Data Analysis in Astronomy and Physics

Lecture 15: Introduction to Neural Networks

M. Röllig

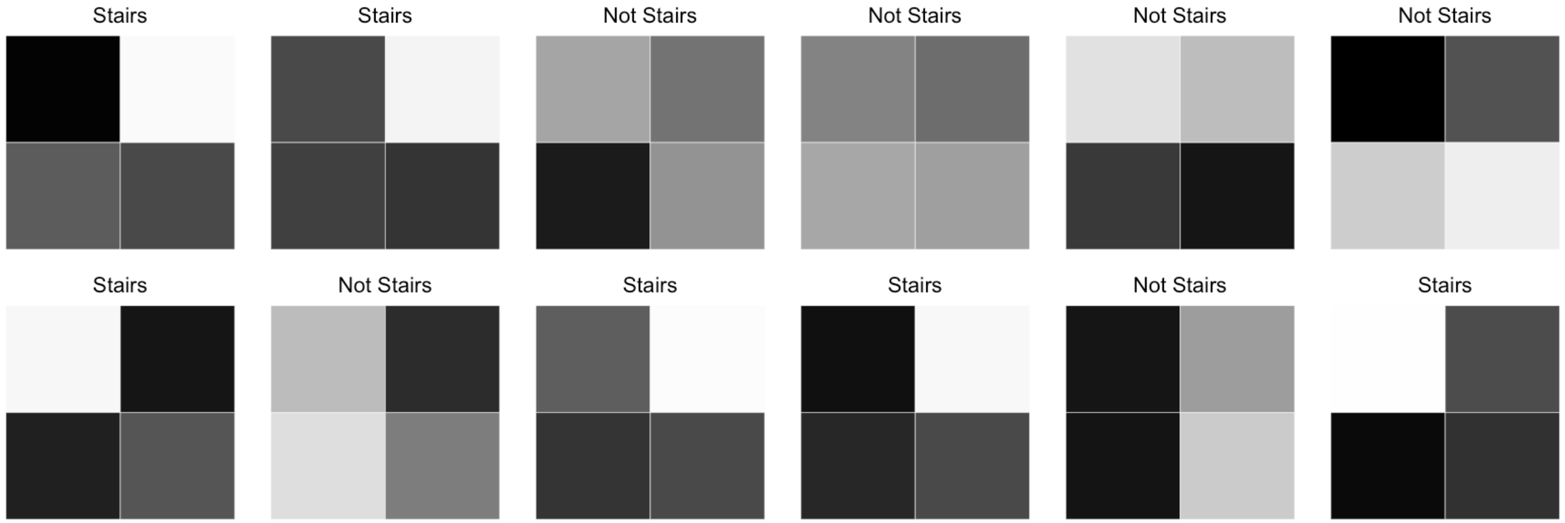
Introduction to Neural Networks

Based on this great BLOG:

https://gormanalysis.com/introduction-to-neural-networks/

Motivation

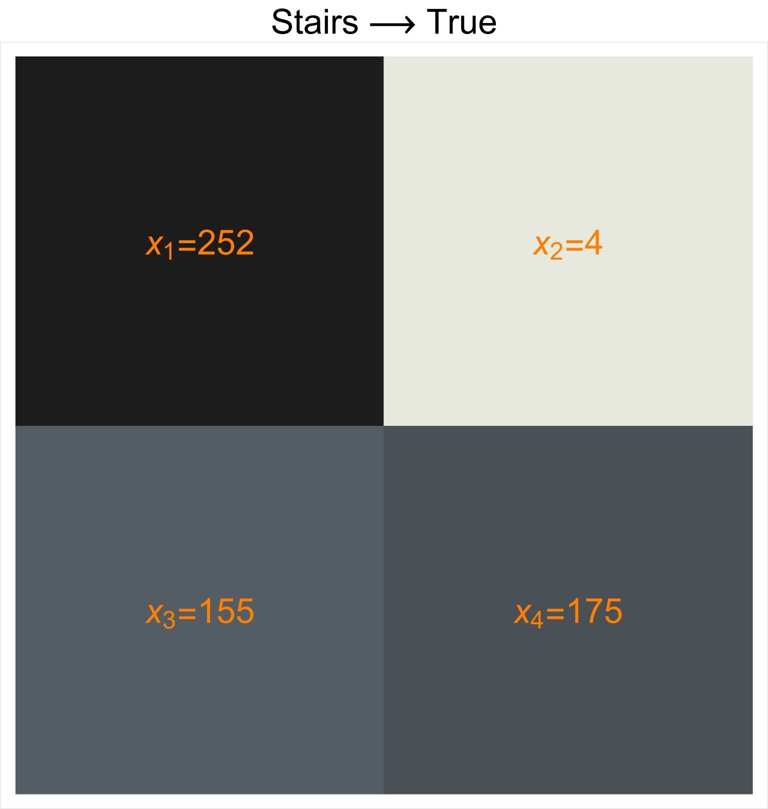
“We’ll start with a motivational problem. Here we have a collection of grayscale images, each a 2×2 grid of pixels where each pixel has an intensity value between 0 (white) and 255 (black). The goal is to build a model that identifies images with a “stairs” pattern.”



Introduction to Neural Networks

```
train = Import[ … ];
```

The data consists of a list of (pixel) brightness values $\{x_1, x_2, x_3, x_4\}$ and a classification (True or False). To display the vector as an image we partition the vector into a 2x2 matrix.

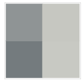
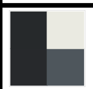
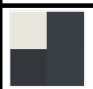
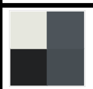
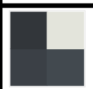


Introduction to Neural Networks

We create a few helper functions to visualize the data and randomly select a few entries from the data:

```
In[]:= fig[l_] := ArrayPlot[Partition[l, 2], ColorFunctionScaling → False, ColorFunction →
          Function[x, ColorData[{"GrayTones", "Reversed"}][x/256.]], ImageSize → 40, FrameTicks → None, FrameStyle → LightGray];
       display[line_List] := Flatten@{line[[1 ;; 5]], fig[line[[2 ;; 5]]], If[line[[-1]] == 1, True, False]}
       Grid[display[#] & /@ RandomSample[train, 8], Frame → All]
```

Out[]=

267	148	190	58	79		False
355	7	188	187	191		True
79	218	2	201	219		True
343	93	45	120	42		False
319	235	1	234	167		True
288	6	206	215	204		True
170	5	169	244	182		True
293	217	9	202	188		True

Single Layer Perceptron

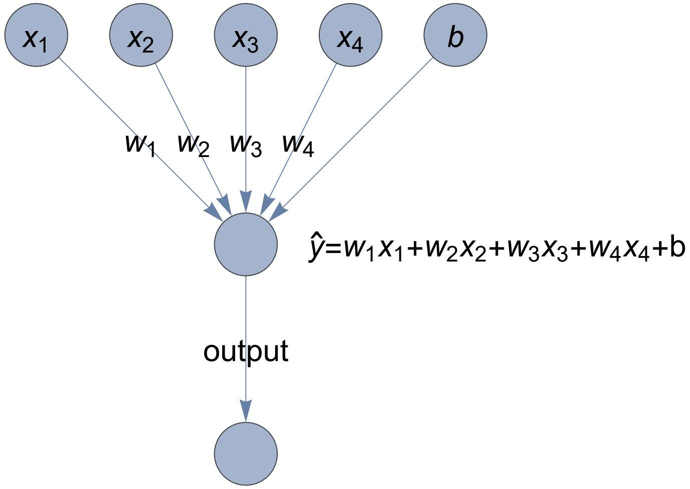
We start with the most simple approach a so-called single layer perceptron. A perceptron uses a weighted linear combination of the inputs to return a prediction score. If the prediction score exceeds a selected threshold, the perceptron predicts True. Otherwise it predicts False.

Out[∘]//TraditionalForm=

$$f(x) = \left(\begin{cases} 1 & \text{if } w_1 x_1 + \ldots + w_n x_n > \text{threshold} \\ 0 & \text{otherwise} \end{cases} \right)$$

Single Layer Perceptron

Out[]=



To start with an example, we use the following weights: $w = \{-0.0019, -0.0016, .0020, 0.0023\}$ and $b = 0.0003$.

Single Layer Perceptron

```
In[◦]:= Clear[perc]
        perc[x_List, w_List, b_?NumberQ] /; Length[x] == Length[w] := Module[{y = w.x + b}, Piecewise[{{1, y > 0}, {0, True}}]]
        result[x_List, w_List, b_?NumberQ] /; Length[x] == Length[w] := Module[{res, im, lab},
          lab = Column[{
              Row[{"x=", x}],
              Row[{"ŷ=", w.x + b, " ⟶ ", perc[x, w, b] /. {0 → "Not stairs", 1 → "Stairs"}}]}];
          Labeled[Show[fig[x], ImageSize → 200], lab, Top]]
```

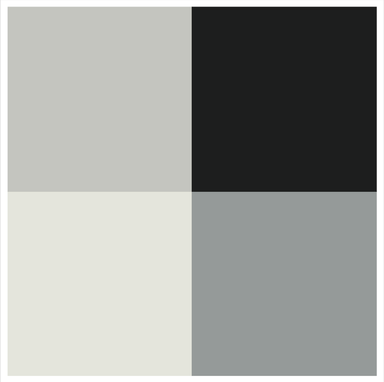
```
In[◦]:= w = {-0.0019, -0.0016, .0020, 0.0023};
        b = 0.0003;
```

Single Layer Perceptron

```
In[]:= xlist = {{47, 250, 8, 88}, {115, 138, 80, 88}, {225, 13, 210, 144}, {181, 5, 157, 152}, {0, 212, 169, 167}, {12, 61, 158, 248}};
       result[#, w, b] & /@ xlist
```

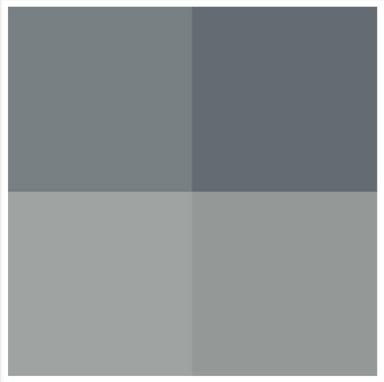
Out[]= {

```
x={47, 250, 8, 88}
ŷ=-0.2706 ⟶ Not stairs
```



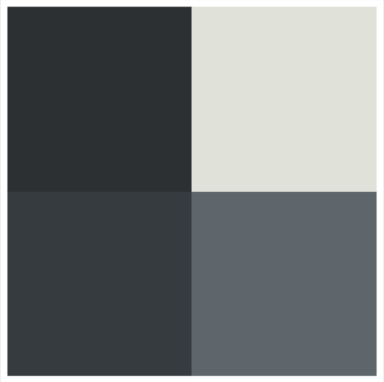
,

```
x={115, 138, 80, 88}
ŷ=-0.0766 ⟶ Not stairs
```



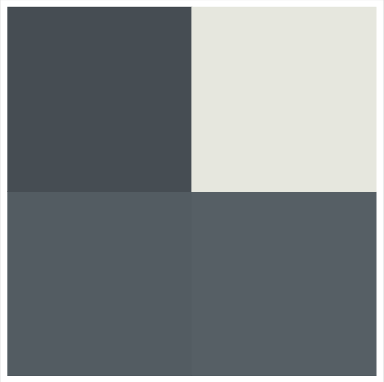
,

```
x={225, 13, 210, 144}
ŷ=0.3032 ⟶ Stairs
```



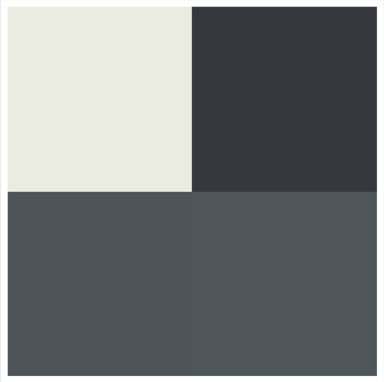
,

```
x={181, 5, 157, 152}
ŷ=0.312 ⟶ Stairs
```



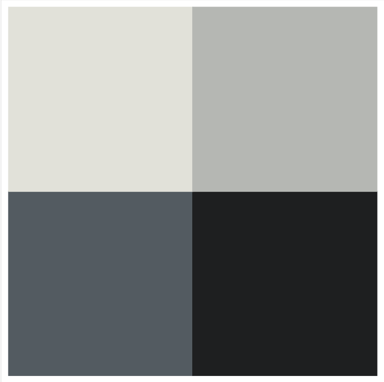
,

```
x={0, 212, 169, 167}
ŷ=0.3832 ⟶ Stairs
```



,

```
x={12, 61, 158, 248}
ŷ=0.7663 ⟶ Stairs
```



}

Single Layer Perceptron

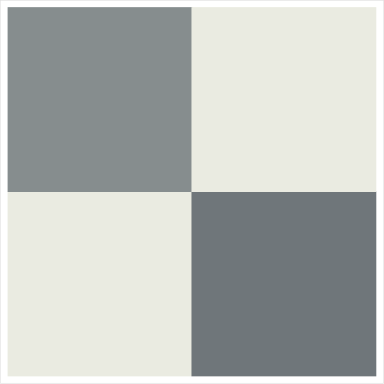
This is not too bad. All the stairs patterns have darkly shaded pixels in the bottom row which supports the larger, positive coefficients for x3 and x4. There are a few problems though:

- The output $\hat{y}$ corresponds to the certainty (likelihood) that the pattern shows stairs. Mathematically this is wrong though, since it can be a number larger than 1!
- The model can’t capture the non-linear relationship. See for example:

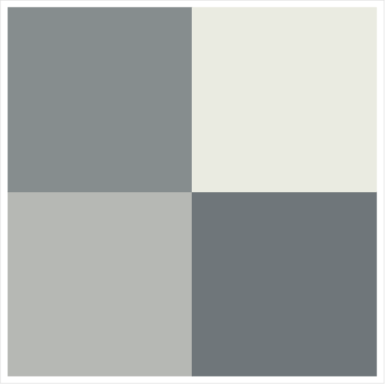
Example A

Out[•]=

x={100, 0, 0, 125}
$\hat{y}$=0.0978 ⟶ Stairs



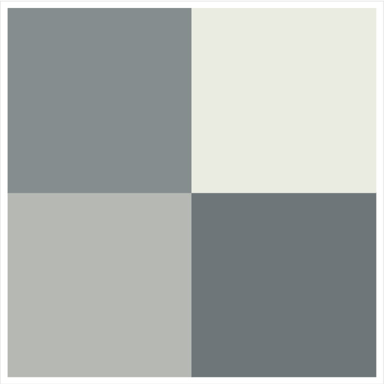
x={100, 0, 60, 125}
$\hat{y}$=0.2178 ⟶ Stairs



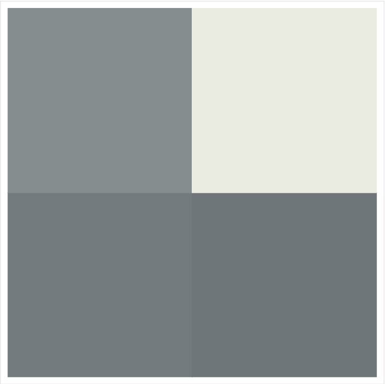
Example B

Out[•]=

x={100, 0, 60, 125}
$\hat{y}$=0.2178 ⟶ Stairs



x={100, 0, 120, 125}
$\hat{y}$=0.3378 ⟶ Stairs



In both examples we increased the value of x3 by 60. This gave an increase of $w_3 \times \Delta x_3 = 0.12$. But looking at the patterns, the increase in Example A should be larger compared to Example B! This is due to the linearity in $\hat{y}$.

