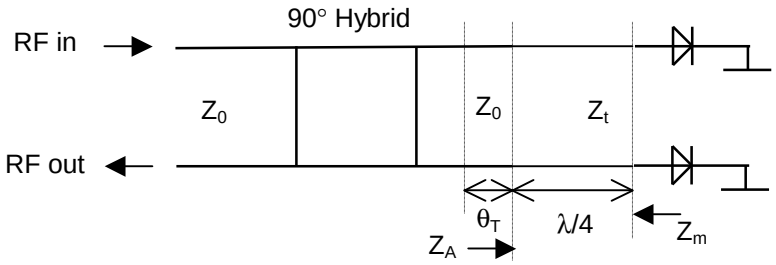

Figure 4 Reflection type phase shifter with PIN diodes and λ/4 transformers

The diodes have the two states of forward bias and reverse bias (biasing circuit not shown). In these two states the diodes have the two reactances X_f and X_r. As a first approximation we neglect loss of the devices (i.e. resistance). X_f originates from the small inductance introduced by packaging and bond wire. This inductance often is not explicitly noted in the data sheet of devices, but can be obtained from the specification of the package. X_r originates in the capacitance of the PIN diode junction in reverse bias. This capacitance is shown in data sheets of devices.

Let's study this circuit, from right to left (starting at the diodes). We want to design a phase shifter with a certain phase shift $\Delta\phi$. To obtain this, the two reflection coefficients at the diodes in the two states (forward and reverse bias) need to have a phase difference of $\Delta\phi$. We can influence the reflection coefficients by changing the impedance Z_m into which the diodes 'look'. As our system, as well as the 90° hybrid are at Z_0 we need an impedance transformer. In this case we use the $\lambda/4$ transformer. Now the two reflection-coefficients have an angle of $\Delta\phi$ in between them (imagine a Smith Chart) and we are at an impedance of Z_0. With no further proof we know that we get the highest bandwidth if the reflection coefficients (seen by the 90° hybrid) lie in the Smith Chart symmetrically on opposite sides of the X=0 line. Therefore we add a length θ_T of transmission line, to 'rotate' the reflection coefficients into the correct position (again, visualize the Smith Chart). Finally the 90° hybrid splits the input and output (reflected) signal.

3.2 Design

Now let's have a look at the design equations [2]. The impedance Z_m into which the diodes have to look in order to create a phase shift of $\Delta\phi$ between the two bias states we can write as:

$$Z_f = Z_m \cdot \frac{Z_r + j \cdot Z_m \cdot \tan\left(\frac{\Delta\phi}{2}\right)}{Z_m + j \cdot Z_r \cdot \tan\left(\frac{\Delta\phi}{2}\right)} \quad \textbf{(1)}$$

Where Z_f is the forward bias impedance and Z_r the reverse bias impedance of the diodes. As Z_m in general is complex, we need an impedance transformer with a variable length. This can be expressed as

$$Z_m = Z_T \cdot \frac{Z_0 + j \cdot Z_T \cdot \tan(\theta_{tr})}{Z_T + j \cdot Z_0 \cdot \tan(\theta_{tr})} \quad \textbf{(2)}$$

where Z_T is the impedance of the transformer and θ_{tr} the electrical length. Numerical results show that the length of the impedance transformer deviates by such small amounts from 90° (this is with typical Z_f and Z_r of PIN diodes) that we can set it constantly to θ_{tr}=90°.

Now the impedances Z_A seen by the Z_0 transmission line into the impedance transformer are:

$$Z_{AF} = \frac{Z_T^{\,2}}{Z_F}$$
$$Z_{AR} = \frac{Z_T^{\,2}}{Z_R} \quad \textbf{(3)}$$

The corresponding reflection coefficients are

$$\Gamma_{AF} = \frac{Z_{AF} - Z_0}{Z_{AF} + Z_0}$$
$$\Gamma_{AR} = \frac{Z_{AR} - Z_0}{Z_{AR} + Z_0} \quad \textbf{(4)}$$

We want the two reflection coefficients seen by the 90° hybrid to be conjugate complexes of each other in order to maximize the bandwidth of the circuit. But we also know that the two reflection coefficients have an angle of $\Delta\phi$ in between them. So we can calculate the electrical transmission line length θ_T as

$$\theta_T = \left(\arg(\Gamma_F) \pm \frac{\Delta\phi}{2} \right) \tag{5}$$

The sign depends on which solution we obtained (there are two solutions to Γ_F). The solution of θ_T is $\pi/4$ periodic.

Solutions for (1) and (2) we find numerically. Taking the left hand side of (1) to the right hand side we get

$$error = Z_m \cdot \frac{Z_r + j \cdot Z_m \cdot \tan\left(\frac{\Delta\phi}{2}\right)}{Z_m + j \cdot Z_r \cdot \tan\left(\frac{\Delta\phi}{2}\right)} - Z_f \tag{6}$$

Plotting the value of that expression depending on the real and imaginary part of Z_m we get a 'feeling' for the solution (see Figure 5).

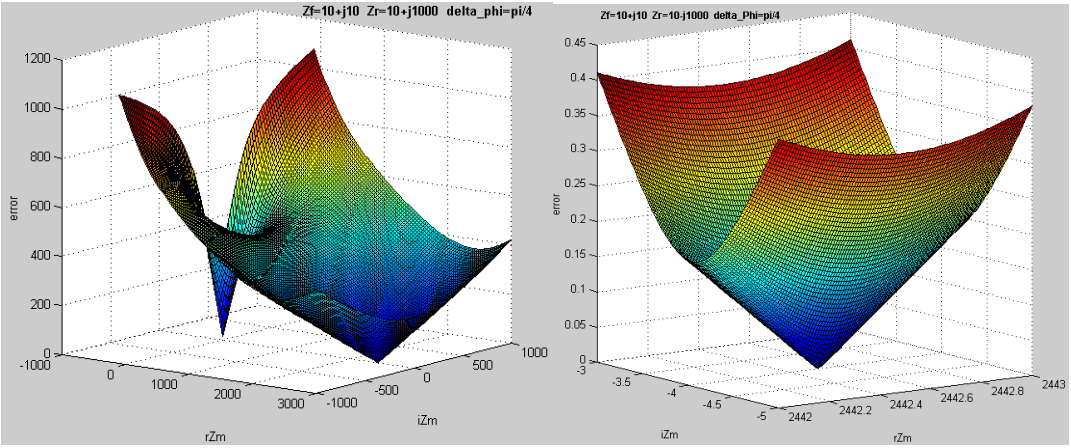

Figure 5 Plot of (6) for Z_f=10+j10, Z_r=10-j1000, $\Delta\phi=\pi/4$

We see there is one solution (the second solution has a negative real part). Details of the numerical solution can be found in the Matlab code in 7.1.1. After obtaining Z_m we can get Z_T and from that θ_T.

3.2.1 Example

Z_f=(10+j10)Ω, Z_r=(10-j1000)Ω, $\Delta\phi=\pi/4$

Then we get from the above analysis:

Z_m=2442.5-j4.046 and from that Z_T=349.5 Ω and θ_T=11.01°

Now let's try to verify these results with an ADS simulation. First we simulate without the lossy element (set the real part of Z_f and Z_r to zero). Figure 6 shows the circuit used for this purpose. Figure 7 shows the result of this simulation. Clearly we can see that we obtained the correct solution.

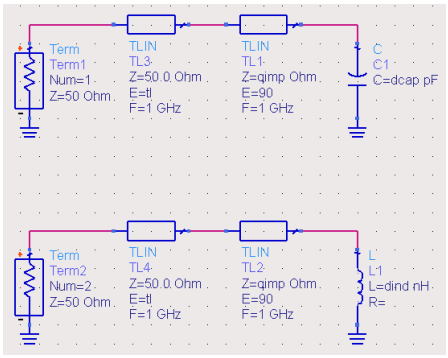

Figure 6 ADS simulation circuit

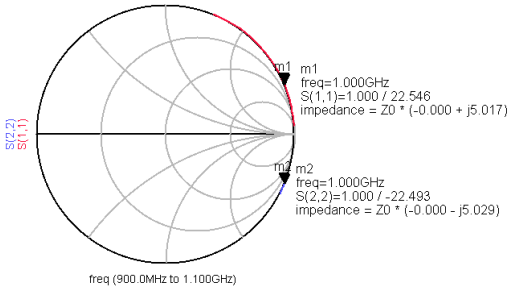

Figure 7 ADS simulation result

Now let's include the lossy element. Figure 8 shows the circuit for this simulation. Figure 9 shows the result of this simulation. Clearly we can see again, that we obtain the correct result.

