DASP3rd Chapter 11 - Exercises

1 Neural Network and Convolutional Neural Network

1. Fit the function $y = \max(sin(x), -0.2)$ with a feedforward neural network in $[-1, 1]$.

   – The content of the file `dasp_chap11_ex1_NN.m` is explained here. At first, random data points are generated to create the input and ground truth labels x, y. Figure 1 shows the sampled data points. More points can be sampled for a better performance.

```
numdatapoints = 1000;
x = 2*(rand(1,numdatapoints)-0.5);
y = max(sin(x),-0.2);
```

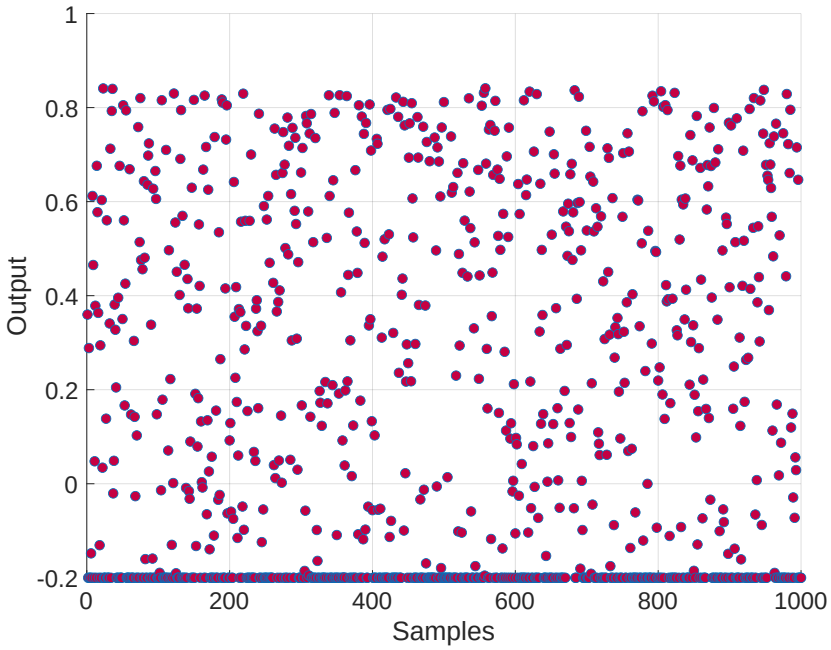

Figure 1: Training data.

   – According to exercise 1 (a), the number of layers and the layer dimensions are defined. 1 hidden layer is selected and there are 3 layers including the input and output. The input layer, the hidden layer, and the output layer sizes are 1, 5, and 1 respectively.

```
numhiddenlayers = 1
n = 1;
k = 5*ones(1,numhiddenlayers);
```

   – The network contains 2 set of weight matrices and bias vectors. The weight matrices between the input and hidden layer, and the hidden and output layer are 1×5 and 5×1 respectively. They are initialized randomly and normalized by their dimensions. The bias vectors are initialized with zeros. The network can be illustrated as in Fig. 2.