Single Layer Perceptron

Seems ok. Now we try to re-discover the weights. For this we build a cost function of the form:

Out[∘]//TraditionalForm=

$$\text{cost} = \sum_{i=1}^{n} (\text{model}_i - \text{data}_i)^2$$

here, model is $\hat{y} = w_1\, x_1 + w_2\, x_2 + w_3\, x_3 + w_4\, x_4 + b$ and data is the classification 0 or 1 from the training data.

In[∘]:=

```
cost[w_List, b_?NumberQ] := Module[{},
         1
  ────────────── Total@Table[(perc[train[[i, 2 ;; 5]], w, b] - train[[i, -1]])^2, {i, 1, Length[train]}]]
  2 Length[train]
cost[w1_, w2_, w3_, w4_, b_?NumberQ] := Module[{},
         1
  ────────────── Total@Table[(perc[train[[i, 2 ;; 5]], {w1, w2, w3, w4}, b] - train[[i, -1]])^2, {i, 1, Length[train]}]]
  2 Length[train]
fit = NMinimize[
   cost[w1, w2, w3, w4, b1]
   ,
   {{w1, -.01, .01}, {w2, -.01, .01}, {w3, -.01, .01}, {w4, -.01, .01}, {b1, -.01, .01}},
   (*Method→"NelderMead",*)MaxIterations → 500
  ]
```

Out[∘]=

```
{0.0575, {w1 → -0.0300972, w2 → -0.0285827, w3 → 0.0169892, w4 → 0.0243446, b1 → -0.234916}}
```

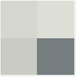
Single Layer Perceptron

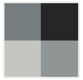
Comparing this with our initial guess of weights, we see that our optimization seems to result a different set of weights. Let’s see how they perform

```
In[ ]:= w
        wfit = {{w1, w2, w3, w4}, b1} /. fit[[2]]

Out[ ]= {-0.0019, -0.0016, 0.002, 0.0023}

Out[ ]= {{-0.0300972, -0.0285827, 0.0169892, 0.0243446}, -0.234916}
```

```
In[•]:= Row[{TableForm[{perc[Most@#, Sequence @@ wfit], fig[Most@#], perc[Most@#, w, b]} & /@ RandomSample[train[[All, 2 ;; 6]], 10],
          TableHeadings → {None, {"our \nclassific.", "case", "initial \nclassific."}}], Spacer[50],
        TableForm[{perc[Most@#, Sequence @@ wfit], fig[Most@#], perc[Most@#, w, b]} & /@ RandomSample[train[[All, 2 ;; 6]], 10],
          TableHeadings → {None, {"our \nclassific.", "case", "initial \nclassific."}}]}]
```

Out[•]=

our classific.	case	initial classific.
0		1
1		1
1		1
0		1
0		1
1		1
1		1
1		1
1		1
1		1

our classific.	case	initial classific.
1		1
1		1
1		1
1		1
1		1
1		1
1		1
0		1
0		0
1		1

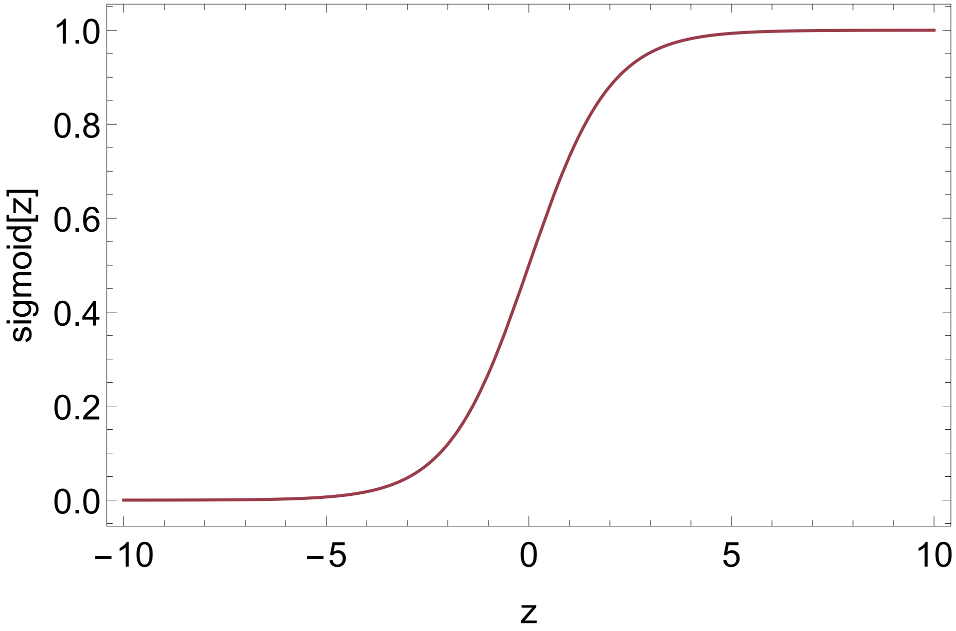
Single Layer Perceptron with Sigmoid activation function

To address the problems mentioned above, we will wrap our perceptron within a sigmoid function (and subsequently choosing different weights). Since the sigmoid function is bound between 0 and 1 it can be used to model the probability of a binary event

Out[ ]//TraditionalForm=

$$\text{sigmoid}(z) = \frac{1}{1 + \exp(-z)}$$

Out[ ]=



Single Layer Perceptron with Sigmoid activation function

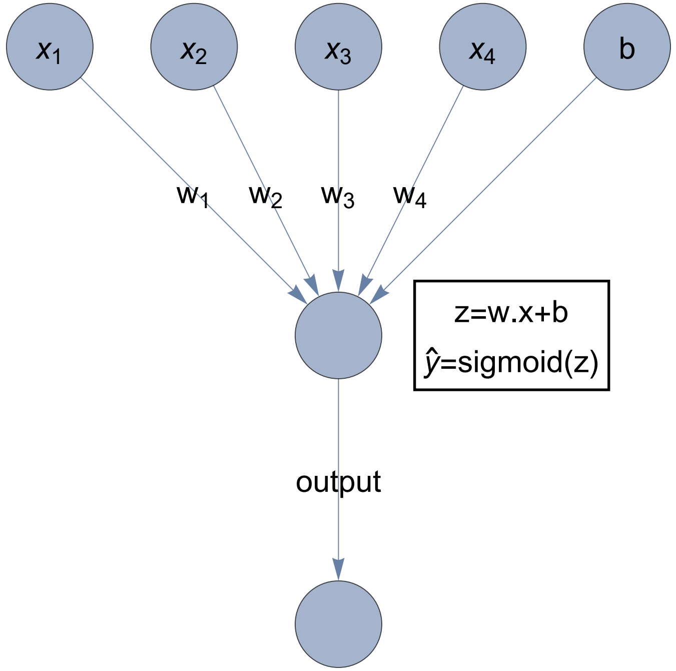
So our new model looks like this:

$z = \text{w}.x + \text{b} = w_1 x_1 + w_2 x_2 + w_3 x_3 + w_4 x_4 + \text{b}$

Out[ ]=

$\hat{y} = \text{sigmoid}(z) = \frac{1}{1+\exp(-z)}$

Out[ ]=



Single Layer Perceptron with Sigmoid activation function

We now have a linear perceptron with a sigmoid “activation function”:

Out[∘]//TraditionalForm=

$$f(x) = \left(\begin{cases} 1 & \text{if } \hat{y} > 0.5 \\ 0 & \text{otherwise} \end{cases} \right)$$

In[∘]:=

```
Clear[perc2, result2]
perc2[x_List, w_List, b_?NumberQ] /; Length[x] == Length[w] :=
 Module[{y = w.x + b}, Piecewise[{{1, LogisticSigmoid[y] > 0.5}, {0, True}}]]
result2[x_List, w_List, b_?NumberQ] /; Length[x] == Length[w] := Module[{res, im, lab},
  lab = Column[{
     Row[{"x=", x}],
     Row[{"ŷ=", Chop@LogisticSigmoid[w.x + b], " ⟶ ", perc2[x, w, b] /. {0 → "Not stairs", 1 → "Stairs"}}]}];
  Labeled[Show[fig[x], ImageSize → 200], lab, Top]]
```

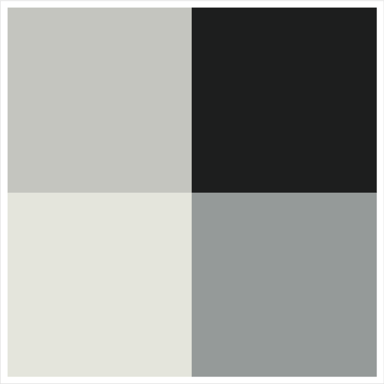
Let’s test this with a new guessed set of weights:

In[∘]:=

```
{w2, b2} = {{-0.196, -0.189, 0.136, 0.109}, -0.006};
```

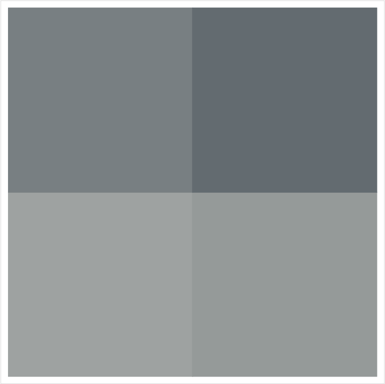
Out[∘]= {

x={47, 250, 8, 88}
ŷ=0 ⟶ Not stairs



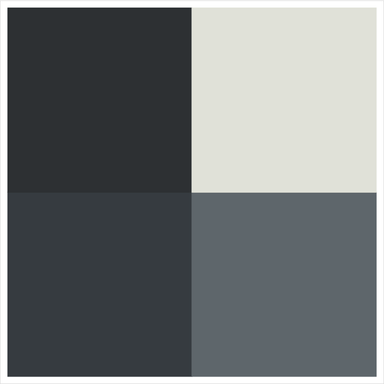
,

x={115, 138, 80, 88}
ŷ=0 ⟶ Not stairs



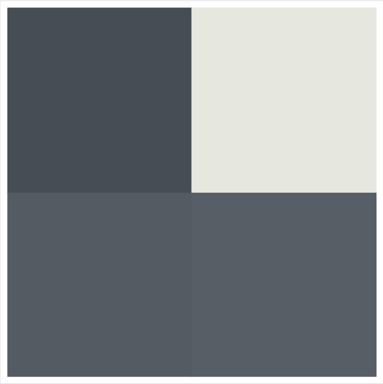
,

x={225, 13, 210, 144}
ŷ=0.0905449 ⟶ Not stairs



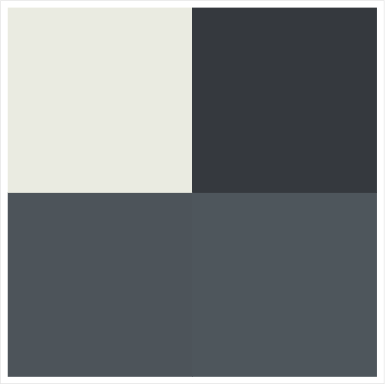
,

x={181, 5, 157, 152}
ŷ=0.816528 ⟶ Stairs



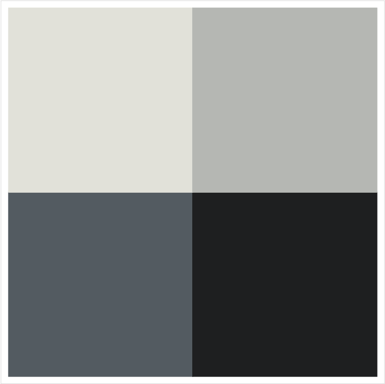
,

x={0, 212, 169, 167}
ŷ=0.752688 ⟶ Stairs



,

x={12, 61, 158, 248}
ŷ=1. ⟶ Stairs



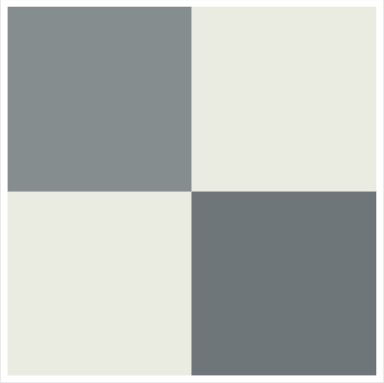
}

Now, $\hat{y}$ is always a number between 0 and 1 and can be interpreted in terms of a probability. How about our second concern?

