Single input, single output Neural Network

In[1046]:=
```
x = 0; Remove["Global`*"]; DateList[Date[]] // Most
```

Out[1046]=
```
{2020, 5, 11, 21, 39}
```

Neural networks have become very popular. However, the usual presentation makes it all seem like magic.

The goal of this lecture is to present the basics of neural networks as a modest generalization of classical approximation methods.
For example:
“Learning” a neural network is just a nonlinear least squares fitting problem.
“Backpropagation” is just automatic differentiation.
“Stochastic gradient descent” is just a MC sampling approach to approximate the steepest descent direction.

I will give you the basic mathematical ideas.
There aren’t many. Neural network research is almost exclusively “experimental”. You think you have a good idea, and you then just try it and see if it works.
Yes, this sounds like computational methods in macroeconomics. The difference is that the definition of success is stricter in the neural network community than in economics.

Define neural net node

G will be our node function. We make it evaluate vectors elementwise.

In[1047]:=
```
Clear[G]
G[list_List] := G /@ list
```

Define the “model”

Select number of nodes:

In[1053]:=
```
numnodes = 5;
```

In[1054]:=
```
avec = Table[aᵢ, {i, 1, numnodes}];
bvec = Table[bᵢ, {i, 1, numnodes}];
avec x + bvec;
```

Define generic basis

In[1057]:=
```
basis = G[avec x + bvec]
```

Out[1057]= $\{G[x\,a_1 + b_1], G[x\,a_2 + b_2], G[x\,a_3 + b_3], G[x\,a_4 + b_4], G[x\,a_5 + b_5]\}$

Specify the sigmoid function

In[1058]:=
```
G[x_] = 1 / (1 + Exp[-x]) // Together
```

Out[1058]=
$$\frac{e^x}{1 + e^x}$$

The model function will be a linearly weighted sum of the outputs of the neural nodels

In[1060]:=
```
model = Sum[c_i basis[[i]], {i, Length[basis]}]
```

Out[1060]=
$$\frac{e^{x\,a_1+b_1}\,c_1}{1 + e^{x\,a_1+b_1}} + \frac{e^{x\,a_2+b_2}\,c_2}{1 + e^{x\,a_2+b_2}} + \frac{e^{x\,a_3+b_3}\,c_3}{1 + e^{x\,a_3+b_3}} + \frac{e^{x\,a_4+b_4}\,c_4}{1 + e^{x\,a_4+b_4}} + \frac{e^{x\,a_5+b_5}\,c_5}{1 + e^{x\,a_5+b_5}}$$

Our free parameters include not only the weights on the “basis functions” but also the a[i] and b[i] coefficients that appear nonlinearly.

Define the set of variables and initial guesses

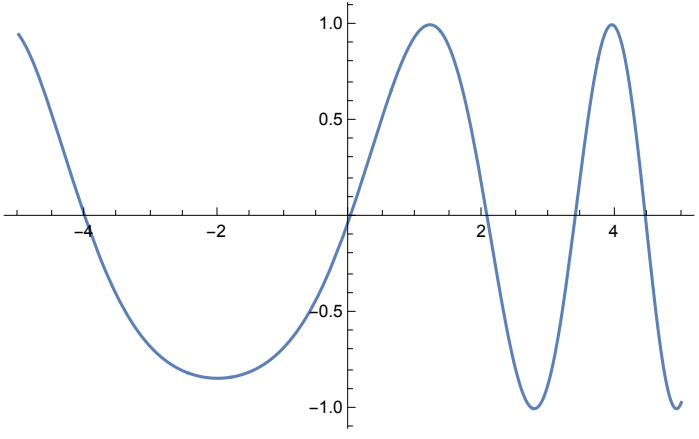
```
vars = Table[{a_i, b_i, c_i}, {i, numnodes}] // Flatten; init = vars - vars + 1;
varsin = {vars, init} // Transpose
```

Out[1063]=
$$\{\{a_1, 1\}, \{b_1, 1\}, \{c_1, 1\}, \{a_2, 1\}, \{b_2, 1\}, \{c_2, 1\}, \{a_3, 1\}, \{b_3, 1\}, \{c_3, 1\}, \{a_4, 1\}, \{b_4, 1\}, \{c_4, 1\}, \{a_5, 1\}, \{b_5, 1\}, \{c_5, 1\}\}$$