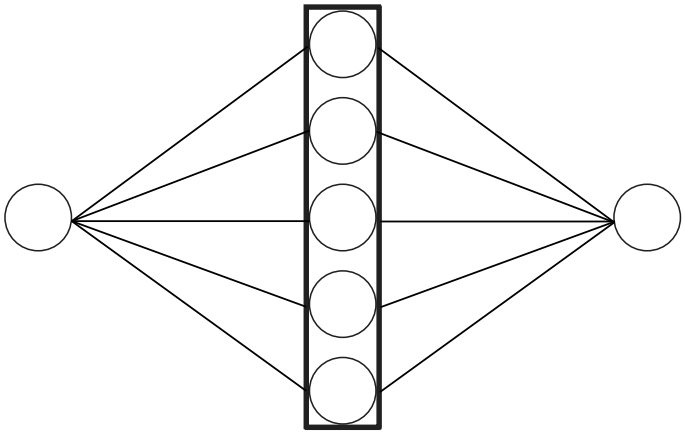

Figure 2: Illustration of the example neural network.

```
w = cell(1,numhiddenlayers+1);
b = cell(1,numhiddenlayers+1);
for i = 1:numhiddenlayers+1
if i == 1
%% input->hidden
w{i} = (2*rand(n,k(i))-1)/(n*k(i));
b{i} = zeros(1,k(i));
elseif i == numhiddenlayers+1
%% hidden->output
w{i} = (2*rand(k(i-1),1)-1)/(k(i-1)*1);
b{i} = zeros(1,1);
else
%% hidden->hidden if numhiddenlayers > 1
w{i} = (2*rand(k(i-1),k(i))-1)/(k(i-1)*k(i));
b{i} = zeros(1,k(i));
end
end
```

– The network is trained for $N = 50$ epochs. The corresponding loss reduces as shown in Fig. 3.

```
for i = 1:N
[E(i),w,b] = functions.nnfit(x,y,w,b,n);
end
```

– The feedforward, backpropagation and parameter update takes place within the function `nnfit.m`. Input data is divided into snippets and processed by the network layers. ReLU is used as the activation function. A sum of squared error is used as a loss metric and the gradient is backpropagated through the layers. The update terms are calculated and multiplied by learning rates before updating the parameters with stochastic gradient descent.

```
function [E,w,b] = nnfit(x,y,w,b,n)
```

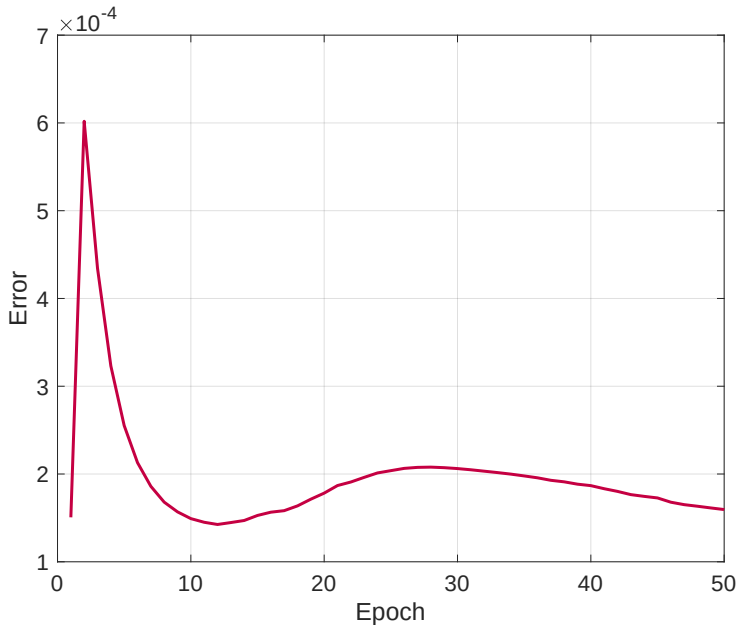

Figure 3: Loss curve.

```
L = length(x);
numlayers = length(w);
%% placeholder for hidden layers
h1 = cell(1,numlayers);
h2 = cell(1,numlayers-1);
dh2 = cell(1,numlayers-1);
dh2h1 = cell(1,numlayers-1);

%% train and update
for i = 1:L-n+1

%% get data snippet
x_i = x(i:i+n-1);
y_i = y(i+n-1);

%% feedforward operation
for j = 1:numlayers
if j == 1
h1{j} = x_i*w{j}+b{j};
h2{j} = max(h1{j},0);
elseif j == numlayers
h1{j} = h2{j-1}*w{j}+b{j};
else
h1{j} = h2{j-1}*w{j}+b{j};
h2{j} = max(h1{j},0);
end
end

%% error and backpropagation
```