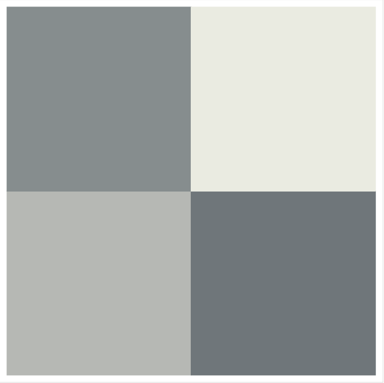
Example A

```
In[ ]:= Row[{result2[{100, 0, 0, 125}, w2, b2], Spacer[50], result2[{100, 0, 60, 125}, w2, b2]}]
```

```
Out[ ]=
x={100, 0, 0, 125}
ŷ=0.00251993 ⟶ Not stairs
```



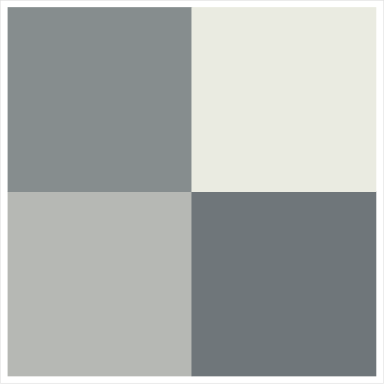
```
x={100, 0, 60, 125}
ŷ=0.898348 ⟶ Stairs
```



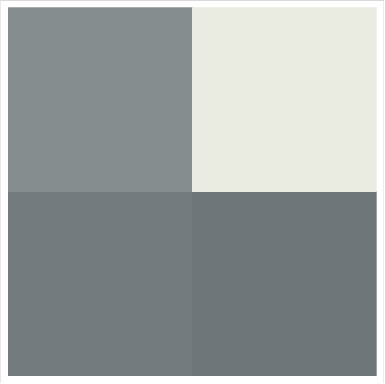
Example B

Out[ ]=

x={100, 0, 60, 125}
ŷ=0.898348 ⟶ Stairs



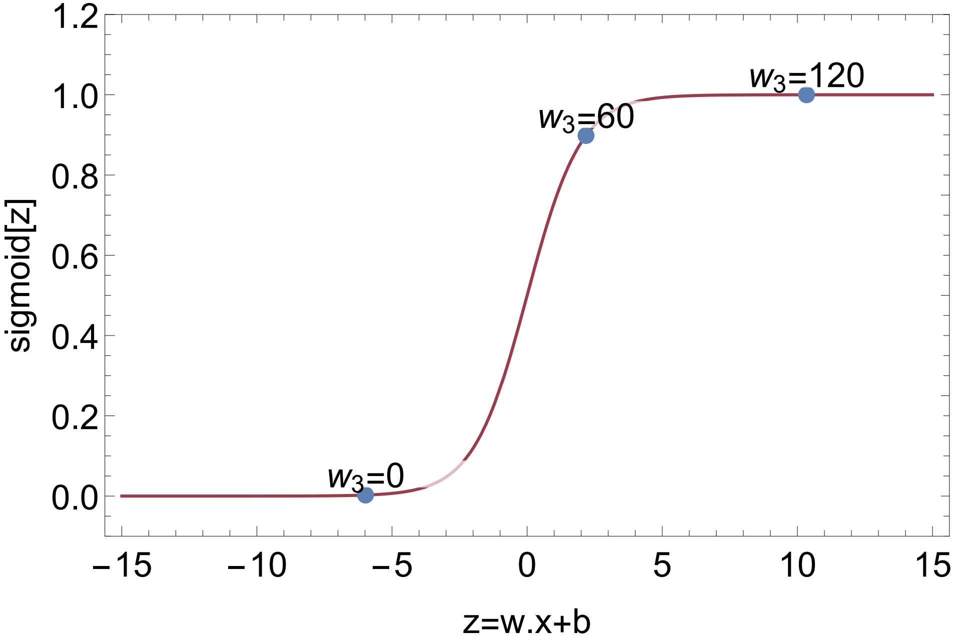
x={100, 0, 120, 125}
ŷ=0.999968 ⟶ Stairs



Single Layer Perceptron with Sigmoid activation function

The non-linearity of the sigmoid function causes the perceptron to fire (activate) when we increase w_3 to 60. The second increase gives only a little additional likelihood.

Out[ ]=



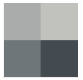
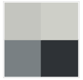
Single Layer Perceptron with Sigmoid activation function

Again, let us try to find the best set of weights for our new model:

```
In[∘]:= Clear[w2];
        cost2[w_List, b_?NumberQ] := Module[{},
               1
          ─────────────── Total@Table[(perc2[train[[i, 2 ;; 5]], w, b] - train[[i, -1]])^2, {i, 1, Length[train]}]]
          2 Length[train]
        cost2[w1_, w2_, w3_, w4_, b_?NumberQ] := Module[{},
               1
          ─────────────── Total@Table[(perc2[train[[i, 2 ;; 5]], {w1, w2, w3, w4}, b] - train[[i, -1]])^2, {i, 1, Length[train]}]]
          2 Length[train]
        fit2 = NMinimize[
          cost2[w1, w2, w3, w4, b1]
          ,
          {{w1, -.01, .01}, {w2, -.01, .01}, {w3, -.01, .01}, {w4, -.01, .01}, {b1, -.01, .01}},
          (*Method→"NelderMead",*)MaxIterations → 500
         ]
        wfit2 = {{w1, w2, w3, w4}, b1} /. fit2[[2]];
Out[∘]= {0.0575, {w1 → -0.0300972, w2 → -0.0285827, w3 → 0.0169892, w4 → 0.0243446, b1 → -0.234916}}
```

Single Layer Perceptron with Sigmoid activation function

Out[•]=

our classific.	case	initial classific.	our classific.	case	initial classific.
1		1	0		0
1		1	0		0
0		0	1		1
1		1	0		0
0		0	0		0
1		1	1		1
0		0	1		1
1		1	0		0
1		1	1		1
1		1	0		0

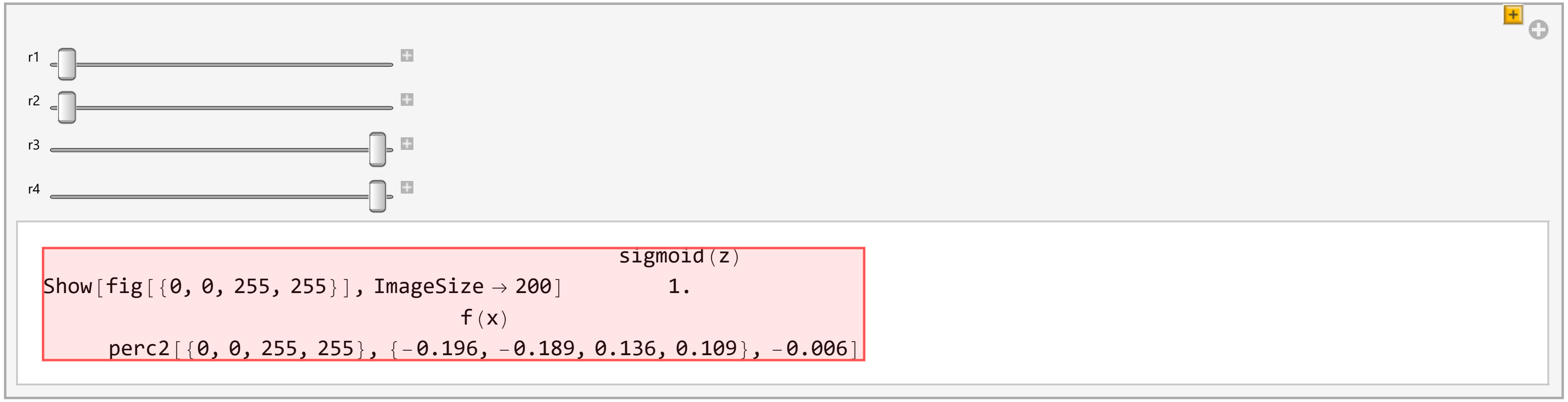
Single Layer Perceptron with Sigmoid activation function

Some issues remain:

- the linearity in w_i will favor darkly shaded stairs. What if we want to identify lightly shaded stairs?
- The model does not account for variable interaction. Assume the bottom row of an image is black. If the top left pixel is white, darkening the top right pixel should increase the probability of stairs. If the top left pixel is black, darkening the top right pixel should decrease the probability of stairs. In other words, increasing x3 should potentially increase or decrease $\hat{y}$ depending on the values of the other variables. Our current model has no way of achieving this.

Single Layer Perceptron with Sigmoid activation function

Out[∘]=



```
                                          sigmoid(z)
Show[fig[{0, 0, 255, 255}], ImageSize → 200]        1.
                               f(x)
    perc2[{0, 0, 255, 255}, {-0.196, -0.189, 0.136, 0.109}, -0.006]
```

Multi-Layer Perceptron with Sigmoid activation function

Example 1: Identify the stairs pattern

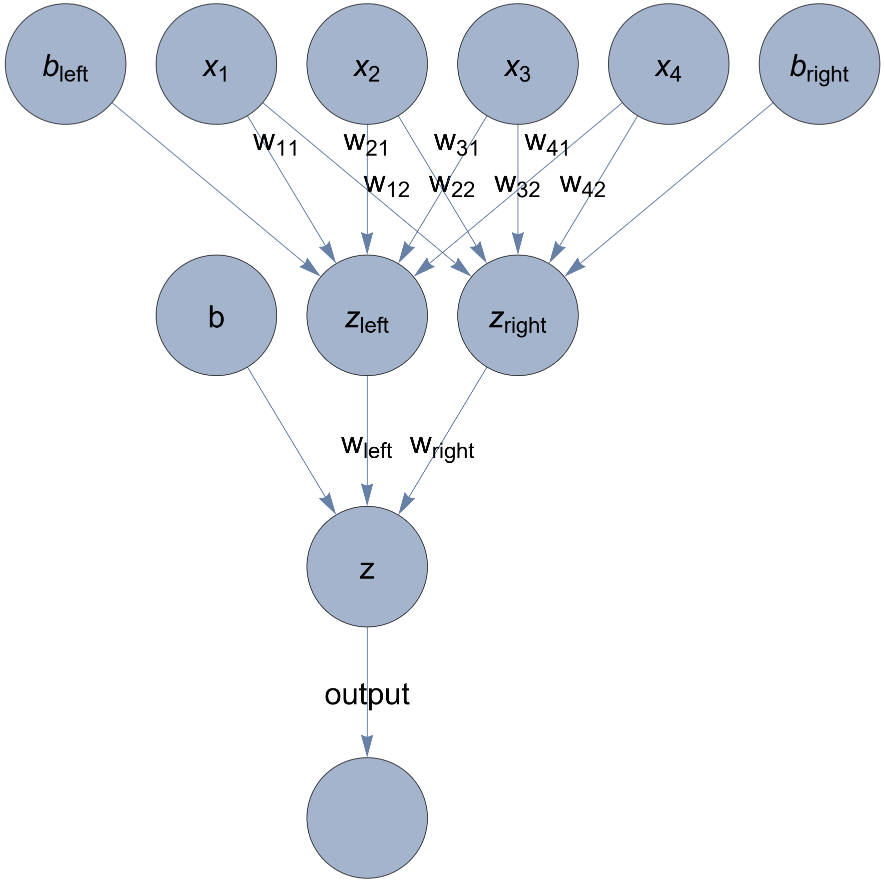
We can solve the issues by adding a seconde layer to our perceptron model. We’ll construct a number of base models like the one above, but then we’ll feed the output of each base model as input into another perceptron. This model is in fact a vanilla neural network.

- Build a model that fires when “left stairs” are identified $\hat{y}_{\text{left}}$
- Build a model that fires when “right stairs” are identified $\hat{y}_{\text{right}}$
- Add the score of the base models so that the final sigmoid function only fires if both $\hat{y}_{\text{left}}$ and $\hat{y}_{\text{right}}$ are large

Example 1: Identify the stairs pattern

It look’s like this:

Out[ ]=



Example 1: Identify the stairs pattern

```
In[∘]:= sol = Flatten[{{W11, W21, W31, W41}, {W12, W22, W32, W42}, Bleft, Bright, {Wleft, Wright}, B1} =
           {{0.002, -0.050, 0.012, 0.012}, {-0.050, 0.002, 0.012, 0.012}, -0.05, -0.05, {3, 3}, -1}]

Out[∘]= {0.002, -0.05, 0.012, 0.012, -0.05, 0.002, 0.012, 0.012, -0.05, -0.05, 3, 3, -1}
```

```
In[∘]:= Clear[multiperc2, multiresult2]
        multiperc2[x_List, w11_, w21_, w31_, w41_, w12_, w22_, w32_, w42_, bleft_, bright_, wleft_, wright_, b_] := Module[
          {y1 = {w11, w21, w31, w41}.x + bleft,
           y2 = {w12, w22, w32, w42}.x + bright, y}, y = wleft * Piecewise[{{1, LogisticSigmoid[y1] > 0.5}, {0, True}}] +
             wright * Piecewise[{{1, LogisticSigmoid[y2] > 0.5}, {0, True}}] + b;
          Piecewise[{{1, LogisticSigmoid[y] > 0.5}, {0, True}}]]

        multiresult2[x_List, w11_, w21_, w31_, w41_, w12_, w22_, w32_, w42_, bleft_, bright_, wleft_, wright_, b_?NumberQ] /;
          Length[x] == Length[w] := Module[{res, im, lab, y1, y2, y},
          y1 = {w11, w21, w31, w41}.x + bleft;
          y2 = {w12, w22, w32, w42}.x + bright;
          y = wleft * Piecewise[{{1, LogisticSigmoid[y1] > 0.5}, {0, True}}] +
             wright * Piecewise[{{1, LogisticSigmoid[y2] > 0.5}, {0, True}}] + b;
          lab = Column[{
              Row[{"x=", x}],
              Row[{"ŷleft=", Chop@LogisticSigmoid[y1], ", ŷright=", Chop@LogisticSigmoid[y2]}],
              Row[{"ŷ=", Chop@y, " ⟶ ", Piecewise[{{1, LogisticSigmoid[y] > 0.5}, {0, True}}] /. {0 → "Not stairs", 1 → "Stairs"}}]}];
          Labeled[Show[fig[x], ImageSize → 200], lab, Top]]
```

Example 1: Identify the stairs pattern

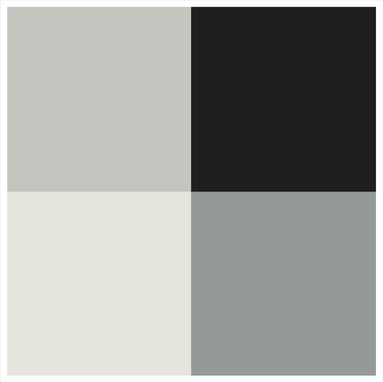
In[∘]:=
```
multiresult2[#, W11, W21, W31, W41, W12, W22, W32, W42, Bleft, Bright, Wleft, Wright, B1] & /@ xlist
```