Fit an example function

In[1096]:=
```
f[x_] = Sin[ x x / 4 + x];
{xmin, xmax} = {-5, 5};
Plot[f[x], {x, xmin, xmax}, PlotRange → All]
```

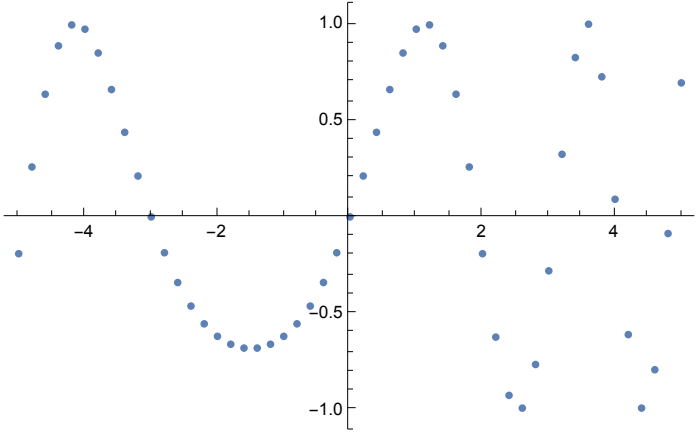
Out[1098]=



Create set of 51 data points.

In[1134]:=
```
mpts = 25;  npts = 2 mpts + 1;
xpts = (Range[npts] - mpts) / 5. - 0.2; ypts = f /@ xpts;
data = {xpts, ypts} // N // Transpose;
ListPlot[data]
```

Out[1137]=

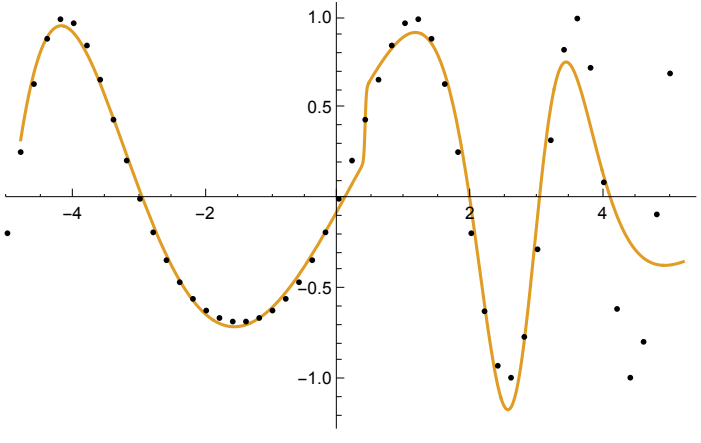


In[1138]:=
```
fit = FindFit[data, model, varsin, x, MaxIterations → 500, Method -> "NMinimize"]
```

Out[1138]=
$$\{a_1 \to -0.456123,\ b_1 \to -3.5786,\ c_1 \to -573.93,\ a_2 \to 107.46,\ b_2 \to -42.7104,$$
$$c_2 \to 0.395407,\ a_3 \to -2.78671,\ b_3 \to 8.04197,\ c_3 \to 203.245,\ a_4 \to -0.554452,$$
$$b_4 \to -3.24477,\ c_4 \to 309.668,\ a_5 \to -2.90008,\ b_5 \to 8.38972,\ c_5 \to -199.308\}$$

In[1139]:=
```
modelf = Function[{x}, Evaluate[model /. fit]];
Plot[{f[x], modelf[t]}, {t, xmin, xmax}, PlotRange → All, Epilog → Map[Point, data]]
```