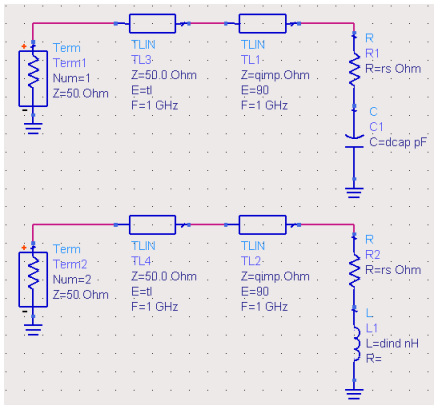

Figure 8 ADS circuit with lossy element

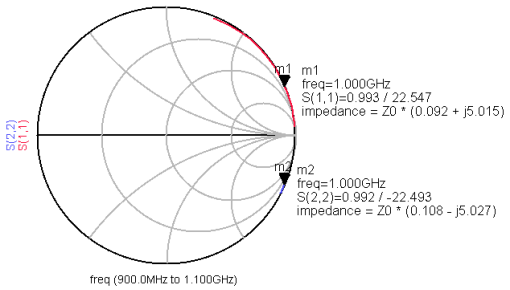

Figure 9 ADS simulation result with lossy element

3.3 Training the ANN

Now we want to train an ANN so we can use it as a KAD module. The input parameters of the design and therefore of the ANN are the desired phase shift $\Delta\phi$, forward and reverse reactances X_f and X_r and the series loss resistance R_S. The design, and therefore the ANN, yields the quarter wave transformer impedance Z_T and the electrical length of the Z_0 transmission line θ_T (see Figure 10).

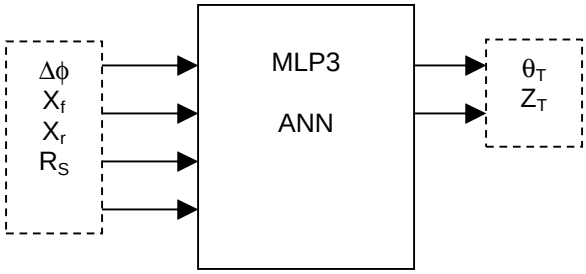

Figure 10 ANN for the reflection type phase shifter

In order to train the ANN we generate a series of training samples. These training samples need to cover the whole range of all of the input parameters. Therefore we need to define a valid range of these parameters. Commonsense, practical considerations and analysis of the resulting design data gives us the ranges shown in Table 2.

Table 2 Ranges of input parameters

Input Parameter	Range	
	Low End	High End
$\Delta\phi$	11.25°	90°
R_S	0Ω	20Ω
X_f	1Ω	26Ω
X_r	-80Ω	-1500Ω

Now we defined in what range the input parameters need to be, but how many data samples do we need? First we set the resolutions of the input parameters to the values shown in Table 3.

Table 3 Resolution of input parameters

Input Parameter	Number of data points in range	Spacing of data points
$\Delta\phi$	8 values	linear spacing
R_S	5 values	linear spacing
X_f	6 values	linear spacing
X_r	30 values	linear spacing

Using the design method outlined in 3.2 we write a Matlab code to generate these data samples. We will use the software NeuroModeler, which is available in its introductory version in [1] to generate and train the ANN. NeuroModeler requires the following file structure:

- → The file is written in ASCII text format.
- → Each line in the file represents an individual data sample.
- → Numerical values within a line are separated by a blank space.
- → The first n values in a line represent the n input values.
- → The following m values in a line represent the m output values.
- → Each line is terminated by a carriage-return line-feed command sequence.

The Matlab code creating this training file is explained and shown in 7.1.1.

Analyzing the resulting data we find that the output parameters don't depend linearly upon some input parameters. As shown in Figure 11 Z_T and especially θ_T depend some kind of exponentially on X_r. Similar the output parameters seem to depend 'exponentially' on $\Delta\phi$. Therefore we change the spacing of the training samples as indicated in Table 4.

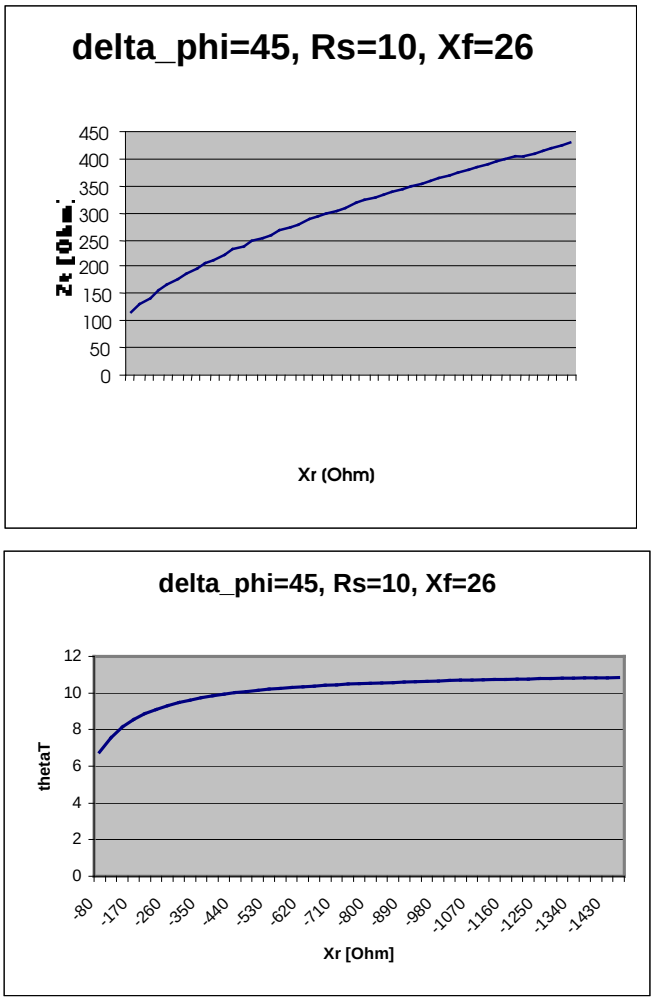

Figure 11 Behaviors of Z_T and θ_T dependent on X_r

Table 4 Resolution of input parameters

Input Parameter	Number of data points in range	Spacing of data points
$\Delta\phi$	11 values	logarithmic spacing
R_S	5 values	linear spacing
X_f	6 values	linear spacing
X_r	30 values	logarithmic spacing

With the obtained data samples we train an MLP3 ANN with 17 hidden neurons. The number 17 comes from several trials and comparing training errors. With 17 hidden neurons the ANN seems able to learn the given samples accurately. Training this ANN with the 9,9000 data samples we get a training error of 0.008, which is very good. To verify how well the ANN is working as a KAD module we generate a file with 5,000 data samples (limited by NeuroModeler) where the input parameters are randomly selected within their respective data range (see 7.1.1 for

the Matlab code). This file is then used to test the ANN. We get an overall average error of 0.66%. The worst-case error is at 9.2%. Table 5 shows some results, compared with results from the conventional design. Also provided is the resulting, actual phase shift using these circuit parameters in an ADS simulation.

Table 5 Some results for the Reflection Type Phase Shifter

Design Input				ANN-KAD-Model			Conventional Design		
$\Delta\phi$	R_S	X_f	X_r	Z_T	θ_T	$\Delta\phi^1$	Z_T	θ_T	$\Delta\phi^1$
22.5	0	5	-400	321.04	5.64	22.2	319.14	5.48	22.500
22.5	5	10	-800	457.42	5.36	21.9	451.34	5.48	22.504
45.0	15	2	-1200	363.29	11.33	48.9	380.97	11.21	45.009
45.0	10	15	-300	192.11	10.08	46.5	195.77	10.11	44.998
90.0	10	4	-600	177.17	22.29	88.2	174.42	22.04	89.952

[1] Computed using circuit parameters in an ADS simulation.