Out[∘]= {

x = {47, 250, 8, 88}

$\hat{y}_{\text{left}}$=0.0000123234, $\hat{y}_{\text{right}}$=0.321257

$\hat{y}$=-1 ⟶ Not stairs

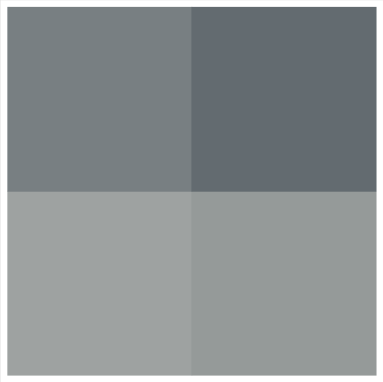


,

x = {115, 138, 80, 88}

$\hat{y}_{\text{left}}$=0.00897764, $\hat{y}_{\text{right}}$=0.0290855

$\hat{y}$=-1 ⟶ Not stairs

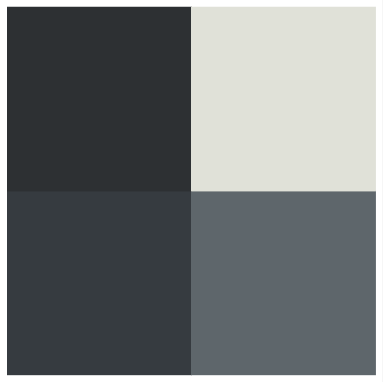


,

x = {225, 13, 210, 144}

$\hat{y}_{\text{left}}$=0.981978, $\hat{y}_{\text{right}}$=0.00088769

$\hat{y}$=2 ⟶ Stairs

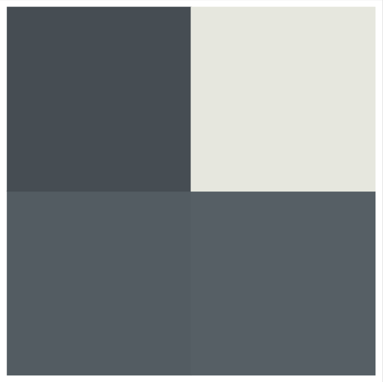


,

x = {181, 5, 157, 152}

$\hat{y}_{\text{left}}$=0.977467, $\hat{y}_{\text{right}}$=0.00457757

$\hat{y}$=2 ⟶ Stairs

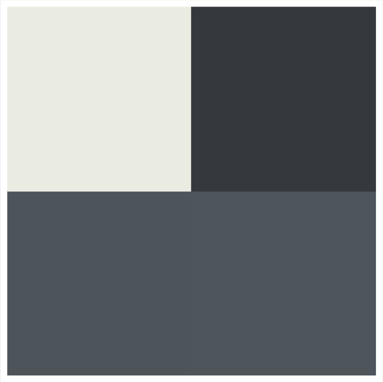


,

x = {0, 212, 169, 167}

$\hat{y}_{\text{left}}$=0.00133432, $\hat{y}_{\text{right}}$=0.987943

$\hat{y}$=2 ⟶ Stairs

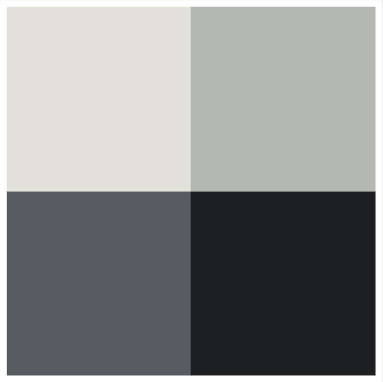


,

x = {12, 61, 158, 248}

$\hat{y}_{\text{left}}$=0.857661, $\hat{y}_{\text{right}}$=0.987182

$\hat{y}$=5 ⟶ Stairs



}

Example 1: Identify the stairs pattern

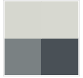
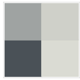
Again, we can try to find a good model fit:

```
In[∘]:= cost[w11_, w21_, w31_, w41_, w12_, w22_, w32_, w42_, bleft_, bright_, wleft_, wright_, b_] := Module[{},
               1
        ────────────────── Total@
        2 Length[train]
          Table[(multiperc2[train[[i, 2 ;; 5]], w11, w21, w31, w41, w12, w22, w32, w42, bleft, bright, wleft, wright, b] - train[[i, -1]])^2,
           {i, 1, Length[train]}]]
```

```
In[∘]:= fit3 = NMinimize[
            cost[w11, w21, w31, w41, w12, w22, w32, w42, bleft, bright, wleft, wright, b1]
            ,
            {{w11, -.01, .01},
             {w21, -.01, .01},
             {w31, -.01, .01},
             {w41, -.01, .01}, {bleft, -.01, .01},
             {w12, -.01, .01},
             {w22, -.01, .01},
             {w32, -.01, .01},
             {w42, -.01, .01}, {bright, -.01, .01},
             {wleft, -10, 10}, {wright, -10, 10}, {b1, -10, 10}},
            Method → "SimulatedAnnealing"
          ];
```

```
In[∘]:= fittedPars = Flatten[{{w11, w21, w31, w41}, {w12, w22, w32, w42}, bleft, bright, {wleft, wright}, b1}] /. fit3[[2]]

Out[∘]= {-0.00420598, -0.0148803, 0.00648986, -0.000534615, 0.0163142,
         0.000646621, -0.00218628, -0.00914712, 0.00573565, 0.00260502, 5.77451, -4.97981, 2.08227}
```

Out[∘]=

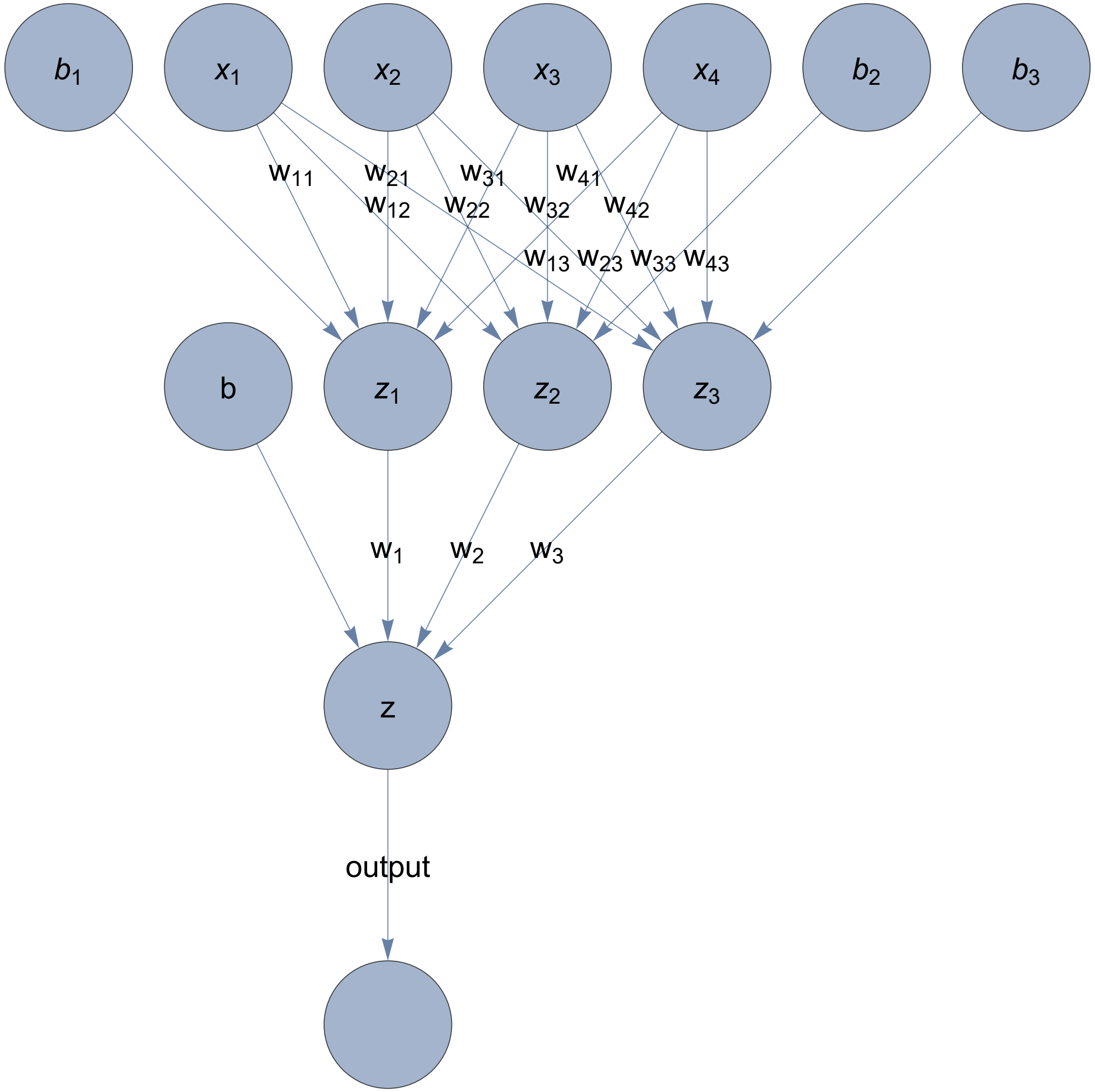
our classific.	case	initial classific.
1		1
1		1
1		1
0		0
1		1
1		1
1		1
1		1
1		1
1		1

our classific.	case	initial classific.
1		1
1		1
0		0
1		1
1		1
1		1
0		0
1		1
1		1
1		1

Alternative

- Build a model that fires when the bottom row is dark $\hat{y}_1$
- Build a model that fires when the top left pixel is dark and the top right pixel is light, $\hat{y}_2$
- Build a model that fires when the top left pixel is light and the top right pixel is dark, $\hat{y}_3$
- Add the base models so that the final sigmoid function only fires if $\hat{y}_1$ and $\hat{y}_2$ are large, or $\hat{y}_1$ and $\hat{y}_3$ are large. (Note that $\hat{y}_2$ and $\hat{y}_3$ cannot both be large)

Out[∘]=


b_1
x_1
x_2
x_3
x_4
b_2
b_3
w_{11}
w_{21}
w_{31}
w_{41}
w_{12}
w_{22}
w_{32}
w_{42}
w_{13}
w_{23}
w_{33}
w_{43}
b
z_1
z_2
z_3
w_1
w_2
w_3
z
output

Alternative

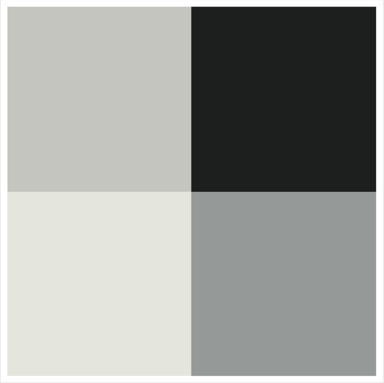
```
In[∘]:= Clear[multiperc3]
        multiperc3[x_List, w11_, w21_, w31_, w41_, w12_, w22_,
          w32_, w42_, w13_, w23_, w33_, w43_, b1_, b2_, b3_, w1_, w2_, w3_, b_] := Module[
          {y1 = {w11, w21, w31, w41}.x + b1,
           y2 = {w12, w22, w32, w42}.x + b2,
           y3 = {w13, w23, w33, w43}.x + b3, y},
          y = w1 * Piecewise[{{1, LogisticSigmoid[y1] > 0.5}, {0, True}}] + w2 * Piecewise[{{1, LogisticSigmoid[y2] > 0.5}, {0, True}}] +
            w3 * Piecewise[{{1, LogisticSigmoid[y3] > 0.5}, {0, True}}] + b;
          Piecewise[{{1, LogisticSigmoid[y] > 0.5}, {0, True}}]]
        multiresult3[x_List, w11_, w21_, w31_, w41_, w12_, w22_, w32_, w42_, w13_, w23_, w33_, w43_, b1_,
           b2_, b3_, w1_, w2_, w3_, b_?NumberQ] /; Length[x] == Length[w] := Module[{res, im, lab, y1, y2, y3, y, z},
          y1 = {w11, w21, w31, w41}.x + b1;
          y2 = {w12, w22, w32, w42}.x + b2;
          y3 = {w13, w23, w33, w43}.x + b3;
          z = w1 * Piecewise[{{1, LogisticSigmoid[y1] > 0.5}, {0, True}}] + w2 * Piecewise[{{1, LogisticSigmoid[y2] > 0.5}, {0, True}}] +
            w3 * Piecewise[{{1, LogisticSigmoid[y3] > 0.5}, {0, True}}] + b;
          y = LogisticSigmoid[z] // N;
          lab = Column[{
              Row[{"x=", x}],
              Row[{"ŷ₁=", Chop@LogisticSigmoid[y1], ", ŷ₂=", Chop@LogisticSigmoid[y2], ", ŷ₃=", Chop@LogisticSigmoid[y3]}],
              Row[{"ŷ=", Chop@y, " ⟶ ", Piecewise[{{1, y > 0.5}, {0, True}}] /. {0 → "Not stairs", 1 → "Stairs"}}]}];
          Labeled[Show[fig[x], ImageSize → 200], lab, Top]]

In[∘]:= sol3 = {0, 0, .2, .2, .1, -.1, 0, 0, -.1, .01, 0, 0, -.5, -.05, -.05, 3, 3, 3, -4};
```

Alternative

In[•]:=
```
multiresult3[#, Sequence @@ sol3] & /@ xlist
```

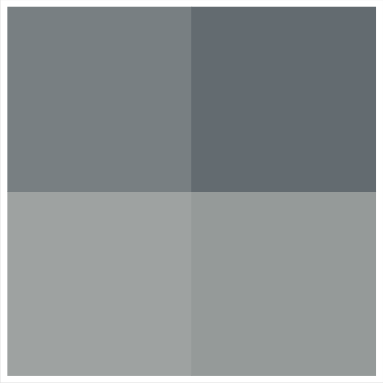
Out[•]= {

x={47, 250, 8, 88}
$\hat{y}_1$=1., $\hat{y}_2=1.45247\times10^{-9}$, $\hat{y}_3$=0.0953495
$\hat{y}$=0.268941 ⟶ Not stairs



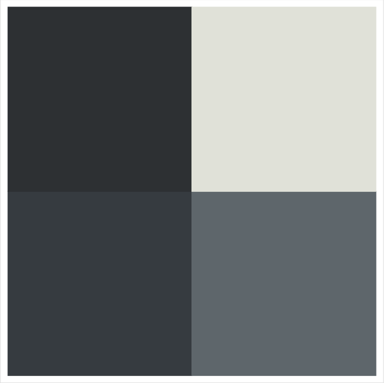
,

x={115, 138, 80, 88}
$\hat{y}_1$=1., $\hat{y}_2$=0.0870658, $\hat{y}_3$=0.0000383009
$\hat{y}$=0.268941 ⟶ Not stairs



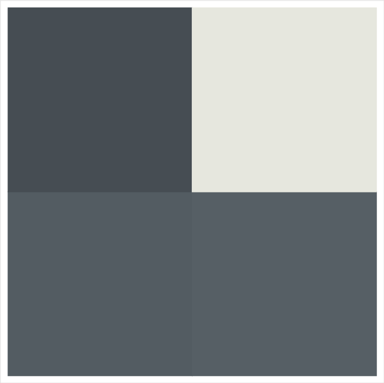
,

x={225, 13, 210, 144}
$\hat{y}_1$=1., $\hat{y}_2$=1., $\hat{y}_3=1.83281\times10^{-10}$
$\hat{y}$=0.880797 ⟶ Stairs