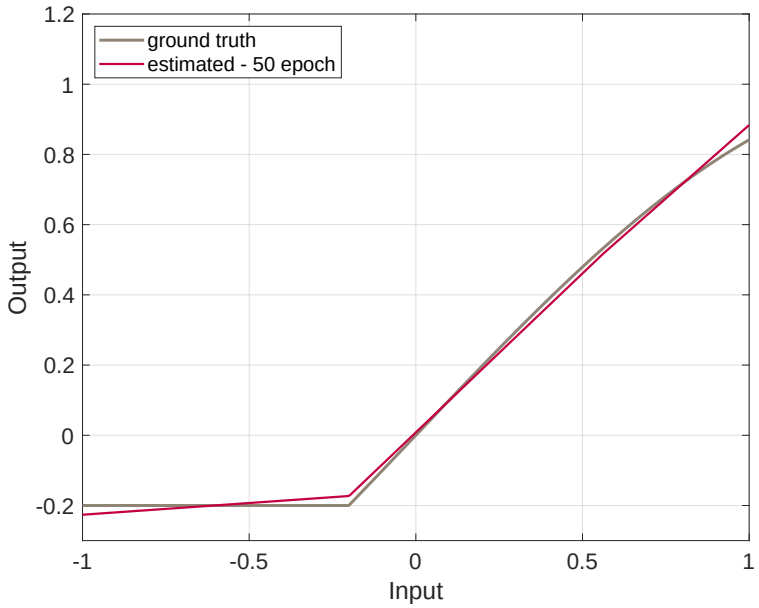

Figure 4: Ground truth vs approximation.

```
de = -2*(y_i-h1{end});
E = sum(de.^2);
for j = numlayers:-1:1
if j == numlayers
dh2{j-1} = de*w{j}';
dw{j} = h2{j-1}'*de;
db{j} = de;
elseif j == 1
dh2h1{j} = h2{j};dh2h1{j}(dh2h1{j}>0)=1;
dw{j} = x_i'*(dh2{j}.*dh2h1{j});
db{j} = dh2{j}.*dh2h1{j};
else
dh2h1{j} = h2{j};dh2h1{j}(dh2h1{j}>0)=1;
dh2{j-1} = (dh2{j}.*dh2h1{j})*w{j}';
dw{j} = h2{j-1}'*(dh2{j}.*dh2h1{j});
db{j} = dh2{j}.*dh2h1{j};
end

%% update SGD
w{j} = w{j}-0.01*dw{j};
b{j} = b{j}-0.001*db{j};
end
end
```

– The network is tested. The `nntest.m` function performs only the feedforward operation. Figure 4 shows the original function and the network approximations.

```
x = -1:0.001:1;
y = max(sin(x),-0.2);
```

```
y_est = functions.nntest(x,w,b,n);
```

– According to exercise 1 (b), the number of epochs is increased. The training with 500 epochs is performed, the error curve is plotted in Fig. 5, and the new network is evaluated as shown in Fig. 6.

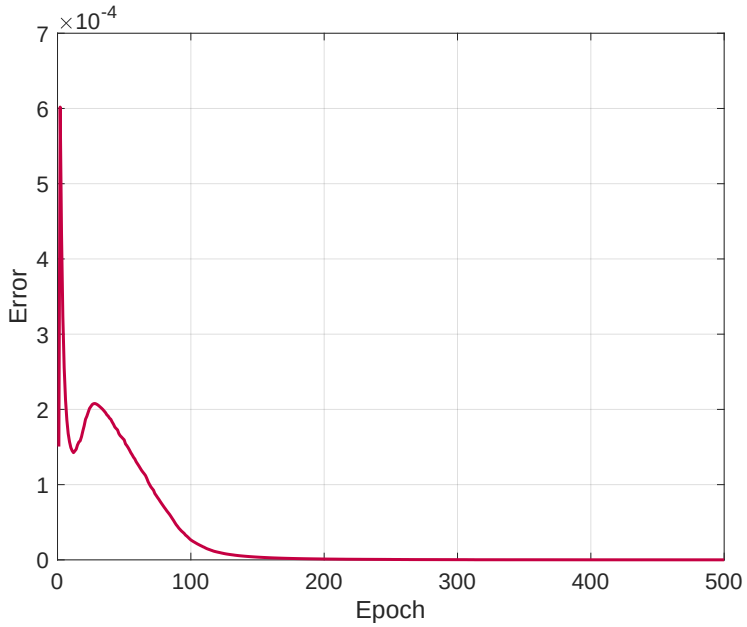

Figure 5: Loss curve.