4 Loaded Line Type Phase Shifter

In this chapter we will develop an ANN as KAD module for the design of loaded line type phase shifters with stub mounted switching devices.

4.1 The Circuit

Lets first have a look at the design shown in Figure 12.

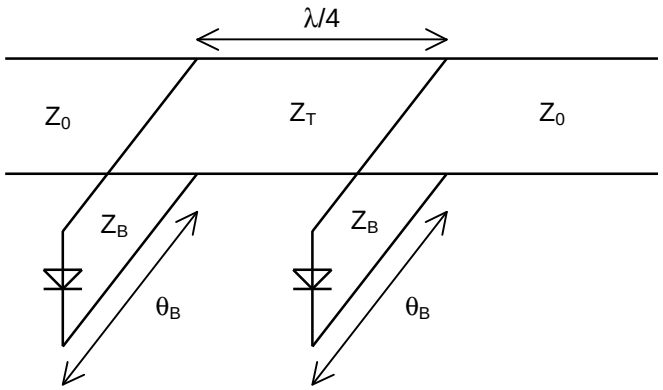

Figure 12 Loaded line phase shifter with stub mounted PIN diodes

First we look only at one stub. Due to the discontinuity (stub) on the transmission line, a part of the energy is reflected and a part transmitted. The reflection coefficient, as well as the transmission coefficient, has a certain angle. The angle of the transmission coefficient is what we use for the phase shifter. The disadvantage is that a part of the energy is reflected. That's why we use two stubs separated by a length of λ/4. Then the wave reflected off the second stub will be 180° out of phase by the time it arrives at the first stub and therefore cancel the reflection. In order to match this circuit we need to have a certain transmission line impedance Z_T in between the stubs. Also we need the absolute value of the reactances of the load (stub) to be constant in the two states.

4.2 Design

Now let's have a look at the design equations derived in [2]. Z_T can simply be expressed as

$$Z_T = Z_0 \cdot \cos\left(\frac{\Delta\phi}{2}\right) \tag{7}$$

The admittance B seen into the stubs in the two bias states needs to be

$$B_{1,2} = \pm \frac{1}{Z_0} \cdot \tan\left(\frac{\Delta\phi}{2}\right) \quad \textbf{(8)}$$

From this we can get the needed stub line impedance Z_B

$$Z_B = \sqrt{\frac{X_F - X_R - X_F X_R (B_1 - B_2)}{B_1 - B_2 - B_1 B_2 (X_F - X_R)}} \quad \textbf{(9)}$$

and the stub length θ_B

$$\theta_B = \tan^{-1}\left(Z_B \cdot \frac{1 + X_F B_1}{X_F - B_1 Z_B{}^2}\right) \quad \textbf{(10)}$$

If θ_B results in a negative value, n*π can be added.

These design equations are good for the lossless case only. For the case of lossy diodes (i.e. $R_S \neq 0$) we need to optimize the circuit. In chapter 4.3 we will treat the lossless case and in chapter 4.4 the lossy case. As we have closed form relations for the lossless case it is not very attractive to train an ANN for this purpose. We anyway do it as a learning experience to get a basis for the lossy case.

4.3 Lossless Case

Again, as in 3.3, we want to train an ANN so we can use it as a KAD module. The input parameters of the design and therefore of the ANN are the desired phase shift $\Delta\phi$, the system impedance Z_0 as well as the forward and reverse reactances X_f and X_r. The design, and therefore the ANN, yields the impedance Z_T of the quarter wave long transmission line between the stubs, the stub impedance Z_B and the electrical stub length θ_B (see Figure 13).

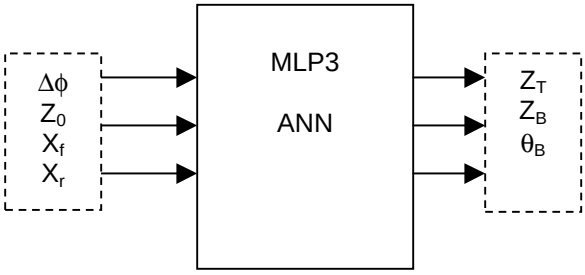

Figure 13 ANN for the loaded line type phase shifter with lossless switching devices

Again we define ranges for the input parameters. The selected values are shown in Table 6. The respective resolutions of the input parameters are shown in Table 7.

Table 6 Ranges of input parameters

Input Parameter	Range	
	Low End	High End
$\Delta\phi$	9°	90°
Z_0	10Ω	100Ω
X_f	0.5Ω	5Ω
X_r	-50Ω	-400Ω

Table 7 Resolution of input parameters

Input Parameter	Number of data points in range	Spacing of data points
$\Delta\phi$	9 values	linear spacing
Z_0	19 values	linear spacing
X_f	10 values	linear spacing
X_r	36 values	linear spacing

This results in 61,530 training samples. A training file as described in 3.3 is generated by a Matlab code (see 7.1.2 for description of code). The resulting file has a size of 2.26 Mega Bytes, which is still handy. With this data we train a MLP3 ANN with 15 hidden neurons. The resulting training error is 0.006. We also generate five files with each 5,000 data samples to test this ANN. Again the input parameters for these test data files are chosen randomly within their respective range. The results are shown in Table 8.

Table 8 Error from the ANN after training with the first training data set

Sample File	Average Error [%]		Worst-Case Error [%]		Correlation Coefficient
	Z_b	θ_b	Z_b	θ_b	
File 1	0.6802	0.7366	5.3717	6.5676	0.99982
File 2	0.6714	0.7305	5.8225	6.3654	0.99982
File 3	0.6684	0.7292	7.1981	6.7791	0.99981
File 4	0.6793	0.7202	6.3722	6.0007	0.99981
File 5	0.6856	0.7192	5.8842	6.7391	0.99980
Average	0.6770	0.7271	6.1297	6.4904	0.99981

This is already quiet good. We could use the results from this ANN as initial values for a design optimization in a circuit simulator like ADS. But let's see if we can improve the quality of the results. We decide to double the number of values for X_r. The result is 123,120 data samples and a file size of 4.17 Mega Bytes. Unfortunately NeuroModeler is not able to load such an amount of data, even though the computer has 256 Mega Bytes of RAM. We try to overcome this limit and reduce the file size by cutting off values at the end. The ANN is trained with this truncated file and then again trained with the same file, but truncated at the beginning. Testing the resulting ANN with the same five test files we get the results shown in Table 9.

Table 9 Error from the ANN after training with the second training data set

Sample File	Average Error [%]		Worst-Case Error [%]		Correlation Coefficient
	Z_b	θ_b	Z_b	θ_b	
File 1	0.4473	0.6423	5.5614	6.3378	0.99984
File 2	0.4349	0.6297	6.6248	6.4100	0.99984
File 3	0.4414	0.6410	7.0718	5.5100	0.99984
File 4	0.4451	0.6329	7.0841	5.7367	0.99984
File 5	0.4576	0.6332	6.9354	5.9683	0.99982
Average	0.4453	0.6358	6.6555	5.9926	0.99984

These results are slightly better than the ones shown in Table 8.

As we decided to use NeuroModeler we have to live with it's limitations. So let's take another shot and in the same time use the same input parameter ranges as for the reflection type phase shifter. To work around the limitation of NeuroModeler we drop Z_0 as input parameter and set it constant to 50Ω. We also apply non-linear spacing to $\Delta\phi$ and X_r.

Table 10 Ranges of input parameters

Input Parameter	Range	
	Low End	High End
$\Delta\phi$	11.25°	90°
Z_0	Constant 50Ω	
X_f	1Ω	26Ω
X_r	-80Ω	-1500Ω

Table 11 Resolution of input parameters

Input Parameter	Number of data points in range	Spacing of data points
$\Delta\phi$	10 values	logarithmic spacing
Z_0	constant	
X_f	6 values	linear spacing
X_r	35 values	logarithmic spacing

Training an ANN with 14 hidden neurons with this training data results in a training error of 0.007. Testing this ANN with one of the test files gives an overall average error of 0.794% and a maximum error of 4.7%.

4.4 Lossy Case

More interesting than the lossless case, where we have analytical solutions, is the lossy case. In this chapter we will try to train an ANN to give circuit parameters based on values of lossy devices.