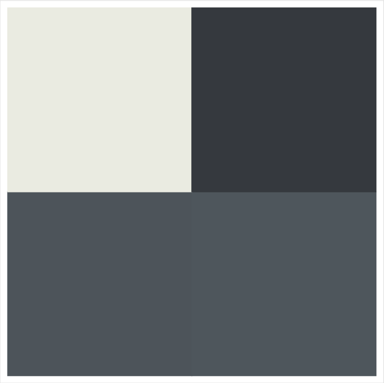
,

x={181, 5, 157, 152}
$\hat{y}_1$=1., $\hat{y}_2$=1., $\hat{y}_3=1.37807\times10^{-8}$
$\hat{y}$=0.880797 ⟶ Stairs



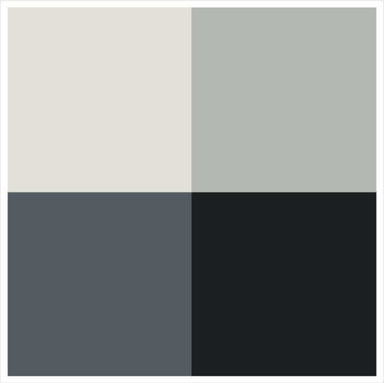
,

x={0, 212, 169, 167}
$\hat{y}_1$=1., $\hat{y}_2=5.9053\times10^{-10}$, $\hat{y}_3$=0.887953
$\hat{y}$=0.880797 ⟶ Stairs



,

x={12, 61, 158, 248}
$\hat{y}_1$=1., $\hat{y}_2$=0.00703359, $\hat{y}_3$=0.345247
$\hat{y}$=0.268941 ⟶ Not stairs



}

Alternative

```
In[∘]:= cost3[w11_, w21_, w31_, w41_, w12_, w22_, w32_, w42_, w13_, w23_, w33_, w43_, b1_, b2_, b3_, w1_, w2_, w3_, b_] := Module[{},
                1
          ─────────────────
          2 Length[train]
          Total@Table[(multiperc3[train[[i, 2 ;; 5]], w11, w21, w31, w41, w12, w22, w32, w42, w13, w23, w33, w43, b1, b2, b3, w1, w2, w3, b] -
                train[[i, -1]])^2, {i, 1, Length[train]}]]
```

```
In[∘]:= Clear[b, b2];
        fit = NMinimize[
          cost3[w11, w21, w31, w41, w12, w22, w32, w42, w13, w23, w33, w43, b1, b2, b3, w1, w2, w3, b]
          ,
          {{w11, -.01, .01},
           {w21, -.01, .01},
           {w31, -.01, .01},
           {w41, -.01, .01}, {b1, -.01, .01},
           {w12, -.01, .01},
           {w22, -.01, .01},
           {w32, -.01, .01},
           {w42, -.01, .01}, {b2, -.01, .01},
           {w13, -.01, .01},
           {w23, -.01, .01},
           {w33, -.01, .01},
           {w43, -.01, .01}, {b3, -.01, .01},
           {w1, -10, 10}, {w2, -10, 10}, {w3, -10, 10}, {b, -10, 10}},
          Method → {"SimulatedAnnealing", "PostProcess" → False}
         ]
```

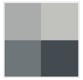
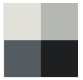
```
Out[∘]= {0.09375, {w11 → -0.00845095, w21 → -0.0073568, w31 → 0.00113542, w41 → 0.00870272, b1 → -0.00695265, w12 → 0.00548636,
          w22 → 0.00334982, w32 → -0.00506733, w42 → -0.00166905, b2 → -0.00736508, w13 → -0.000389208, w23 → 0.000853505,
          w33 → -0.00114991, w43 → 0.0110203, b3 → 0.00233228, w1 → -0.118846, w2 → -10.7137, w3 → 10.7233, b → -3.5155}}
```

```
In[]:= (*Generate some variables*)
vars3 = Block[{x}, Unique[ConstantArray[x, 19], Temporary]];
fit = NMinimize[
   cost3[Sequence @@ vars3]
   ,
   vars3,
   Method → {"SimulatedAnnealing", "PostProcess" → False}
  ]

Out[]= {0.0875, {x$11049314 → 0.399941, x$11049315 → 0.32055, x$11049316 → -0.764994, x$11049317 → -1.60258,
   x$11049318 → 0.785255, x$11049319 → 1.11436, x$11049320 → -0.102837, x$11049321 → -1.16122, x$11049322 → 0.90093,
   x$11049323 → -0.303492, x$11049324 → 0.899475, x$11049325 → -0.00579884, x$11049326 → -0.549336, x$11049327 → 1.47554,
   x$11049328 → -1.63101, x$11049329 → 0.719496, x$11049330 → -1.37765, x$11049331 → -0.355232, x$11049332 → 0.940447}}
```

```
In[]:= fittedPars3 = vars3 /. fit[[2]]

Out[]= {0.399941, 0.32055, -0.764994, -1.60258, 0.785255, 1.11436, -0.102837, -1.16122, 0.90093,
  -0.303492, 0.899475, -0.00579884, -0.549336, 1.47554, -1.63101, 0.719496, -1.37765, -0.355232, 0.940447}
```

Alternative

Out[ ]=

our classific.	case	initial classific.
1		1
1		1
1		0
1		1
1		1
0		0
0		0
1		1
1		1
1		1

our classific.	case	initial classific.
0		1
1		1
1		1
1		1
0		1
1		1
1		1
0		0
1		1
1		0

Example 2: Identify lightly shaded stairs

- Build models that fire for “shaded bottom row”, “shaded x1 and white x2”, “shaded x2 and white x1”, $\hat{y}_1$, $\hat{y}_2$, and $\hat{y}_3$
- Build models that fire for “dark bottom row”, “dark x1 and white x2”, “dark x2 and white x1”$\hat{y}_4$, $\hat{y}_5$, and $\hat{y}_6$
- Combine the models so that the “dark” identifiers are essentially subtracted from the “shaded” identifiers before squashing the result with a sigmoid function

Variable number of hidden layer neurons

```
In[∘]:= neuron[y_] := Piecewise[{{1, LogisticSigmoid[y] > 0.5}, {0, True}}]
        Clear[multipercN]
        multipercN[x_, wlist_, b_, dim_] := Module[
          {wpart, yhidden, wout},
          wpart = Partition[Drop[wlist, -dim], Length[x]];
          wout = Take[wlist, -dim];
          yhidden = neuron /@ Table[wpart[[i]].x + b[[i]], {i, 1, dim}];
          neuron[yhidden.wout + b[[-1]]]
         ]
        testTable[func_, fittedPars_, func2_, pars2_, dim_] := With[{},
          Row[{TableForm[{func[Most@#, Sequence @@ fittedPars, dim], fig[Most@#], func2[Most@#, Sequence @@ pars2]} & /@ RandomSample[
                train[[All, 2 ;; 6]], 10], TableHeadings → {None, {"our \nclassific.", "case", "initial \nclassific."}}], Spacer[50],
            TableForm[{func[Most@#, Sequence @@ fittedPars, dim], fig[Most@#], func2[Most@#, Sequence @@ pars2]} & /@
              RandomSample[train[[All, 2 ;; 6]], 10], TableHeadings → {None, {"our \nclassific.", "case", "initial \nclassific."}}]}]]
        multiresultN[x_, wlist_, b_, dim_] /; Length[x] == Length[w] := Module[{res, im, lab, y1, y2, y3, y, z, wpart, yhidden, wout},
          wpart = Partition[Drop[wlist, -dim], Length[x]];
          wout = Take[wlist, -dim];
          yhidden = neuron /@ Table[wpart[[i]].x + b[[i]], {i, 1, dim}];
          y = neuron[yhidden.wout + b[[-1]]];
          lab = Column[{
              Row[{"x=", x}],
              Row[{"ŷ=", Chop@y, " ⟶ ", Piecewise[{{1, y > 0.5}, {0, True}}] /. {0 → "Not stairs", 1 → "Stairs"}}]}];
          Labeled[Show[fig[x], ImageSize → 200], lab, Top]]
```

as a reference

```
In[∘]:= vars3 = Block[{x}, Unique[ConstantArray[x, 19], Temporary]];
        fit = NMinimize[
          cost3[Sequence @@ vars3]
          ,
          vars3,
          Method → {"SimulatedAnnealing", "PostProcess" → False}
         ]

Out[∘]= {0.0875, {x$11050595 → 0.399941, x$11050596 → 0.32055, x$11050597 → -0.764994, x$11050598 → -1.60258,
          x$11050599 → 0.785255, x$11050600 → 1.11436, x$11050601 → -0.102837, x$11050602 → -1.16122, x$11050603 → 0.90093,
          x$11050604 → -0.303492, x$11050605 → 0.899475, x$11050606 → -0.00579884, x$11050607 → -0.549336, x$11050608 → 1.47554,
          x$11050609 → -1.63101, x$11050610 → 0.719496, x$11050611 → -1.37765, x$11050612 → -0.355232, x$11050613 → 0.940447}}
```

```
In[∘]:= costN[wvars_List, bvars_List, dim_] := Module[{},
              1
          ─────────────── Total@Table[(multipercN[train[[i, 2 ;; 5]], wvars, bvars, dim] - train[[i, -1]])^2, {i, 1, Length[train]}]]
          2 Length[train]
```

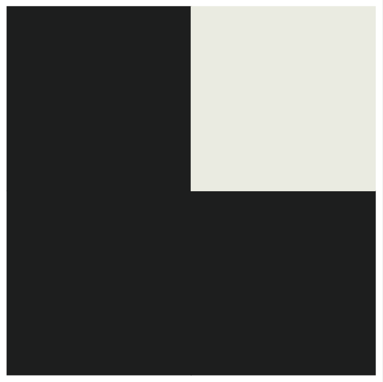
6 hidden neurons

```
In[◦]:= dim = 6;
        len = Length[train[[1]]];
        wvarsN = Block[{x}, Unique[ConstantArray[x, len * dim + dim], Temporary]];
        bvarsN = Block[{x}, Unique[ConstantArray[x, dim + 1], Temporary]];
        fit = NMinimize[
           costN[wvarsN, bvarsN, dim]
           ,
           Flatten[{wvarsN, bvarsN}],
           Method → {"SimulatedAnnealing", "PostProcess" → False}
          ];
        fittedParsN = {wvarsN /. fit[[2]], bvarsN /. fit[[2]]}

Out[◦]= {{2.02583, 0.206402, 0.237031, -0.701094, -0.635576, -2.68429, 1.17745, -0.276328, 2.37513, -2.13776, 0.635347,
          -0.0707879, 0.0888537, 0.976746, -0.0956164, 1.35121, -0.763909, -2.74726, -0.240621, 0.742415, 0.966362,
          1.11892, 0.702946, -2.02144, 0.912925, -0.386727, 0.314224, -0.147556, 0.944787, 0.613136, 0.043005, 2.46665,
          0.369599, -2.23704, 0.751625, -1.22305, -0.945495, -0.256221, 0.251903, -0.301909, -1.10736, -0.657544},
         {-1.98997, -0.119695, 1.349, -1.18864, 0.12037, 0.373991, 1.2653}}
```

6 hidden neurons

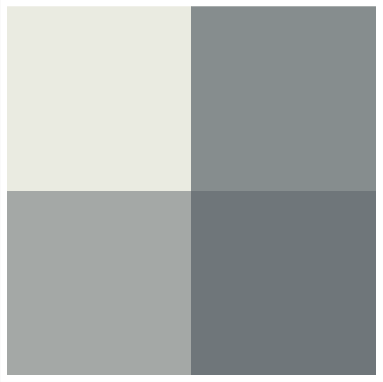
Out[∘]= {

x={250, 0, 250, 250}
ŷ=1 ⟶ Stairs



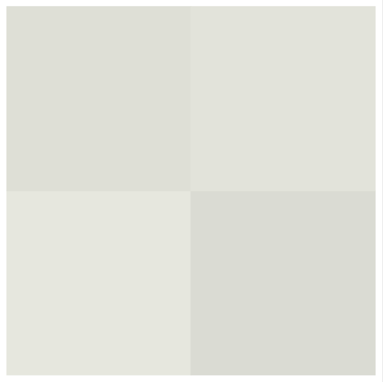
,

x={0, 100, 75, 125}
ŷ=1 ⟶ Stairs



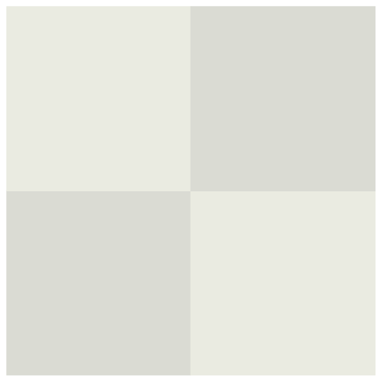
,

x={15, 10, 5, 20}
ŷ=1 ⟶ Stairs



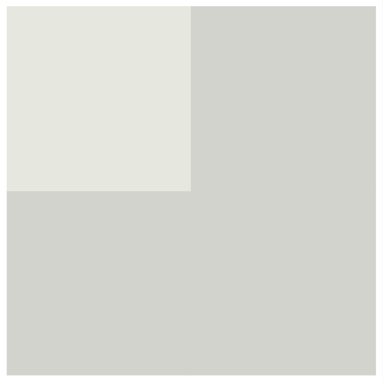
,

x={0, 20, 20, 0}
ŷ=0 ⟶ Not stairs



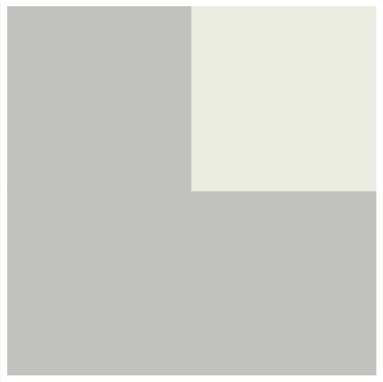
,

x={5, 30, 30, 30}
ŷ=1 ⟶ Stairs



,

x={50, 0, 50, 50}
ŷ=1 ⟶ Stairs

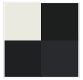
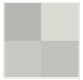


}

6 hidden neurons

```
In[•]:= testTable[multipercN, fittedParsN, multiperc3, sol3, dim]
```

Out[•]=

our classific.	case	initial classific.	our classific.	case	initial classific.
0		0	1		1
1		1	0		1
1		1	1		0
1		1	0		1
1		1	0		1
1		1	0		0
1		1	0		0
1		1	0		1
0		1	1		1
1		1	0		0