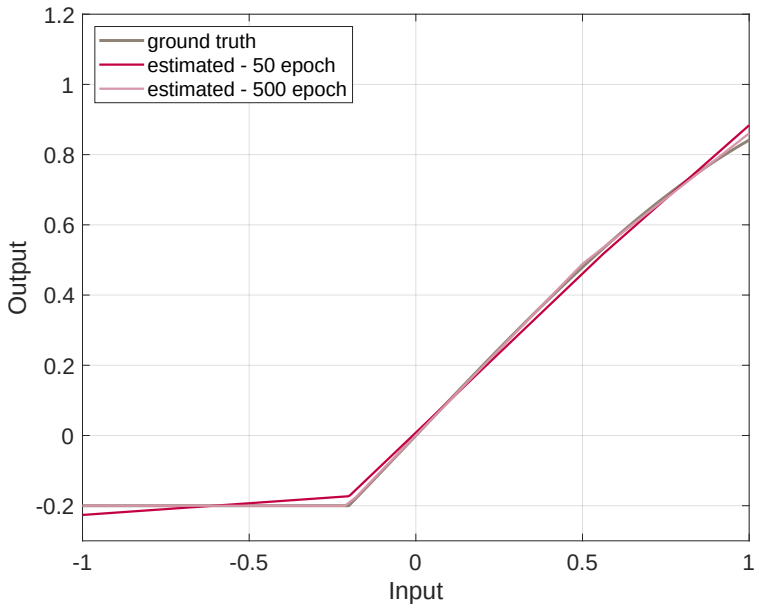

Figure 6: Ground truth vs approximations.

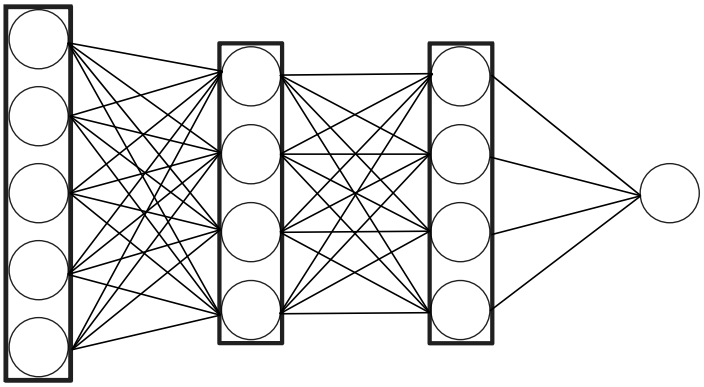

Figure 7: Illustration of the extended neural network.

– According to exercise 1 (c), the number of layers are increased. However, 2000 points are sampled instead of 1000 for this example. 2 hidden layers are selected which leads to 4 layers including the input and output. The input layer, the hidden layer, and the output layer sizes are also changed to 5, 4, 4, and 1 respectively.

```
numhiddenlayers = 2
n = 5;
k = 4*ones(1,numhiddenlayers);
```

– The network contains 3 set of weight matrices and bias vectors. The weight matrices between the input and first hidden layer, the first and second hidden layer, and the second hidden layer and output layer are 5×4, 4×4, and 4×1 respectively. They are initialized randomly and normalized by their dimensions. The bias vectors are initialized with zeros. The new network is illustrated in Fig. 7.

```
w = cell(1,numhiddenlayers+1);
b = cell(1,numhiddenlayers+1);
for i = 1:numhiddenlayers+1
if i == 1
%% input->hidden/ Weight Matrix - 5x4
w{i} = (2*rand(n,k(i))-1)/(n*k(i));
b{i} = zeros(1,k(i));
elseif i == numhiddenlayers+1
%% hidden->output/ Weight Matrix - 4x1
w{i} = (2*rand(k(i-1),1)-1)/(k(i-1)*1);
b{i} = zeros(1,1);
else
%% hidden->hidden/ Weight Matrix - 4x4
w{i} = (2*rand(k(i-1),k(i))-1)/(k(i-1)*k(i));
b{i} = zeros(1,k(i));
end
end
```

– The network is trained for $N = 100$ epochs. The corresponding loss reduces as shown in Fig. 8. The network is then tested and Fig. 9 shows the original function and the network

approximation.