4.4.1 ADS Optimization

First we try to generate training data with an optimization in ADS. We set up the simulation shown in Figure 14. The optimization yields near-optimum parameters for the line impedance Z_T, the stub impedance Z_B and stub length θ_T.

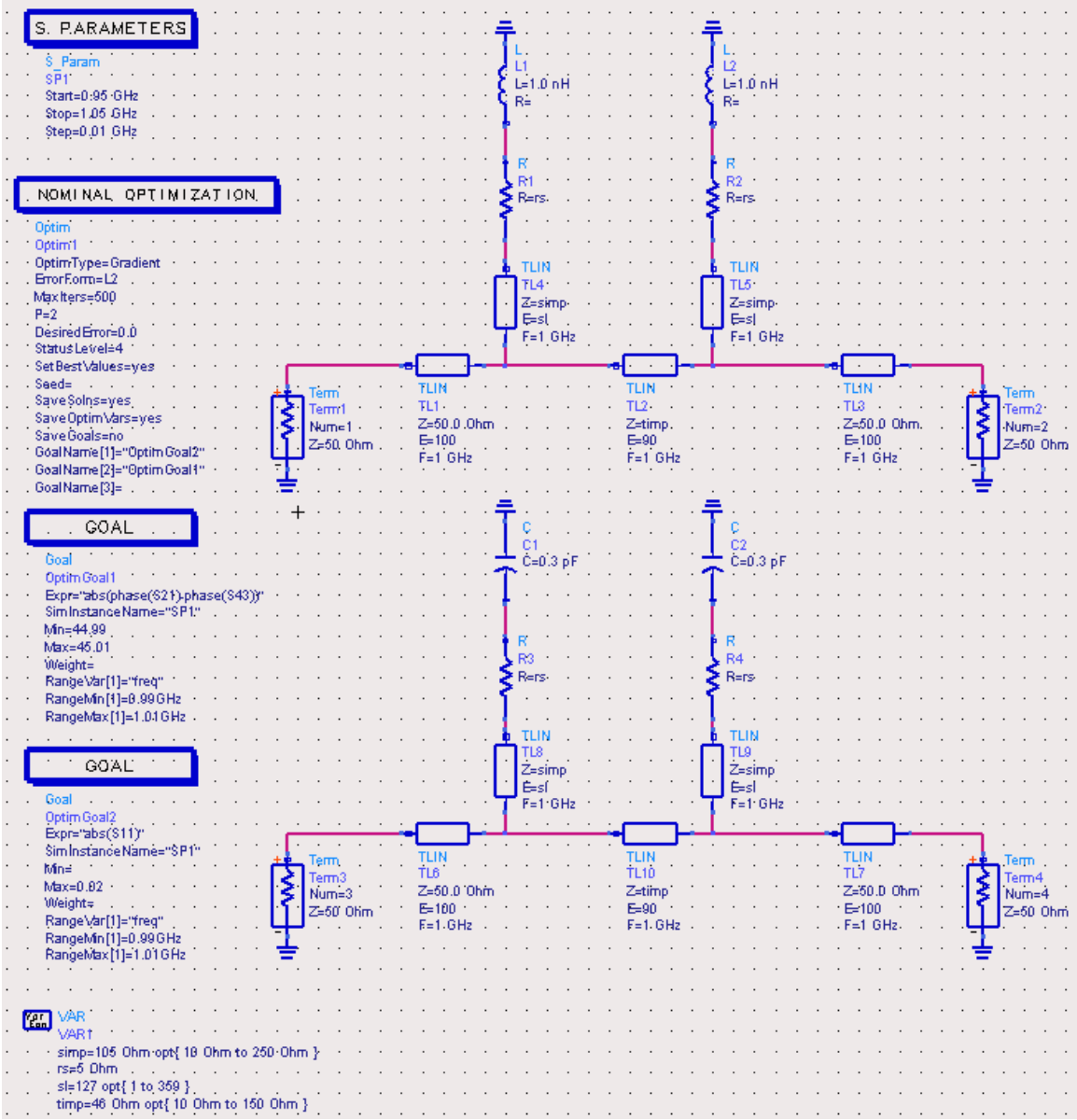

Figure 14 ADS simulation

Let's have a look at this ADS simulation / optimization setup. The phase shifter circuit was built twice, once with the diodes represented in their forward bias state and once in its reverse bias state. In the low impedance state (forward bias) we see the package inductance and a series resistance. This resistance was neglected when we did the design for the lossless case. In the high impedance state of the diode

(reverse bias) we see the junction capacitance plus again the same resistance in series.

Now a S-parameter analysis is performed, in a narrow band around the targeted frequency (this phase shifter is a fairly narrow banded device). In this example it is from 0.95 GHz to 1.05 GHz in steps of 0.01 GHz.

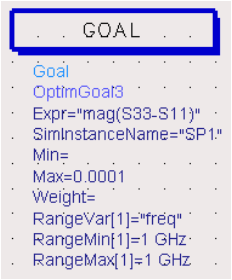

Figure 15 Additional Optimization Goal to keep S11 constant

We set the optimization method to 'Gradient'. This is after different methods were tried and the 'gradient' method seems to be the fastest when we take the calculated parameters as start value. The maximum number of iterations we set to 500. The optimization has three goals: 'OptimGoal1', 'OptimGoal2' and 'OptimGoal3' (third goal shown in Figure 15). 'OptimGoal1' represents the phase difference $\Delta\phi$ of the phase shifter. $\Delta\phi$ is calculated as the absolute value of the difference in the phase of S21. We give a narrow tolerance of $\Delta\phi \pm 0.01$ within the frequency of 0.99 GHz to 1.01 GHz. The advantage of setting a narrow frequency range instead of only the single target frequency is that the optimization will find a flat slope (maybe an extremum) instead of a steep slope where the specified $\Delta\phi$ is only satisfied at exactly the design frequency (i.e. in Figure 16 rather (a) than (b)). This naturally will result in a higher bandwidth.

(a) (b)

Figure 16 Phase shift dependent on frequency

'OptimGoal2' represents the optimization goal to have a phase shifter matched to the system impedance (here 50 Ohm). The goal is to keep the absolute value of S11 (same as S22, as the circuit is symmetric) as low as possible. Here we set a maximum value of 0.02. The third goal is to keep S11 in both states the same.

For the stub impedance Z_B we define the variable 'simp' which can be optimized in the limits of 10 Ohms up to 250 Ohms. For the stub length θ_B we define the variable 'sl' which can be optimized in the limits of 1 degree up to 359 degree. Finally for the $\lambda/4$ transmission line impedance Z_T we define the variable 'timp' which can be optimized in the limits of 10 Ohm up to 150 Ohm (actually Z_T should always be smaller or same as Z_0 [2]). One more variable 'rs' is defined. This is the value of the series resistance of the diode. As start values of the optimization variables we use the values for the lossless case as we calculated them in chapter 4.3.

Running the simulation (the example as shown in Figure 14) with the optimization we get the results shown in Figure 17. The blue lines are the final results.

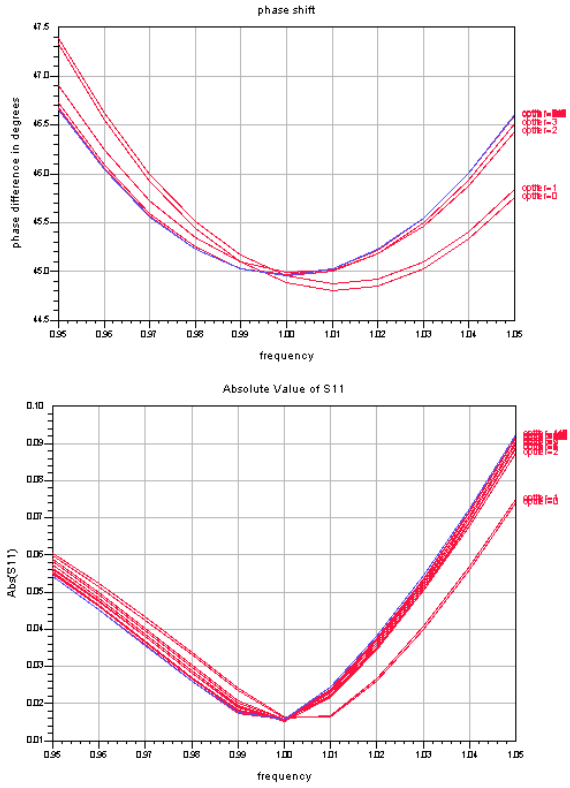

Figure 17 ADS optimization result

Now in order to generate training data for an ANN we need to sweep the input parameters over their respective range, as we did in the lossless case. ADS needs then to optimize the circuit at every combination of input parameters and save the result. An ANN then can be trained with the resulting data. A sweep over all values of the series resistance R_S was implemented, but unfortunately ADS would always quit with the error message 'OptimGoal / YieldSpec number of residuals changed'. Time was spent to resolve this problem, but neither examples nor any hints were

found in the documentation of ADS how to run an optimization together with a sweep. Also the ADS help desk didn't come up with an answer. Therefore we had to postpone this investigation and concentrate on alternative solutions. These attempts are outlined in the following chapter.