10 hidden neurons

```
In[•]:= dim = 10;
        len = Length[train[[1]]];
        wvarsN = Block[{x}, Unique[ConstantArray[x, len * dim + dim], Temporary]];
        bvarsN = Block[{x}, Unique[ConstantArray[x, dim + 1], Temporary]];
        fit = NMinimize[
            costN[wvarsN, bvarsN, dim]
            ,
            Flatten[{wvarsN, bvarsN}],
            Method → {"SimulatedAnnealing", "PostProcess" → False}
          ];
        fittedParsN = {wvarsN /. fit[[2]], bvarsN /. fit[[2]]}
```

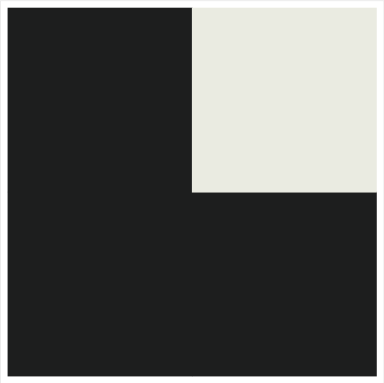
```
Out[•]= {{0.205449, -0.260999, 0.256316, 0.898359, -0.196817, 0.0919456, 0.320884, -0.341711, 0.550715, 1.53098, -0.658185, -0.504411,
          -0.287051, 1.04755, -0.127735, -0.468566, -0.304404, 0.0575273, -0.78058, 0.347659, 0.70052, 0.104203, 0.531865, -0.476583,
          -0.839882, -0.759456, 0.879841, 0.584988, 0.0737989, -0.513292, 0.37122, -0.0126992, 0.367914, -0.0553497, -0.275402,
          0.1456, 1.07766, -0.310888, -0.105349, -0.238082, -0.443265, -0.139522, -0.194712, -1.15143, -0.737395, -0.23176,
          1.13461, 0.442998, -0.305222, -1.17847, -0.433567, 0.00860119, 0.190128, 1.19222, 1.12701, 0.565783, 0.387154, -0.891047,
          -1.22309, -0.86649, -1.32942, 0.940917, -0.419884, -0.62525, 0.411171, 0.146271, 1.15826, 0.0581324, -0.264517, -0.6493},
         {-1.02141, -0.477148, -0.00201365, 1.12278, 0.273289, -0.297404, -0.146621, 0.159325, -0.958299, -0.808803, 1.21979}}
```

10 hidden neurons

```
In[∘]:= multiresultN[#, Sequence @@ fittedParsN, 10] & /@ xlistLight
```

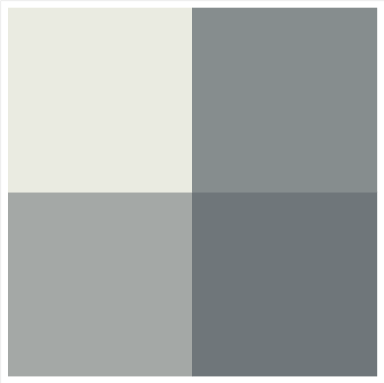
Out[∘]= {

x={250, 0, 250, 250}
ŷ=1 ⟶ Stairs



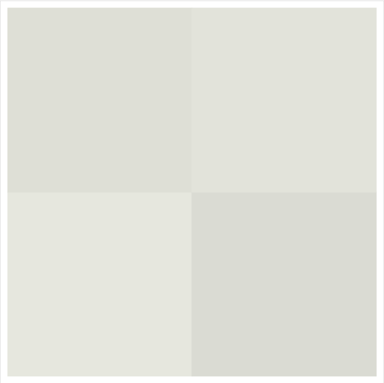
,

x={0, 100, 75, 125}
ŷ=1 ⟶ Stairs



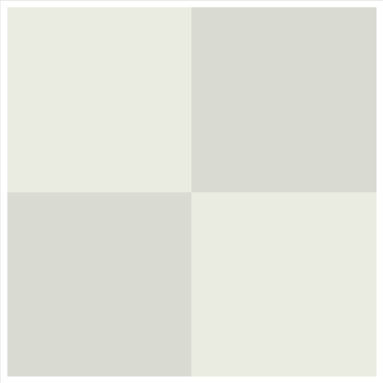
,

x={15, 10, 5, 20}
ŷ=0 ⟶ Not stairs



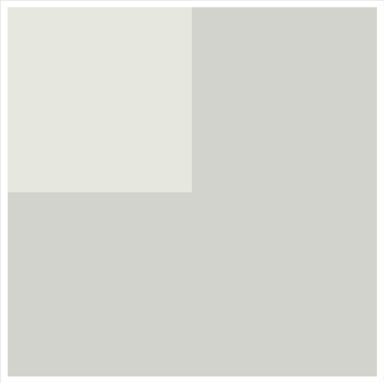
,

x={0, 20, 20, 0}
ŷ=1 ⟶ Stairs



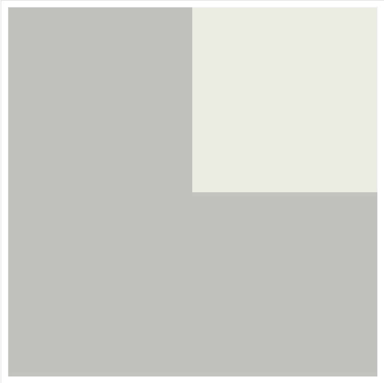
,

x={5, 30, 30, 30}
ŷ=1 ⟶ Stairs



,

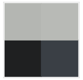
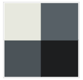
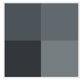
x={50, 0, 50, 50}
ŷ=1 ⟶ Stairs



}

10 hidden neurons

```
testTable[multipercN, fittedParsN, multiperc3, sol3, 10]
```

our classific.	case	initial classific.	our classific.	case	initial classific.
1		1	1		1
1		1	0		0
0		0	1		1
0		1	0		0
1		1	1		1
1		1	1		1
1		1	1		1
1		0	1		1
1		1	0		1
1		1	1		1

20 hidden neurons

```
In[•]:= dim = 20;
        len = Length[train[[1]]];
        wvarsN = Block[{x}, Unique[ConstantArray[x, len * dim + dim], Temporary]];
        bvarsN = Block[{x}, Unique[ConstantArray[x, dim + 1], Temporary]];
        fit = NMinimize[
           costN[wvarsN, bvarsN, dim]
           ,
           Flatten[{wvarsN, bvarsN}],
           Method → {"SimulatedAnnealing", "PostProcess" → False}
          ];
        fittedParsN = {wvarsN /. fit[[2]], bvarsN /. fit[[2]]}
```

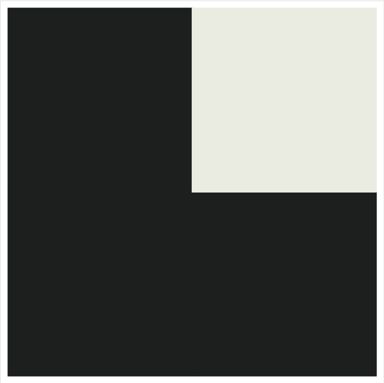
```
Out[•]= {{-0.946446, -0.162026, -1.4861, -0.573792, 2.34132, -0.977974, -1.15274, -0.430333, 0.109348, -1.44433, -2.06802, 0.249012,
          0.78354, -0.476024, 0.458633, -1.34471, 0.448698, -0.282597, -0.885189, -1.39472, 1.39812, -0.309155, -0.916219, 0.670742,
          0.110664, -2.01972, -0.590292, -0.284498, 0.00940101, 1.77342, -0.255983, -0.370408, -2.09346, 1.82699, 0.541999, 1.73674,
          -1.71693, -0.531323, 0.884154, -0.408858, -0.00863027, 2.01352, -1.38017, -1.27792, 0.971402, 1.09152, -0.168301,
          -0.347065, -0.485515, 0.682908, 0.395228, 0.486712, -0.0400939, -0.76169, 2.14546, 0.267101, -0.657876, -0.816633,
          0.57269, 0.288957, -1.48711, 1.84582, -1.20776, 0.272658, 1.08234, -0.111427, -0.952565, -1.68435, -0.247622, -0.516669,
          -0.353757, 0.227344, 0.558118, 1.03976, -0.172813, 0.798622, -2.56948, 0.910374, -0.527361, -0.0522759, 0.150644, 1.68987,
          -0.231813, -0.669382, -1.01562, 1.66439, -1.21234, 0.154666, 0.225746, -0.718148, 0.351991, -1.91078, -1.89288, 0.203486,
          0.0055919, 0.270338, -0.0560665, -0.775819, 0.211941, -1.19144, 0.535294, -0.500507, -2.00757, 0.399007, -0.14672,
          -0.154361, 1.31028, -0.225357, -1.16426, -0.106391, -1.13829, -2.21371, 0.388534, -0.167731, 1.1358, -1.42148, 0.529728,
          0.984752, -0.296704, 1.54999, 1.84067, -0.754825, -0.321556, 0.229746, -0.349183, -0.293138, -0.991506, -0.749218,
          -0.482715, 1.29726, -1.52517, -0.0127165, 0.161566, 1.21232, 0.833027, 1.0994, 0.253497, -2.35628, -0.0428724, 1.26605},
         {-0.0341885, 0.365126, 0.438084, -0.00163203, 0.08849, 1.07552, -0.0685528, 1.31061, 0.802687, -0.520468,
          1.73, -0.021149, 0.885356, -0.546477, 0.821996, 0.19557, 1.76707, 0.303449, -0.317198, -0.39908, -0.258702}}
```

20 hidden neurons

```
In[•]:= multiresultN[#, Sequence @@ fittedParsN, 20] & /@ xlistLight
```

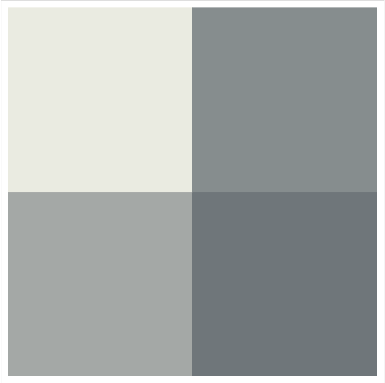
Out[•]= {

x={250, 0, 250, 250}
ŷ=1 ⟶ Stairs



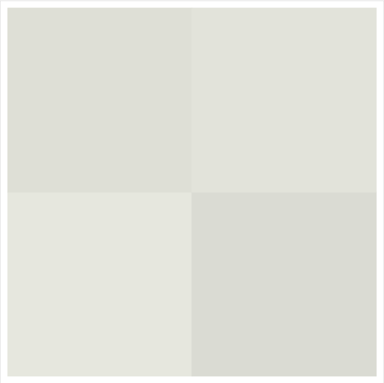
,

x={0, 100, 75, 125}
ŷ=1 ⟶ Stairs



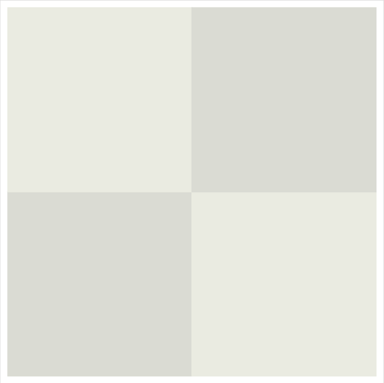
,

x={15, 10, 5, 20}
ŷ=0 ⟶ Not stairs



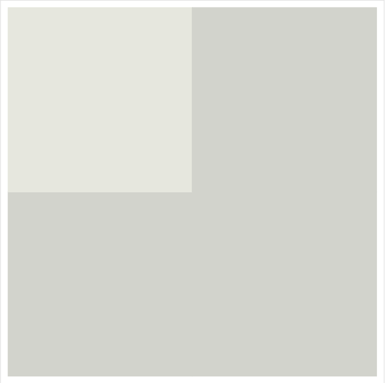
,

x={0, 20, 20, 0}
ŷ=1 ⟶ Stairs



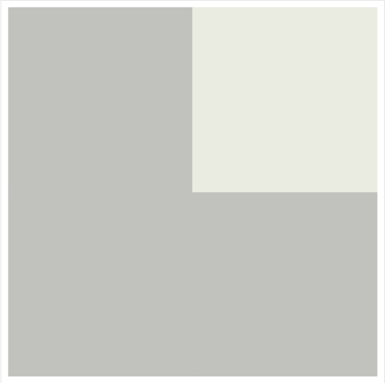
,

x={5, 30, 30, 30}
ŷ=1 ⟶ Stairs



,

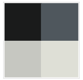
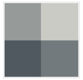
x={50, 0, 50, 50}
ŷ=1 ⟶ Stairs



}

20 hidden neurons

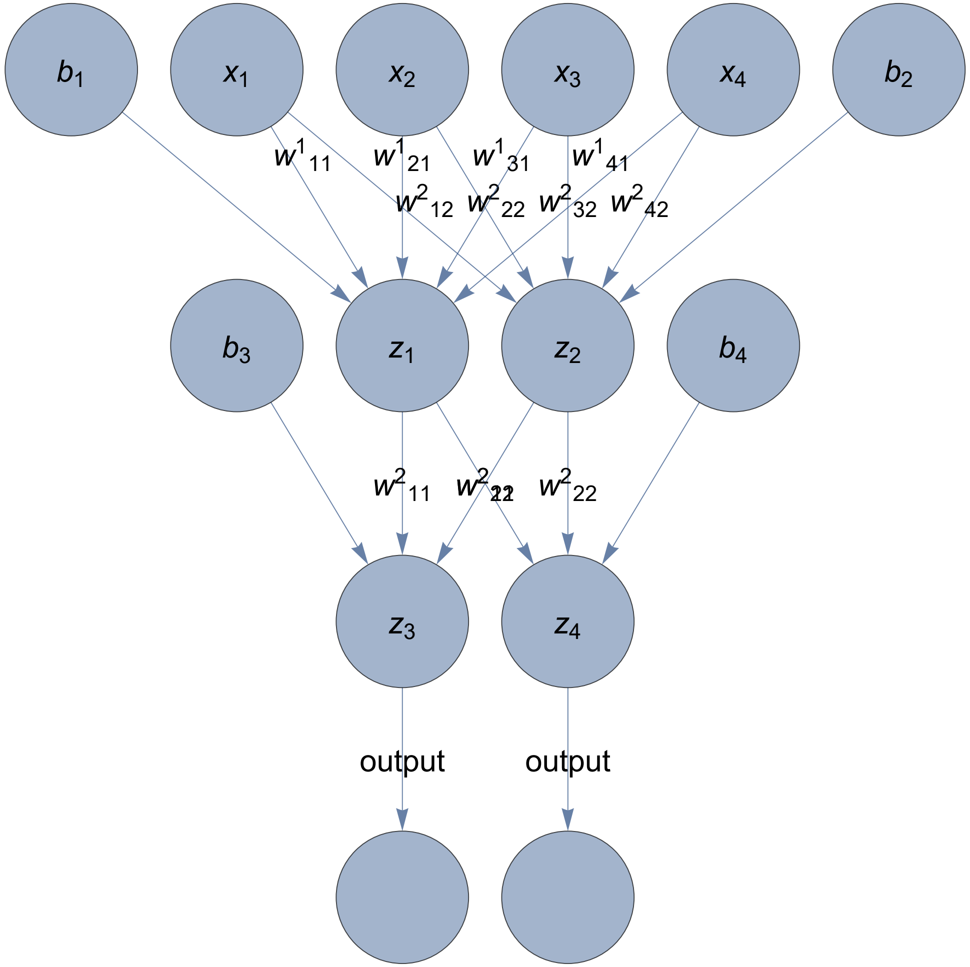
```
testTable[multipercN, fittedParsN, multiperc3, sol3, 20]
```

our classific.	case	initial classific.	our classific.	case	initial classific.
1		0	0		0
0		1	1		1
1		1	1		1
1		1	1		1
1		1	0		1
1		1	0		0
1		1	1		1
1		1	0		1
1		1	1		0
1		1	1		1

Numerical example

We choose a network with one hidden layer and two output channels.