Out[1140]=

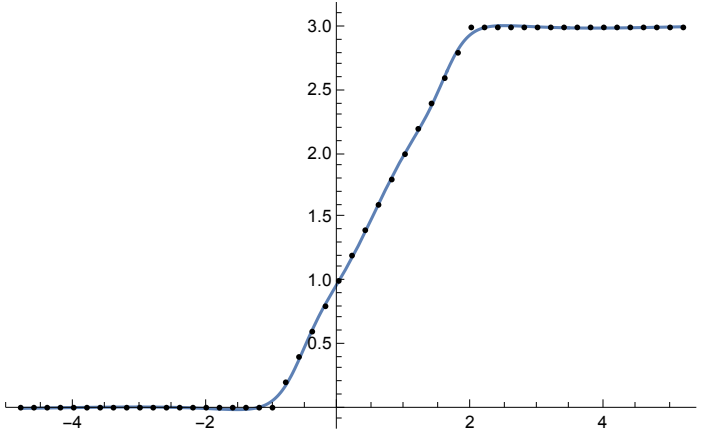


Define script

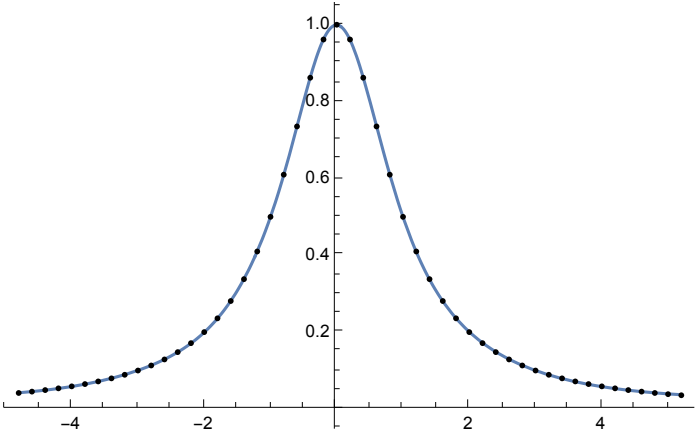
In[1151]:=
```
script := (
  xpts = (Range[51] - 25) / 5;
  {xmin, xmax} = {Min[xpts], Max[xpts]};
  ypts = f /@ xpts;
  data = {xpts, ypts} // Transpose;
  fit = FindFit[data, model, varsin, x, Method → NMinimize];
  modelf = Function[{x}, Evaluate[model /. fit]];
  Plot[modelf[t], {t, xmin, xmax}, Epilog → Map[Point, data]] // Print;
  (* basisfit=basis/.fit;
  basisfit//Print;
  Plot[basisfit,{x,xmin,xmax}]//Print *);
 )
```

Fit example functions

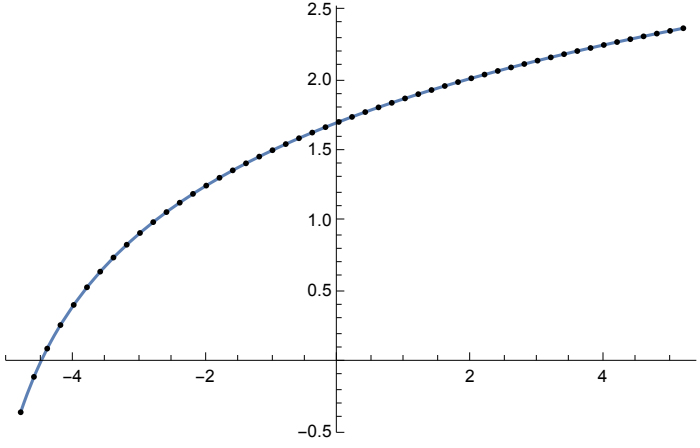
In[1152]:=
```
f[x_] = Min[Max[x + 1, 0], 3];
script
```



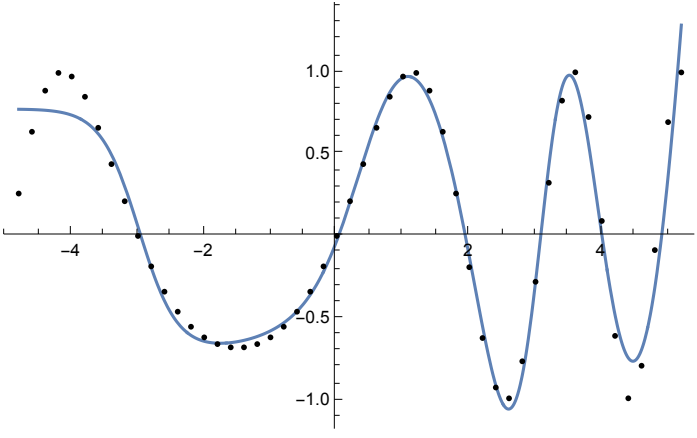
In[1145]:=
```
f[x_] = 1 / (1 + x^2);
script
```



In[1147]:=
```
f[x_] = Log[x + 5.5];
script
```



In[1149]:=
```
f[x_] = Sin[x^2 / 3 + x];
script
```



Comments

This is a nonlinear approximation approach that appears to be able to handle a wide variety of functions.