4.4.2 ANN for the Lossy Case

As we were not able to perform an ADS optimization within a sweep we cannot get a large amount of training data. Each training sample needs to be entered by hand, optimized and the results copied to the training file. This manual work prohibits large training data amounts in the given time frame. In this chapter we attempt to train an ANN with a few training samples and still get accurate results. It is to mention in advance that those attempts were unsuccessful.

First we need to get some training samples by the way of optimization. We decide to get a sample at all the endpoints of the input parameter ranges. For this reason we again define the ranges of the input parameters. As we deal this time with a circuit representation of the diodes we represent them not with a reactance, but with a capacitor respectively an inductor. So we will define the ranges in terms of capacitances respectively inductances. This is shown in Table 12.

Table 12 Ranges of circuit elements and their values at 1GHz

Input Parameter	Range	
	Low End	High End
$\Delta\phi$	11.25°	90°
Z_0	Constant 50Ω	
L_D (X_f)	0.1nH (0.628Ω)	4nH (25.13Ω)
C_D (X_r)	2pF (-79.58Ω)	0.1pF (-1.592kΩ)

Doing an optimization at all the endpoints (2^4=16 optimizations) we get the data shown in Table 13.

Table 13 Optimized parameters at endpoints

Input parameters				Optimization result		
$\Delta\phi$ [°]	C_D [pF]	L_D [nH]	R_S [Ω]	Z_B [Ω]	θ_B [°]	Z_T [Ω]
11.25	0.1	0.1	0.5	397.528	129.643	49.736
11.25	0.1	0.1	20.0	396.789	129.721	49.721
11.25	0.1	4.0	0.5	416.810	125.940	49.760
11.25	0.1	4.0	20.0	416.810	125.940	49.760
11.25	2.0	0.1	0.5	138.807	113.057	49.418
11.25	2.0	0.1	20.0	138.479	114.460	49.374
11.25	2.0	4.0	0.5	161.274	103.003	49.620
11.25	2.0	4.0	20.0	161.094	103.572	49.583
90.00	0.1	0.1	0.5	112.920	133.145	46.116
90.00	0.1	0.1	20.0	106.568	136.222	45.084
90.00	0.1	4.0	0.5	133.883	126.862	46.257
90.00	0.1	4.0	20.0	129.374	128.464	45.387
90.00	2.0	0.1	0.5	60.034	119.564	45.240
90.00	2.0	0.1	20.0	56.724	129.742	47.796
90.00	2.0	4.0	0.5	75.925	105.366	45.854
90.00	2.0	4.0	20.0	73.156	108.895	43.766

Now what can we do with this data? We have several options.

One approach is to take an untrained ANN and train it with the 16 data samples. But we know that the output parameters don't depend linearly upon the input parameters. So we won't get a good performance from such an ANN.

Another approach is to train an ANN in the so-called prior knowledge configuration. With this method we feed the ANN not only with the design parameters (i.e. $\Delta\phi$, X_f, X_r, R_S) but also with the corresponding solutions for the lossless case (i.e. Z_{Bll} θ_{Bll} Z_{Tll}) as shown in Figure 18.

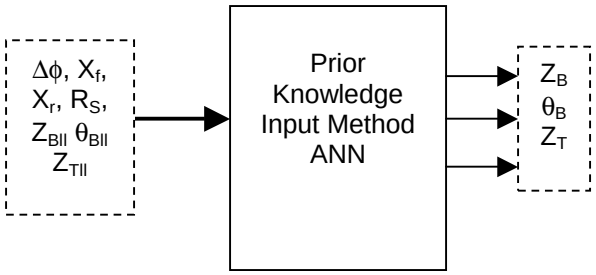

Figure 18 Prior knowledge Input Method

The advantage of this method is that we give the ANN an additional help by providing the solutions for the lossless case. Those are very close to the solutions for the lossy case. In this configuration the ANN needs only to modify the provided lossless solutions according to the provided loss resistance R_S. If the change of the

output parameters depends linearly upon R_S the ANN should be able to give accurate results, also if trained only with a few data samples.

Now we even can take a step further. Looking at the ANN in Figure 18 we realize that we can simplify the ANN by only providing the lossless solutions (Z_{Bll} θ_{Bll} Z_{Tll}) plus the loss resistance R_S. The ANN's job then is to modify the lossless results according to R_S. This ANN is shown in Figure 19

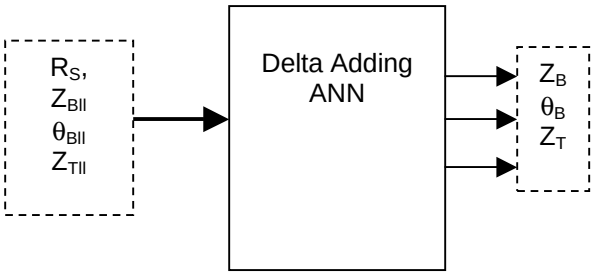

Figure 19 ANN to add delta based on R_S

Yet another ANN configuration is similar to the one shown in Figure 19, but the output is not formed by the circuit parameters, but by the difference in the circuit parameters based on R_S. The advantage is that the small change, and not the large circuit parameters plus the small change is resulting from the ANN. In the second case a small percentage error of the result can erase the small change based on R_S.

All the above configurations have been applied and trained with the 16 training samples. Training errors have been in the magnitude of 0.01. Testing the ANNs with a test sample at the center of all input parameters resulted in errors of 30% to 60%. This clearly shows that the 16 samples are not sufficient. It seems that the change in output parameters is not linearly dependent on R_S. Finally we do what should have been done earlier. We analyze the dependence of the output parameters upon the change in R_S. We fix the input parameters at one value and vary R_S within its range. The resulting numbers are shown in Table 14. The interpolated plots are shown in Figure 20. The graphs show that the parameters are quiet close linear dependent on R_S. θ_B seems a little more non-linear dependent than the other two. But we wouldn't expect an ANN to generate an error of 40% from this behavior. Of course we don't know if this behavior gets more non-linear at other input values. Another point is how the ANN tends to interpolate between two training samples. This might depend on the number of Neurons, the training method and the general architecture of the ANN.

Finally we can say that we need more data samples than just the ones at the edges of the input ranges. It might be that already with the double or triple amount of training data we can get an accurate result. Further investigations will be needed.