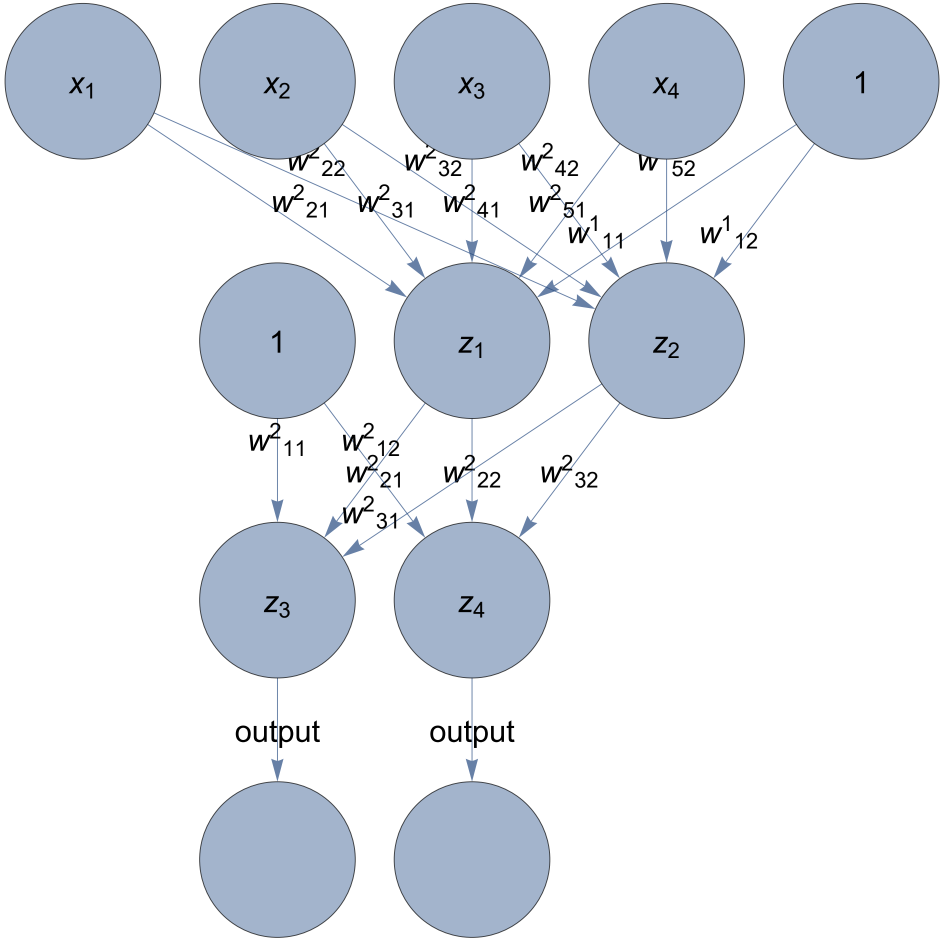
Out[ ]=



Numerical example

This can be simplified be rearranging the network a little. The bias terms is treated as addition input node that we fix to 1 and assign weights to.

Out[∘]=



Numerical example

Finally, the z_1 and z_2 are computed using a sigmoid function, while the z_3 and z_4 are computed using a softmax function.

Note here that we’re using the subscript i to refer to the i-th training sample as it gets processed by the network. We use superscripts to denote the layer of the network. And for each weight matrix, the term $(w^l)_{ab}$ represents the weight from the a-th node in the l-th layer to the b-th node in the (l+1)-th layer. Since keeping track of notation is tricky and critical, we will supplement our algebra with this sample of training data

Out[ ]//TableForm=

```
Image ID   p1    p2    p3    p4    IsSTairs
-------------------------------------------
1          252   4     155   175   1
2          175   10    186   200   1
3          82    131   230   100   0
4          115   138   80    88    0
```

Numerical example

The entire network can be formulated in matrix form:

Out[ ]//TraditionalForm=

$$X^1 = \begin{pmatrix} x^1_{11} & x^1_{12} & x^1_{13} & x^1_{14} & x^1_{15} \\ x^1_{21} & x^1_{22} & x^1_{23} & x^1_{24} & x^1_{25} \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ x^1_{N1} & x^1_{N2} & x^1_{N3} & x^1_{N4} & x^1_{N5} \end{pmatrix} = \begin{pmatrix} 1 & p_{11} & p_{12} & p_{13} & p_{14} \\ 1 & p_{21} & p_{22} & p_{23} & p_{24} \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ 1 & p_{N1} & p_{N2} & p_{N3} & p_{N4} \end{pmatrix} = \begin{pmatrix} 1 & 252 & 4 & 155 & 175 \\ 1 & 175 & 10 & 186 & 200 \\ 1 & 82 & 131 & 230 & 100 \\ 1 & 115 & 138 & 80 & 88 \end{pmatrix}$$

Out[ ]=

$$W^1 = \begin{pmatrix} w^1_{11} & w^1_{12} \\ w^1_{21} & w^1_{22} \\ w^1_{31} & w^1_{32} \\ w^1_{41} & w^1_{42} \\ w^1_{51} & w^1_{52} \end{pmatrix} \qquad Z^1 = \begin{pmatrix} z^1_{11} & z^1_{12} \\ z^1_{21} & z^1_{22} \\ \ldots & \ldots \\ z^1_{N1} & z^1_{N2} \end{pmatrix}$$

Numerical example

Out[•]= $$X^2 = \begin{pmatrix} x^2_{11} & x^2_{12} & x^2_{13} \\ x^2_{21} & x^2_{22} & x^2_{23} \\ \cdots & \cdots & \cdots \\ x^2_{N1} & x^2_{N2} & x^2_{N3} \end{pmatrix} = \begin{pmatrix} 1 & x^2_{12} & x^2_{13} \\ 1 & x^2_{22} & x^2_{23} \\ \cdots & \cdots & \cdots \\ 1 & x^2_{N2} & x^2_{N3} \end{pmatrix} \qquad W^2 = \begin{pmatrix} w^2_{11} & w^2_{12} \\ w^2_{21} & w^2_{22} \\ w^2_{31} & w^2_{32} \end{pmatrix}$$

Out[•]= $$Z^2 = \begin{pmatrix} z^2_{11} & z^2_{12} \\ z^2_{21} & z^2_{22} \\ \cdots & \cdots \\ z^2_{N1} & z^2_{N2} \end{pmatrix} \qquad Y = \begin{pmatrix} y_{11} & y_{12} \\ y_{21} & y_{22} \\ \cdots & \cdots \\ y_{N1} & y_{N2} \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 1 \end{pmatrix} \qquad \hat{Y} = \begin{pmatrix} \hat{y}_{11} & \hat{y}_{12} \\ \hat{y}_{21} & \hat{y}_{22} \\ \cdots & \cdots \\ \hat{y}_{N1} & \hat{y}_{N2} \end{pmatrix}$$

Initialize the weights

We’ll pick uniform random values between -0.01 and 0.01.

Out[ ]= $W^1 = \begin{pmatrix} w^1_{11} & w^1_{12} \\ w^1_{21} & w^1_{22} \\ w^1_{31} & w^1_{32} \\ w^1_{41} & w^1_{42} \\ w^1_{51} & w^1_{52} \end{pmatrix} = \begin{pmatrix} -0.00469 & 0.00797 \\ -0.00256 & 0.00889 \\ 0.00146 & 0.00322 \\ 0.00816 & 0.00258 \\ -0.00597 & -0.00876 \end{pmatrix}$ $W^2 = \begin{pmatrix} w^2_{11} & w^2_{12} \\ w^2_{21} & w^2_{22} \\ w^2_{31} & w^2_{32} \end{pmatrix} = \begin{pmatrix} -0.00588 & -0.00232 \\ -0.00647 & 0.0054 \\ 0.00374 & -0.00005 \end{pmatrix}$

Forward pass

Now let’s walk through the forward pass to generate predictions for each of our training samples.

Step 1

Compute the signal going into the hidden layer, Z^1.

Out[∘]//TraditionalForm=

$$Z^1 = X^1.W^1$$

Out[∘]//TraditionalForm=

$$\begin{pmatrix} z^1_{11} & z^1_{12} \\ z^1_{21} & z^1_{22} \\ \cdots & \cdots \\ z^1_{N1} & z^1_{N2} \end{pmatrix} = \begin{pmatrix} x^1_{11} & x^1_{12} & x^1_{13} & x^1_{14} & x^1_{15} \\ x^1_{21} & x^1_{22} & x^1_{23} & x^1_{24} & x^1_{25} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ x^1_{N1} & x^1_{N2} & x^1_{N3} & x^1_{N4} & x^1_{N5} \end{pmatrix} . \begin{pmatrix} w^1_{11} & w^1_{12} \\ w^1_{21} & w^1_{22} \\ w^1_{31} & w^1_{32} \\ w^1_{41} & w^1_{42} \\ w^1_{51} & w^1_{52} \end{pmatrix}$$

Step 1

Squash the signal to the hidden layer with the sigmoid function to determine the inputs to the output layer, X^2.

Out[◦]//TraditionalForm=

$$X^2 = \left(1 \;\; \text{sigmoid}(Z^1)\right)$$

Out[◦]//TraditionalForm=

$$\begin{pmatrix} x^2_{11} & x^2_{12} & x^2_{13} \\ x^2_{21} & x^2_{22} & x^2_{23} \\ \dots & \dots & \dots \\ x^2_{N1} & x^2_{N2} & x^2_{N3} \end{pmatrix} = \begin{pmatrix} 1 & \text{sigmoid}\left(z^2_{11}\right) & \text{sigmoid}\left(z^2_{12}\right) \\ 1 & \text{sigmoid}\left(z^2_{21}\right) & \text{sigmoid}\left(z^2_{22}\right) \\ \dots & \dots & \dots \\ 1 & \text{sigmoid}\left(z^2_{N1}\right) & \text{sigmoid}\left(z^2_{N2}\right) \end{pmatrix} = \begin{pmatrix} 1 & \frac{1}{1+\exp\left(-z^2_{11}\right)} & \frac{1}{1+\exp\left(-z^2_{12}\right)} \\ 1 & \frac{1}{1+\exp\left(-z^2_{21}\right)} & \frac{1}{1+\exp\left(-z^2_{22}\right)} \\ \dots & \dots & \dots \\ 1 & \frac{1}{1+\exp\left(-z^2_{N1}\right)} & \frac{1}{1+\exp\left(-z^2_{N2}\right)} \end{pmatrix}$$

Step 3

Calculate the signal going into the output layer, Z^3.

Out[•]//TraditionalForm=

$$Z^3 = X^2.W^2$$

Out[•]//TraditionalForm=

$$\begin{pmatrix} z^2_{11} & z^2_{12} \\ z^2_{21} & z^2_{22} \\ \cdots & \cdots \\ z^2_{N1} & z^2_{N2} \end{pmatrix} = \begin{pmatrix} x^2_{11} & x^2_{12} & x^2_{13} \\ x^2_{21} & x^2_{22} & x^2_{23} \\ \cdots & \cdots & \cdots \\ x^2_{N1} & x^2_{N2} & x^2_{N3} \end{pmatrix} . \begin{pmatrix} w^2_{11} & w^2_{12} \\ w^2_{21} & w^2_{22} \\ w^2_{31} & w^2_{32} \end{pmatrix} = \begin{pmatrix} x^2_{11}\, w^2_{11} + x^2_{12}\, w^2_{21} + x^2_{13}\, w^2_{31} & x^2_{11}\, w^2_{12} + x^2_{12}\, w^2_{22} + x^2_{13}\, w^2_{32} \\ x^2_{21}\, w^2_{11} + x^2_{22}\, w^2_{21} + x^2_{23}\, w^2_{31} & x^2_{21}\, w^2_{12} + x^2_{22}\, w^2_{22} + x^2_{23}\, w^2_{32} \\ \cdots & \cdots \\ x^2_{N1}\, w^2_{11} + x^2_{N2}\, w^2_{21} + x^2_{N3}\, w^2_{31} & x^2_{N1}\, w^2_{12} + x^2_{N2}\, w^2_{22} + x^2_{N3}\, w^2_{32} \end{pmatrix}$$

Step 4

Squash the signal to the output layer with the softmax function to determine the predictions, $\hat{Y}$. Recall that the softmax function is a mapping from $\mathbb{R}^n$ to $\mathbb{R}^n$. In other words, it takes a vector θ as input and returns an equal size vector as output. For the k-th element of the output,

Out[∘]//TraditionalForm=

$$\operatorname{softmax}(\theta)_k = \frac{\exp(\theta_k)}{\sum_{j=1}^{n} \exp(\theta_j)}$$

In our model, we apply the softmax function to each vector of predicted probabilities. In other words, we apply the softmax function “row-wise” to Z^3