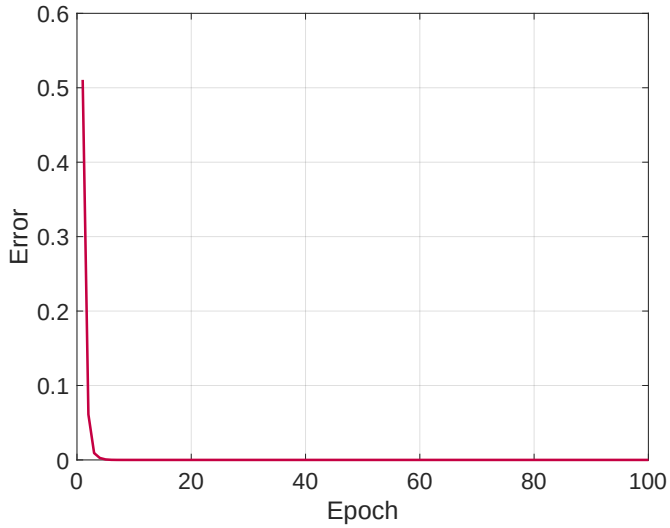

Figure 8: Loss curve.

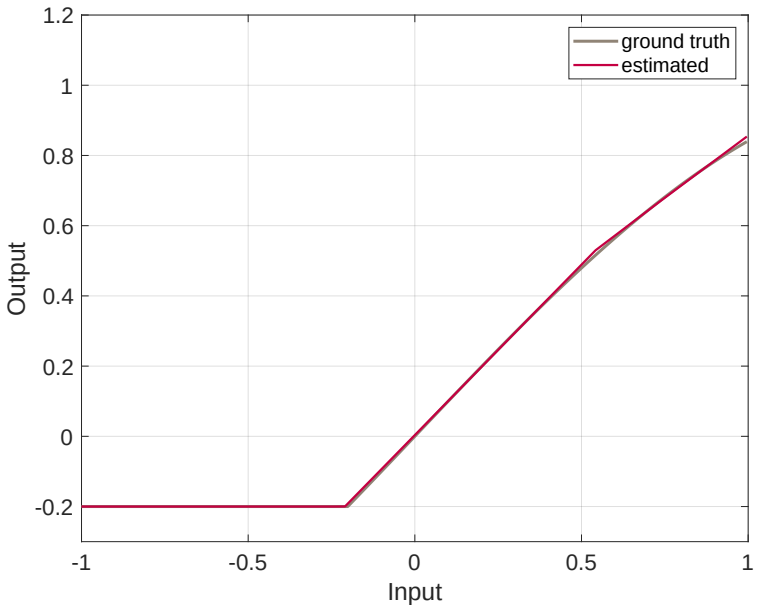

Figure 9: Ground truth vs approximation.

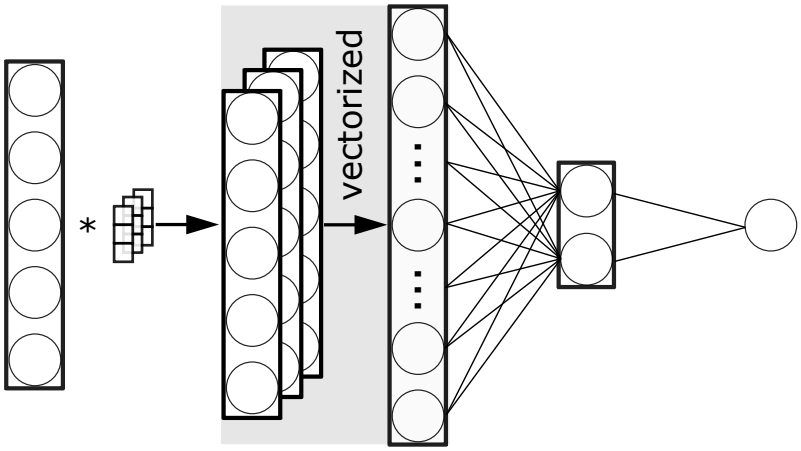

Figure 10: Illustration of the example convolutional neural network.