Table 14 Parameters obtained from optimization, input parameters fixed

R_S [Ω]	Z_B [Ω]	θ_B [°]	Z_T [Ω]
0	93.510	127.200	46.112
1	93.461	127.249	46.085
2	93.334	127.311	46.015
3	93.164	127.390	45.927
4	92.981	127.484	45.840
5	92.800	127.592	45.763
6	92.604	127.717	45.687
7	92.399	127.857	45.615
8	92.183	128.015	45.546
9	91.954	128.192	45.480
10	91.710	128.388	45.416
15	90.226	129.736	45.118
20	88.117	132.051	44.851

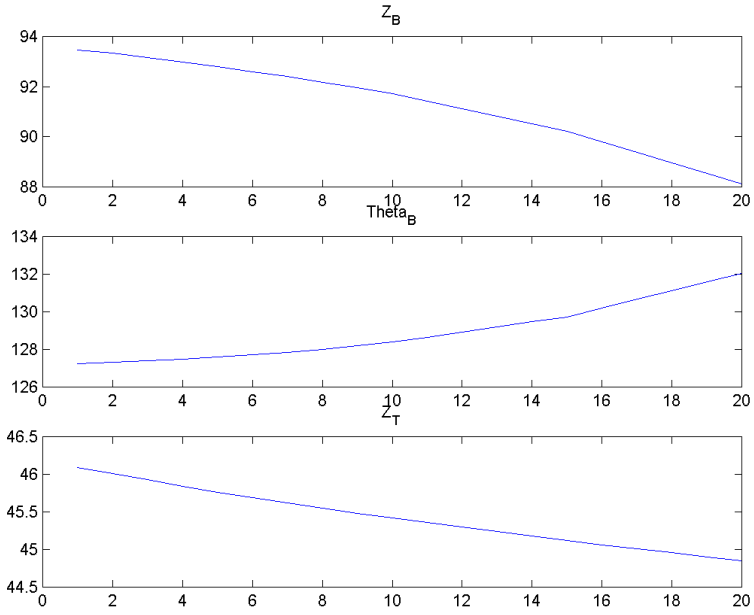

Figure 20 Dependence of output parameters on R_S

5 Other Applications

5.1 S → Γ mapping for maximum gain amplifier

As a third example an ANN should have been trained to get the source and load reflection coefficients for maximum gain amplifier from the s-parameters of a transistor. The desired ANN is shown in Figure 21.

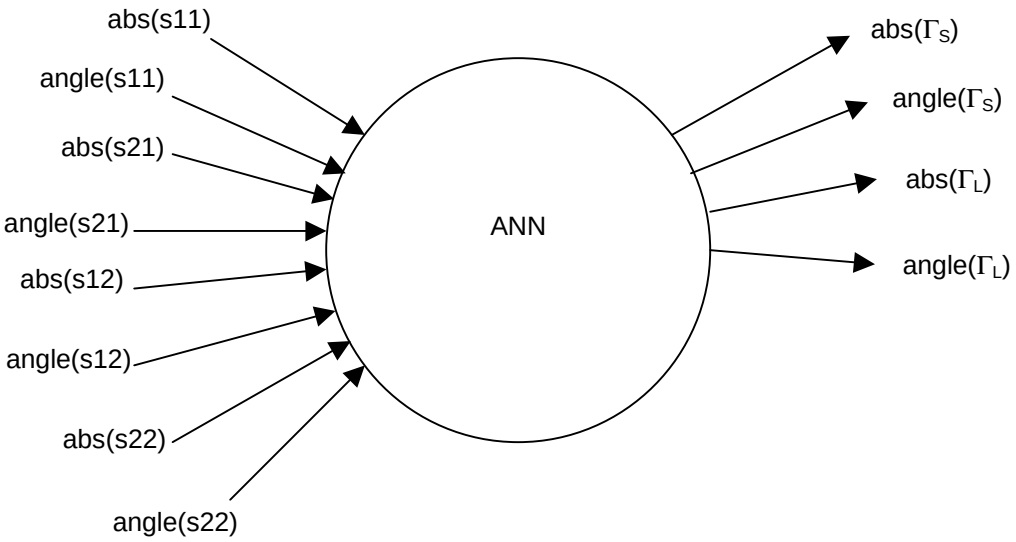

Figure 21 ANN for S to Γ mapping

A Matlab core was written to generate the training and test data. The training of the ANN could not be completed due to limitations of NeuroModeler. As we have eight input parameters to this ANN we get an enormous amount of training data. NeuroModeler in its introductory version is not able to handle such large data amounts. The advanced version of NeuroModeler became too late available to obtain any results for this report.

Some effort was taken to reduce the required amount of training data by reducing the valid range of input parameters. For this purpose a small transistor survey was made. The Agilent transistor models (RF FETs and PHEMTs) shown in Table 15 were included in this survey.

Table 15 Models included in survey

Agilent:
ATF-10 family
all
ATF-13 family
13336
13736
13786
ATF-26 family
26836
26884
ATF-36 family
all
AT-420xx power family
42000
42010
42035
42036
42070
42085
42086
ATF-44 power family
44101

The device s-parameters were plotted in different ways in the hope to detect a certain limited range for valid s-parameters. The results are shown in Figure 22 through Figure 26. Even though there are some 'holes' where we can save on training data, there is no real breakthrough to reduce the training data to an amount that can be handled by NeuroModeler.

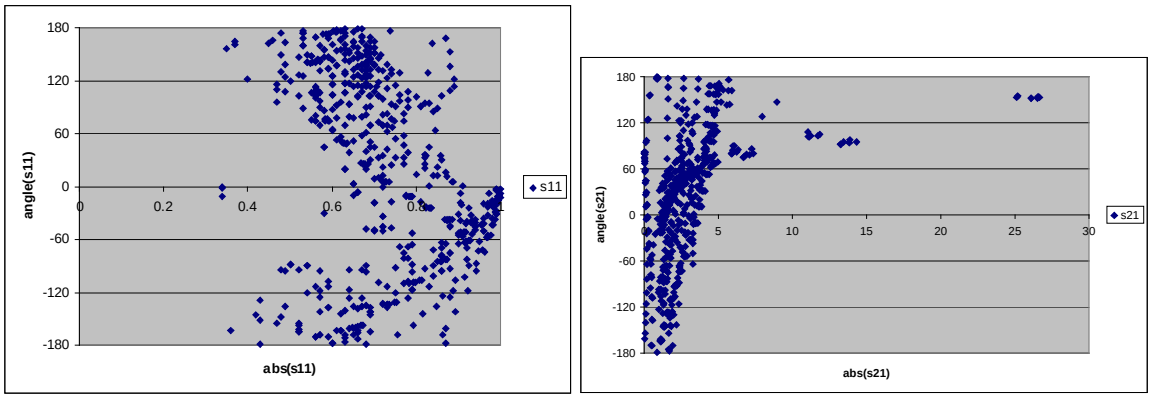

Figure 22

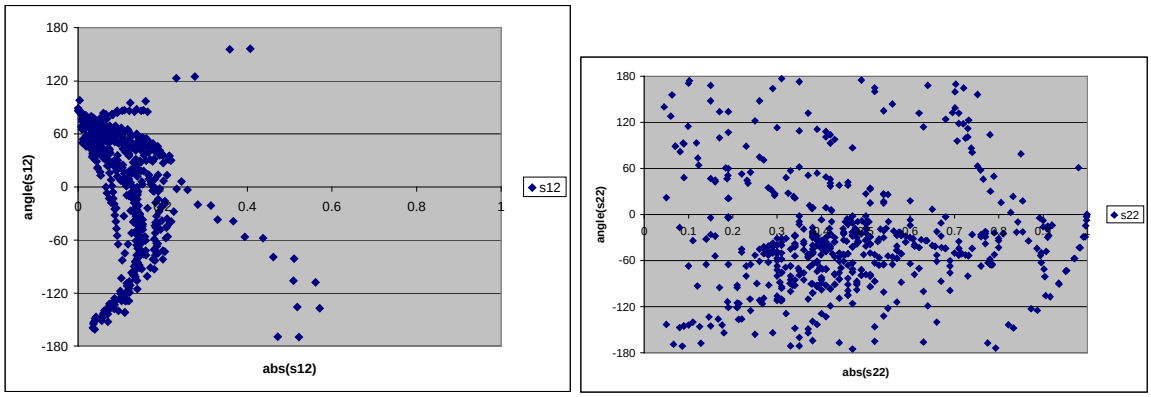

Figure 23

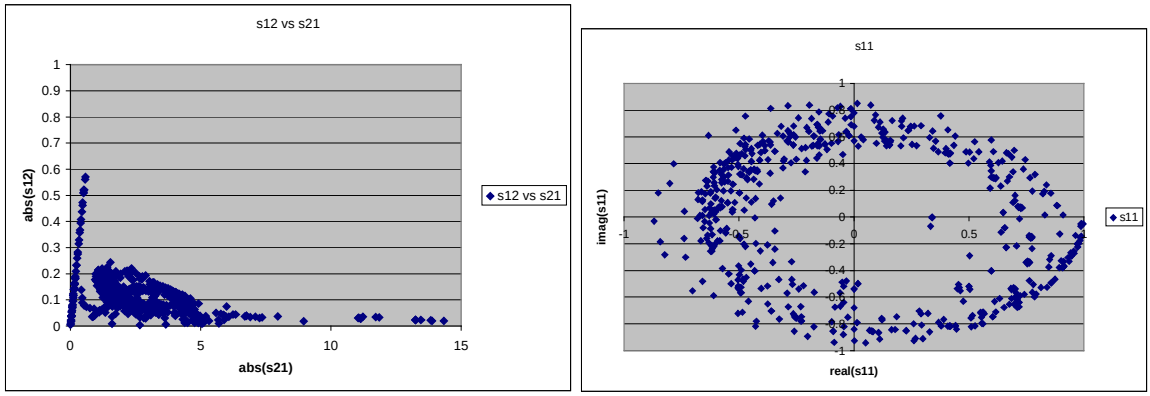

Figure 24

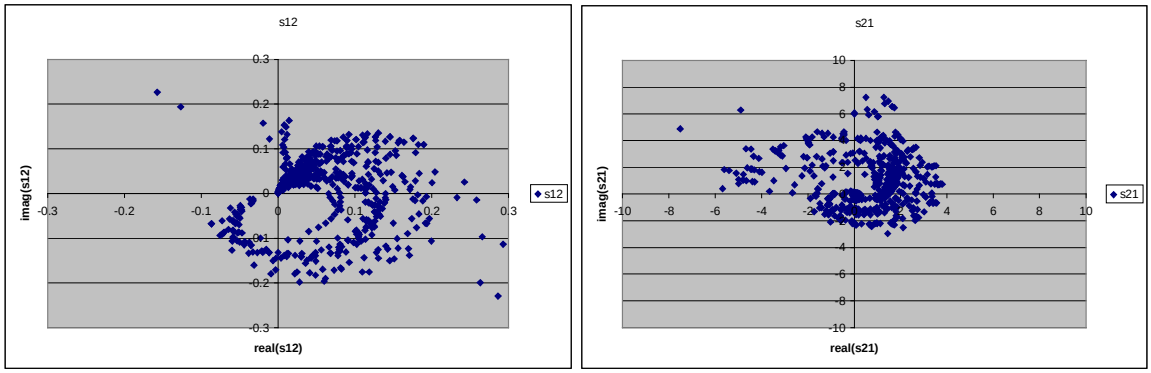

Figure 25

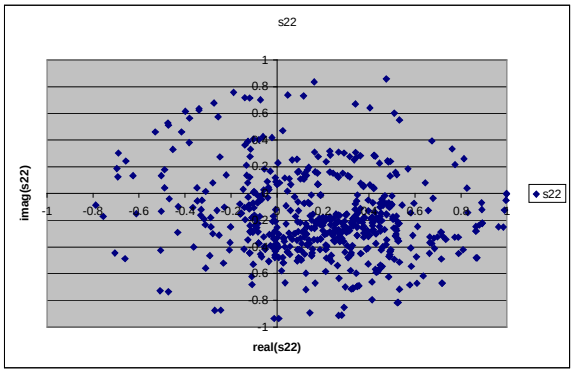

Figure 26

5.2 Open Stub Matching Network

As a fourth example we developed an ANN to get the circuit parameters of an open stub matching network from the desired reflection coefficient and the system impedance. The ANN is shown in Figure 27. Training the ANN resulted in a training error of 0.014. Testing the ANN gives us an average error of 1.5%. These are good results.

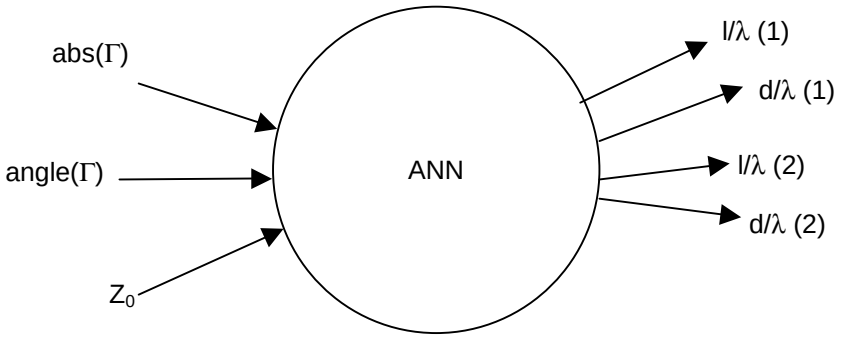

Figure 27 ANN to get open stub matching network parameters

6 Reference

[1] Q. J. Zhang, K. C. Gupta, Neural Networks for RF and Microwave Design, Norwood: Artech House, 2000.

[2] K. C. Gupta, “Microwave Control Circuits”, New York: John Wiley and Sons, 1988.

7 Appendix

7.1 Matlab Codes

7.1.1 Reflection type Phase Shifter

The Matlab codes ‘make_refl_train.m’ and ‘make_refl_test.m’ generate a file with sample data for a reflection type phase shifter. The only difference is that ‘make_refl_train.m’ generates this data in a certain, specified grid of input parameters and ‘make_refl_test.m’ generates data for randomly selected input parameters. Both codes use the function ‘refl_ps’ to determine the circuit parameters. The call and function use structure the presented code is the following:

--- make_refl_train.m or make_refl_test.m
|
|--- findZm.m
| |
| |--- vf924.m
|
|--- find_trafo.m
|
|--- trafo_eq.m

make_refl_train.m

```
% reflection type phase shifter with 90deg hybrid
%
% input is:
%   Zo: system impedance
%   Xf: forward bias reactance of PIN diode
%   Xr: reverse bias reactance of PIN diode
%   delta_phi: desired phase shift

% output is:
%   Zt: required impedance of lambda/4 transformer
%   thetaT: required length of Zo transmission line between hybrid and lambda/4
transformer


clear;

[fid,message]=fopen('refl_train_pli11s_rli5_xfli6_xrlo30cmp70_100.dat','wt');
if fid==-1
  error(strcat('file refl_train.dat could not be opened. Message
is:\n',message));
end

% given values ++++++++++++++++++++++++++++++++++++++

% system impedance
Z0=50;
% given a required phase shift delta_phi
```

```
cmp=1; % compression (expansion) factor
%dp=logspace(log10(pi/16+cmp),log10(pi/2+cmp),20)-cmp;
dp=[linspace(pi/16,2*pi/16,3) linspace(3*pi/16,5*pi/16,3)
linspace(6*pi/16,8*pi/16,5)];
% given series resistance Rs
rs=linspace(0,20,5);
% given diode forward reactance Xf
xf=linspace(1,26,6);
% given diode reverse reactance Xr
cmp=70;
xr=-(logspace(log10(100-cmp),log10(1500-cmp),30)+cmp);

for delta_phi=dp
  for Rs=rs
    for Xf=xf
      for Xr=xr
         [thetaT,ZT,thetaS]=refl_ps(Z0,Xf,Xr,Rs,delta_phi);

         % write to file
         %             |----------------------delta_phi  -|
         %             |      |--------------Rs           | design
         %             |      |    |---------Xf           | input
         %             |      |    |     |---Xr         -|
         %             |      |    |     |
         %             |      |    |     |     |--------------ZT      -|
         %             |      |    |     |     |     |--------thetaT   | output
         %             |      |    |     |     |     |     |--thetaS  -|
         fprintf(fid,'%1.2f %1.0f %1.0f %1.0f %1.4f %1.4f
%1.4f\n',delta_phi*180/pi,Rs,Xf,Xr,ZT,thetaT*180/pi,thetaS*180/pi);
      end
    end
  end
end
fclose(fid);
```