2. Use a convolution layer in the previous network.

   – The content of the file `dasp_chap11_ex2_CNN.m` is explained here. The data generation is done similarly as in the previous example. The network structure is defined in the next step. The layer after the input is convolutional which generates a hidden layer. The remaining layers are fully connected. The `numfclayers` denote the number of hidden layers after the first hidden layer generated by the convolutional operation. The input layer, the hidden layers, and the output layer sizes are 5, $5 \times 1 \times 3$ or 15 when vectorized, 2, and 1 respectively.

```
%% number of FC layers and input/hidden layer dims
numfclayers = 1;
n = 5;
k = 2*ones(1,numfclayers);
d = 3; % num filter groups for conv
```

   – The network contains 3 set of weight matrices/ filter coefficients and bias vectors. In the convolutional layer, 3 filter groups are used during the filtering operation having a size of $3 \times 1 \times 1$, each. The weight matrices in the fully connected layers have sizes of 15×2 and 2×1. The network weights are initialized randomly while the bias values are initialized by zero. The CNN is illustrated in Fig. 10.

```
w = cell(1,numfclayers+2);
b = cell(1,numfclayers+2);
for i = 1:numfclayers+2
if i == 1
%% input -> conv (hidden)
w{i} = (2*rand(min(n,3),1,1,d)-1)/(min(n,3)*d);
b{i} = zeros(1,3);
elseif i == 2
%% conv (hidden) -> fc (hidden)
w{i} = (2*rand(n*d,k(i-1))-1)/(n*d*k(i-1));
b{i} = zeros(1,k(i-1));
```

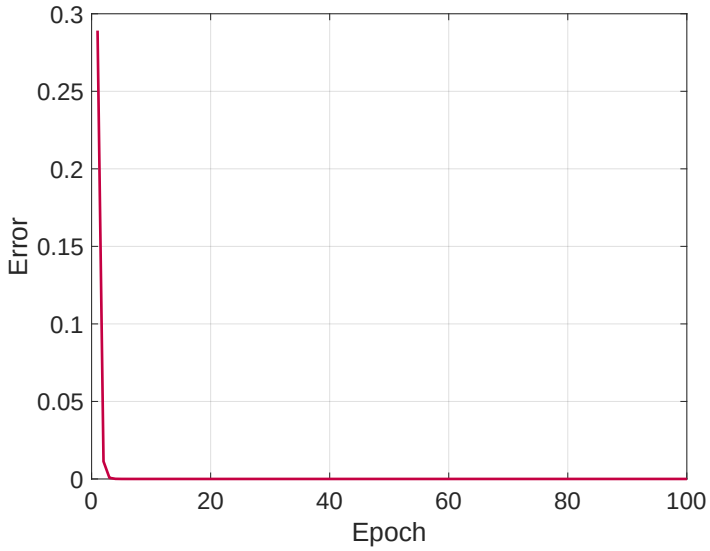

Figure 11: Loss curve.

```
elseif i == numfclayers+2
%% fc (hidden) -> output
w{i} = (2*rand(k(i-2),1)-1)/(k(i-2)*1);
b{i} = zeros(1,1);
else
%% fc (hidden) -> fc (hidden) - if numfclayers > 1
w{i} = (2*rand(k(i-2),k(i-1))-1)/(k(i-2)*k(i-1));
b{i} = zeros(1,k(i-1));
end
end
```

– The network is trained for $N = 100$ epochs. The corresponding loss reduces eventually and the network converges as shown in Fig. 11.

```
for i = 1:N
[E(i),w,b] = functions.cnnfit(x,y,w,b,n);
end
```