refl_ps.m

```
function [thetaT,ZT,thetaS]=refl_ps(Z0,Xf,Xr,Rs,delta_phi)
% [thetaT,ZT,thetaS]=refl_ps(Z0,Xf,Xr,Rs,delta_phi)
% calculate the circuit parameters of a reflection type phase shifter
%
% input:
% Z0: system impedance
% Xf: forward reactance of switching device
% Xr: reverse reactance of switching device
% Rs: series resistance of switching device
% delta_phi: desired phase shift
%
% output:
% thetaT: transformer transmission line (ZT) length
% ZT:     transformer transmission line (thetaT) impedance
% thetaS: transmission line (Z0) length


Zf=Rs+j*Xf;
Zr=Rs+j*Xr;

% calculate Zm
Zm=findZm(Zf,Zr,delta_phi);

% find transformer to get Zm
[ZT thetaT]=find_trafo(Zm,Z0);
```

```
% calculate ZA
ZAf=ZT*(Zf+i*ZT*tan(thetaT))/(ZT+i*Zf*tan(thetaT));

% calculate thetaS
thetaS=find_thetaS(Z0,ZAf,delta_phi);
```

findZm.m

```
function Zm=findZm(Zf,Zr,delta_phi)
%
options = optimset('Display','off');

Xf=imag(Zf);
Xr=imag(Zr);

% first get rZm for the lossless case
rZm=(Xf-Xr)/(2*tan(delta_phi/2))+sqrt(((Xf-Xr)/(2*tan(delta_phi/2)))^2-Xf*Xr);

vZm=fminsearch('vf924',[rZm 0],options,Zf,Zr,delta_phi);
Zm=vZm(1)+i*vZm(2);
```

vf924

```
function err=vf924(vZm,Z1,Z2,deltaPhi)
%
rZm=vZm(1);
iZm=vZm(2);
Zm=rZm+(i*iZm);
t=tan(deltaPhi/2);
err=abs(Zm*(Z2+i*Zm*t)./(Zm+i*Z2*t)-Z1);
```

find_trafo.m

```
function [ZT,thetaT]=find_trafo(Zm,Z0)
%

options = optimset('Display','off');

s=sign(imag(Zm));
if(s==0)
   % imaginary part is 0 -> quarter wave transformer
   ZT=sqrt(Zm*Z0);
   thetaT=pi/2;
else
   Re=real(Zm);
   x=fminsearch('trafo_eq',[s*20 sqrt(Re*Z0)],options,Zm,Z0);
   ZT=x(2);
   thetaT=atan(x(1));
   if(s<0)
      thetaT=thetaT+pi;
   end
end
```

trafo_eq.m

```
function err=trafo_eq(x,Zm,Z0)

err=abs(x(2)*(Z0+i*x(2)*x(1))/(x(2)+i*Z0*x(1))-Zm);
```

make_refl_test.m

```
% Reflection-type phase shifter
% Create training files
```

```
for x=1:4
  % create random file number
  fn=round(rand(1)*100);

  fin=strcat('refl_test_',num2str(fn),'.dat');
  [fid,message]=fopen(fin,'wt');
  if fid==-1
    error(strcat('file ',fin,' could not be opened. Message is:/n',message));
  end

  rand('state',sum(100*clock))
  Z0=50;

  for i=1:5000
    % generate random value for phase shift in interval [pi/16,pi/2] to avoid 0.0
    delta_phi=rand(1)*pi*7/16+pi/16;

    % generate random value for diode forward reactance Xf in interval [1,26]
    Xf=rand(1)*25+1;

    % generate random value for diode reverse reactance Xr in interval [-80,-
1500]
    Xr=-(rand(1)*1420+80);

    % generate random value for diode series resistance Rs in interval [0,20]
    Rs=rand(1)*20;

    % calculated values ++++++++++++++++++++++++++++++++++
    [thetaT,ZT,thetaS]=refl_ps(Z0,Xf,Xr,Rs,delta_phi);

    % write to file
    %             |---------------------delta_phi  -|
    %             |      |--------------Rs           | design
    %             |      |    |---------Xf           | input
    %             |      |    |     |---Xr         -|
    %             |      |    |     |
    %             |      |    |     |     |--------------ZT      -|
    %             |      |    |     |     |     |--------thetaT   | output
    %             |      |    |     |     |     |     |--thetaS  -|
    fprintf(fid,'%1.2f %1.0f %1.0f %1.0f %1.4f %1.4f
%1.4f\n',delta_phi*180/pi,Rs,Xf,Xr,ZT,thetaT*180/pi,thetaS*180/pi);
  end
  fclose(fid);
end
```

7.1.2 Loaded Line Type Phase Shifter

make_loaded_train.m

The Matlab program ‘make_loaded_train.m’ generates a file of training data to train an ANN with NeuroModeler. The data is for a loaded line type phase shifter without loss. The name of the generated file is ‘loaded_train.dat’. This file is filled with training samples. Each line in the file represents one training sample. Single values are separated by a blank space. The numerical values are written in ASCII text. The first three values on each line represent the three input parameters $\Delta\phi$, X_f and X_r. The following three values represent the output parameters Z_B, θ_B and Z_T.

```
% stub-mounted loaded-line phase shifter
% assume: theta=pi/2 at f0


[fid,message]=fopen('loaded_train.dat','wt');
if fid==-1
  error(strcat('file loaded_train.dat could not be opened. Message
is:\n',message));
end

% given values ++++++++++++++++++++++++++++++++++++++

Z0=50;

% given a required phase shift deltaPhi
dp=logspace(log10(pi/16),log10(pi/2),10); % 11.25 : 11.25 : 90

% given diode forward reactance Xf
xf=1:5:26;

% given diode reverse reactance Xr
xr=-logspace(log10(80),log10(1530),35); % 35 values

for deltaPhi=dp
   for Xf=xf
      for Xr=xr
        % calculated values ++++++++++++++++++++++++++++++++++
        % calculate needed suceptance B
        B1=tan(deltaPhi/2)/Z0;
        B2=-B1;

        % calculate transmission line impedance Zt between shunt impedances
        Zt=Z0*cos(deltaPhi/2);

        % calculate transmission line impedance Zb for stubs (assume B2 = -B1)
        Zb=sqrt((Xf-Xr-Xf*Xr*(B1-B2))/(B1-B2-B1*B2*(Xf-Xr)));

        % calculate length thetaB of stubs
        thetaB=atan(Zb*(1+Xf*B1)/(Xf-B1*Zb^2))+pi;

        % write to file
        fprintf(fid,'%1.2f %1.0f %3.0f %3.3f %1.4f %3.3f\n', deltaPhi/pi*180, Xf,
Xr, Zb, thetaB/pi*180, Zt);
      end
   end
end
fclose(fid);
```

make_loaded_test.m

The Matlab code 'make_loaded_test.m' does basically the same as 'make_loaded_train.m', but the input parameters are selected randomly within their respective range. Each 5,000 test data samples are written in five output files with the names 'loaded_test#.dat', where # stands for the numbers 1 through 5.

```
% stub-mounted loaded-line phase shifter
% assume: theta=pi/2 at f0
```

```
for fn=1:5

  fin=strcat('loaded_test',num2str(fn),'.dat');
  [fid,message]=fopen(fin,'wt');
  if fid==-1
    error(strcat('file ',fin,' could not be opened. Message is:/n',message));
  end

  rand('state',sum(100*clock))

  Z0=50;

  for i=1:5000
    % generate random value for phase shift in interval [pi/16,pi/2]
    deltaPhi=rand(1)*pi*7/16+pi/16;

    % generate random value for diode forward reactance Xf in interval [1,26]
    Xf=rand(1)*25+1;

    % generate random value for diode reverse reactance Xr in interval [-80,-
1530]
    Xr=-(rand(1)*1450+80);

    % calculated values ++++++++++++++++++++++++++++++++++
    % calculate needed suceptance B
    B1=tan(deltaPhi/2)/Z0;
    B2=-B1;

    % calculate transmission line impedance Zt between shunt impedances
    Zt=Z0*cos(deltaPhi/2);

    % calculate transmission line impedance Zb for stubs (assume B2 = -B1)
    Zb=sqrt((Xf-Xr-Xf*Xr*(B1-B2))/(B1-B2-B1*B2*(Xf-Xr)));

    % calculate length thetaB of stubs
    thetaB=atan(Zb*(1+Xf*B1)/(Xf-B1*Zb^2))+pi;

    % write to file
    fprintf(fid,'%1.4f %1.4f %3.4f %3.4f %1.4f
%3.3f\n',deltaPhi/pi*180,Xf,Xr,Zb,thetaB/pi*180,Zt);

  end
  fclose(fid);
end
```