– The feedforward, backpropagation and parameter update takes place within the function `cnnfit.m`. Input data is divided into snippets and processed by the network layers. The first layer performs a filtering operation. Zero padding is performed and a stride of 1 is used to retain the input height and width. The corresponding output is vectorized before application of ReLU activation function. A sum of squared error is used as a loss metric and the gradient is backpropagated through the layers. The update terms are calculated and multiplied by learning rates before updating the parameters with stochastic gradient descent.

```
function [E,w,b] = cnnfit(x,y,w,b,n)
L = length(x);
numlayers = length(w);
```

```
%% placeholder for hidden layers
h1 = cell(1,numlayers);
h2 = cell(1,numlayers-1);
dh2 = cell(1,numlayers-1);
dh2h1 = cell(1,numlayers-1);

%% train and update
for i = 1:L-n+1

%% get data snippet
x_i = x(i:i+n-1);
y_i = y(i+n-1);

%% feedforward operation
for j = 1:numlayers
if j == 1
% convolution with zero padding
% conv1 function is defined separately
h1{j} = conv1(x_i,w{j},b{j});
% h1{j} vectorized before ReLU
h2{j} = max(h1{j}(:)',0);
elseif j == numlayers
h1{j} = h2{j-1}*w{j}+b{j};
else
h1{j} = h2{j-1}*w{j}+b{j};
h2{j} = max(h1{j},0);
end
end

%% error and backpropagation
de = -2*(y_i-h1{end});
E = sum(de.^2);
for j = numlayers:-1:1
if j == numlayers
dh2{j-1} = de*w{j}';
dw{j} = h2{j-1}'*de;
db{j} = de;
elseif j == 1
dh2h1{j} = h2{j};dh2h1{j}(dh2h1{j}>0)=1;
% Gradient vector reshaped to tensor
dh1 = reshape((dh2{j}.*dh2h1{j}),size(x_i,1),...
              size(x_i,2),size(w{j},4));
[~,dw{j},db{j}] = dconv1(x_i,w{j},b{j},dh1);
else
dh2h1{j} = h2{j};dh2h1{j}(dh2h1{j}>0)=1;
dh2{j-1} = (dh2{j}.*dh2h1{j})*w{j}';
dw{j} = h2{j-1}'*(dh2{j}.*dh2h1{j});
db{j} = dh2{j}.*dh2h1{j};
```

```
end

%% update SGD
w{j} = w{j}-0.01*dw{j};
b{j} = b{j}-0.001*db{j};
end
end
end
```

– The network is tested with `cnntest.m` which performs the feedforward operation. Figure 12 shows the original function and the network estimate.

```
x = -1:0.001:1;
y = max(sin(x),-0.2);
y_est = functions.cnntest(x,w,b,n);
```

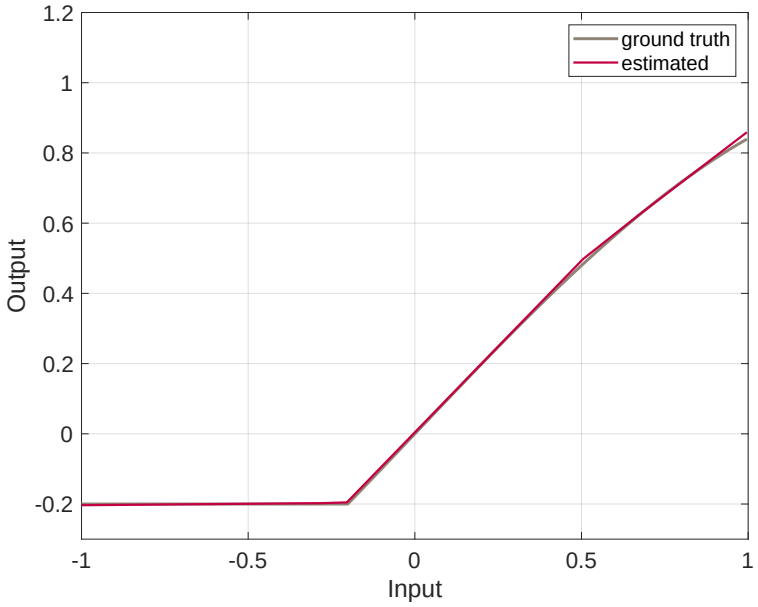

Figure 12: Ground truth vs approximation.

Table 1: Matlab scripts

1	`dasp_chap11_ex1_NN.m`
	`nnfit.m` `nntest.m`
2	`dasp_chap11_ex2_CNN.m`
	`cnnfit.m` `cnntest.m`