Out[∘]//TraditionalForm=

$$\hat{Y} = \operatorname{softmax}_{\text{row-wise}}\left(Z^2\right)$$

Out[∘]//TraditionalForm=

$$\begin{pmatrix} \hat{y}_{11} & \hat{y}_{12} \\ \hat{y}_{21} & \hat{y}_{22} \\ \cdots & \cdots \\ \hat{y}_{\text{N1}} & \hat{y}_{\text{N2}} \end{pmatrix} = \begin{pmatrix} \operatorname{softmax}\left(\begin{pmatrix} z_{11}^2 & z_{12}^2 \end{pmatrix}\right)_1 & \operatorname{softmax}\left(\begin{pmatrix} z_{11}^2 & z_{12}^2 \end{pmatrix}\right)_2 \\ \operatorname{softmax}\left(\begin{pmatrix} z_{21}^2 & z_{22}^2 \end{pmatrix}\right)_1 & \operatorname{softmax}\left(\begin{pmatrix} z_{21}^2 & z_{22}^2 \end{pmatrix}\right)_2 \\ \cdots & \cdots \\ \operatorname{softmax}\left(\begin{pmatrix} z_{\text{N1}}^2 & z_{\text{N2}}^2 \end{pmatrix}\right)_1 & \operatorname{softmax}\left(\begin{pmatrix} z_{\text{N1}}^2 & z_{\text{N2}}^2 \end{pmatrix}\right)_2 \end{pmatrix} = \begin{pmatrix} \frac{\exp(z_{11}^2)}{\exp(z_{11}^2)+\exp(z_{12}^2)} & \frac{\exp(z_{12}^2)}{\exp(z_{11}^2)+\exp(z_{12}^2)} \\ \frac{\exp(z_{21}^2)}{\exp(z_{21}^2)+\exp(z_{22}^2)} & \frac{\exp(z_{22}^2)}{\exp(z_{21}^2)+\exp(z_{22}^2)} \\ \cdots & \cdots \\ \frac{\exp(z_{\text{N1}}^2)}{\exp(z_{\text{N1}}^2)+\exp(z_{\text{N2}}^2)} & \frac{\exp(z_{\text{N2}}^2)}{\exp(z_{\text{N1}}^2)+\exp(z_{\text{N2}}^2)} \end{pmatrix}$$

Step 4

Running the forward pass on our sample data gives

Out[•]//TraditionalForm=

$$Z^1 = \begin{pmatrix} -0.42392 & 1.12803 \\ -0.11433 & 0.3238 \\ 1.25645 & 0.87617 \\ 0.02983 & 0.9102 \end{pmatrix}$$

Out[•]//TraditionalForm=

$$X^2 = \begin{pmatrix} 1 & 0.395579 & 0.755475 \\ 1 & 0.471449 & 0.58025 \\ 1 & 0.778414 & 0.706028 \\ 1 & 0.507457 & 0.713041 \end{pmatrix}$$

Step 4

Out[•]//TraditionalForm=

$$Z^2 = \begin{pmatrix} 1 & 0.395579 & 0.755475 \\ 1 & 0.471449 & 0.58025 \\ 1 & 0.778414 & 0.706028 \\ 1 & 0.507457 & 0.713041 \end{pmatrix} . \begin{pmatrix} -0.00588 & -0.00232 \\ -0.00647 & 0.0054 \\ 0.00374 & -0.00005 \end{pmatrix} = \begin{pmatrix} -0.00561392 & -0.000221647 \\ -0.00676014 & 0.00019681 \\ -0.0082758 & 0.00184814 \\ -0.00649647 & 0.000384615 \end{pmatrix}$$

Out[•]//TraditionalForm=

$$\hat{Y} = \begin{pmatrix} \frac{\exp(-0.00561392)}{\exp(-0.00561392)+\exp(0.000221647)} & \frac{\exp(-0.000221647)}{\exp(-0.00561392)+\exp(0.000221647)} \\ \frac{\exp(-0.00676014)}{\exp(-0.00676014)+\exp(0.00019681)} & \frac{\exp(0.00019681)}{\exp(-0.00676014)+\exp(0.00019681)} \\ \frac{\exp(-0.0082758)}{\exp(-0.0082758)+\exp(0.00184814)} & \frac{\exp(0.00184814)}{\exp(-0.0082758)+\exp(0.00184814)} \\ \frac{\exp(-0.00649647)}{\exp(-0.00649647)+\exp(0.000384615)} & \frac{\exp(0.000384615)}{\exp(-0.00649647)+\exp(0.000384615)} \end{pmatrix} = \begin{pmatrix} 0.498541 & 0.501237 \\ 0.498261 & 0.501739 \\ 0.497469 & 0.502531 \\ 0.49828 & 0.50172 \end{pmatrix}$$

Backpropagation

We will now try to find the optimal weights by using the gradient descent method. We start by measuring the performance of our network using our loss function, cross entropy. The loss associated with the ith prediction would be

Out[∘]//TraditionalForm=

$$\boxed{\mathrm{CE}_i = \mathrm{CE}\left(\hat{Y}\; Y_i\right) = \sum_{c=1}^{C} y_{\mathrm{ic}} \log\left(\hat{y}_{\mathrm{ic}}\right)}$$

and c iterates over the target classes. Note here that CE is only affected by the prediction value associated with the True instance. For example, if we were doing a 3-class prediction problem and y = [0, 1, 0], then $\hat{y}$ = [0, 0.5, 0.5] and $\hat{y}$ = [0.25, 0.5, 0.25] would both have CE=0.69.

Backpropagation

The cross entropy loss of our entire training dataset would then be the average CE_i over all samples. For our training data, after our initial forward pass we'd have

Out[ ]//TraditionalForm=

$$\begin{pmatrix} 1 & 252 & 4 & 155 & 175 & 1 \\ 2 & 175 & 10 & 186 & 200 & 1 \\ 3 & 82 & 131 & 230 & 100 & 0 \\ 4 & 115 & 138 & 80 & 88 & 0 \end{pmatrix} \quad \hat{Y} = \begin{pmatrix} 0.498541 & 0.501237 \\ 0.498261 & 0.501739 \\ 0.497469 & 0.502531 \\ 0.49828 & 0.50172 \end{pmatrix} \quad \text{CE}_i = \begin{pmatrix} -0.693212 \\ -0.693141 \\ -0.693134 \\ -0.693141 \end{pmatrix}; \quad \text{CE} = -0.693157$$

To determine how "small" changes of the weights affects the current loss function we need to determine $\frac{\partial \text{CE}}{\partial (w^1)_{11}}$, ..., $\frac{\partial \text{CE}}{\partial (w^2)_{32}}$. Leaving aside parts of the derivation (check out the full derivation) we make use of the fact that

Out[ ]//TraditionalForm=

$$\boxed{\frac{d\,\text{sigmoid}(z)}{\text{dz}} = \text{sigmoid}(z)\,(1 - \text{sigmoid}(z))}$$

Recapping

we then have:

Out[∘]//TraditionalForm=

$$\frac{\partial \mathrm{CE}_1}{\partial Z_1^2} = \hat{Y}_1 - Y_1$$

Out[∘]//TraditionalForm=

$$\frac{\partial \mathrm{CE}_1}{\partial X_1^2} = \frac{\partial \mathrm{CE}_1}{\partial X_1^2} (W^2)^\mathsf{T}$$

Out[∘]//TraditionalForm=

$$\frac{\partial \mathrm{CE}_1}{\partial Z_{1,}^1} = \frac{\partial \mathrm{CE}_1}{\partial X_1^2} \otimes \left(X_{1,2:}^2 \otimes \left(1 - X_{1,2:}^2\right)\right)$$

Out[∘]//TraditionalForm=

$$\boxed{\frac{\partial \mathrm{CE}_1}{\partial W^2} = (X_{1,}^2)^\mathsf{T} \frac{\partial \mathrm{CE}_1}{\partial Z_{1,}^2}}$$

Out[∘]//TraditionalForm=

$$\boxed{\frac{\partial \mathrm{CE}_1}{\partial W^1} = (X_{1,}^1)^\mathsf{T} \frac{\partial \mathrm{CE}_1}{\partial Z_{1,}^1}}$$

Recapping

We can generalize these formulas to compute how the cross entropy changes for every training sample:

Out[◦]//TraditionalForm=

$$\nabla_{X^2}\mathrm{CE} = \nabla_{Z^2}\mathrm{CE}\,(w^2)^{\mathsf{T}}$$

Out[◦]//TraditionalForm=

$$\nabla_{Z^2}\,\mathrm{CE} = \hat{Y} - Y$$

Out[◦]//TraditionalForm=

$$\nabla_{X^2}\mathrm{CE} = \nabla_{Z^2}\mathrm{CE}\,(w^2)^{\mathsf{T}}$$

Out[◦]//TraditionalForm=

$$\nabla_{Z^1}\,\mathrm{CE} = \nabla_{X^2_{,2:}}\,\mathrm{CE} \otimes \left(X^2_{,2:} \otimes \left(1 - X^2_{,2:}\right)\right)$$

Out[◦]//TraditionalForm=

$$\boxed{\nabla_{w^2}\,\mathrm{CE} = (X^2)^{\mathsf{T}}\,\nabla_{Z^2}\,\mathrm{CE}}$$

Out[◦]//TraditionalForm=

$$\boxed{\nabla_{w^1}\,\mathrm{CE} = (X^1)^{\mathsf{T}}\,\nabla_{Z^1}\,\mathrm{CE}}$$

Recapping

This convenient, because X^1, W^2, W^2, and Y are already known to us and we already computed X^2 and $\hat{Y}$ during the forward pass. The reason is that we chose activation functions such that their derivative could be written as a function of their current value.

Out[∘]//TraditionalForm=

$$\nabla_{Z^2}\mathrm{CE} = \begin{pmatrix} 0.498541 & 0.501237 \\ 0.498261 & 0.501739 \\ 0.497469 & 0.502531 \\ 0.49828 & 0.50172 \end{pmatrix} - \begin{pmatrix} 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} -0.501459 & 0.501237 \\ -0.501739 & 0.501739 \\ 0.497469 & -0.497469 \\ 0.49828 & -0.49828 \end{pmatrix}$$

Out[∘]//TraditionalForm=

$$\nabla_{X^2}\mathrm{CE} = \begin{pmatrix} -0.501459 & 0.501237 \\ -0.501739 & 0.501739 \\ 0.497469 & -0.497469 \\ 0.49828 & -0.49828 \end{pmatrix} . \begin{pmatrix} -0.00588 & -0.00232 \\ -0.00647 & 0.0054 \\ 0.00374 & -0.00005 \end{pmatrix}^{\mathsf{T}} = \begin{pmatrix} 0.00178571 & 0.00595112 & -0.00190052 \\ 0.00178619 & 0.00595564 & -0.00190159 \\ -0.00177099 & -0.00590496 & 0.00188541 \\ -0.00177388 & -0.00591458 & 0.00188848 \end{pmatrix}$$

Recapping

Out[∘]//TraditionalForm=

$$\nabla_{Z^1}\text{CE} = \begin{pmatrix} 0.00109936 & -0.000351087 \\ 0.00145056 & -0.000463152 \\ -0.00122559 & 0.000391321 \\ -0.0012102 & 0.000386409 \end{pmatrix}$$

Out[∘]//TraditionalForm=

$$\nabla_{w^2}\text{CE} = \begin{pmatrix} 1 & 0.395579 & 0.755475 \\ 1 & 0.471449 & 0.58025 \\ 1 & 0.778414 & 0.706028 \\ 1 & 0.507457 & 0.713041 \end{pmatrix}^{\mathsf{T}} . \begin{pmatrix} -0.501459 & 0.501237 \\ -0.501739 & 0.501739 \\ 0.497469 & -0.497469 \\ 0.49828 & -0.49828 \end{pmatrix} = \begin{pmatrix} -0.00744934 & 0.0072271 \\ 0.205182 & -0.20527 \\ 0.036547 & -0.0367149 \end{pmatrix}$$

Out[∘]//TraditionalForm=

$$\nabla_{w^1}\text{CE} = \begin{pmatrix} 1 & 252 & 4 & 155 & 175 \\ 1 & 175 & 10 & 186 & 200 \\ 1 & 82 & 131 & 230 & 100 \\ 1 & 115 & 138 & 80 & 88 \end{pmatrix}^{\mathsf{T}} . \begin{pmatrix} 0.00109936 & -0.000351087 \\ 0.00145056 & -0.000463152 \\ -0.00122559 & 0.000391321 \\ -0.0012102 & 0.000386409 \end{pmatrix} = \begin{pmatrix} 0.000114129 & -0.0000365093 \\ 0.291216 & -0.0930002 \\ -0.308657 & 0.0985516 \\ 0.0615033 & -0.0196482 \\ 0.253443 & -0.0809345 \end{pmatrix}$$

Recapping

Now we have the gradient and can update the weights choosing a certain stepsize

Out[•]//TraditionalForm=

$$w^1 := w^1 - \text{stepsize}\, \nabla_{w^1} \text{CE}$$

Out[•]//TraditionalForm=

$$w^2 := w^2 - \text{stepsize}\, \nabla_{w^2} \text{CE}$$

Out[•]//TraditionalForm=

$$w^1 := \begin{pmatrix} -0.00469 & 0.00797 \\ -0.00256 & 0.00889 \\ 0.00146 & 0.00322 \\ 0.00816 & 0.00258 \\ -0.00597 & -0.00876 \end{pmatrix} - 0.1 \begin{pmatrix} 0.000114129 & -0.0000365093 \\ 0.291216 & -0.0930002 \\ -0.308657 & 0.0985516 \\ 0.0615033 & -0.0196482 \\ 0.253443 & -0.0809345 \end{pmatrix} = \begin{pmatrix} -0.00470141 & 0.00797365 \\ -0.0316816 & 0.01819 \\ 0.0323257 & -0.00663516 \\ 0.00200967 & 0.00454482 \\ -0.0313143 & -0.000666547 \end{pmatrix}$$

Out[•]//TraditionalForm=

$$w^2 := \begin{pmatrix} -0.00588 & -0.00232 \\ -0.00647 & 0.0054 \\ 0.00374 & -0.00005 \end{pmatrix} - 0.1 \begin{pmatrix} -0.00744934 & 0.0072271 \\ 0.205182 & -0.20527 \\ 0.036547 & -0.0367149 \end{pmatrix} = \begin{pmatrix} -0.00513507 & -0.00304271 \\ -0.0269882 & 0.025927 \\ 0.0000852996 & 0.00362149 \end{pmatrix}$$

Recapping

With these new weights you re-compute the steps from above and iteratively try to reach an optimal solution.

```
In[•]:= Length[train]

Out[•]= 400
```

```
In[•]:= trainingData = Thread[train[[1 ;; 350, 2 ;; 5]] -> train[[1 ;; 350, -1]]];
        testData = Thread[train[[351 ;; -1, 2 ;; 5]] -> train[[351 ;; -1, -1]]];
```

Recapping

In[∘]:=
```
net = NetChain[{LinearLayer[30], ElementwiseLayer[Ramp], LinearLayer[2], SoftmaxLayer[]},
  "Input" → {4}, "Output" → NetDecoder[{"Class", {1, 0}}]]
```

Out[∘]=
```
NetChain[
  uninitialized       Input          vector (size: 4)
                    1 LinearLayer    vector (size: 30)
                    2 Ramp           vector (size: 30)
                    3 LinearLayer    vector (size: 2)
                    4 SoftmaxLayer   vector (size: 2)
                      Output         class
]
```

In[∘]:=
```
CompleteGraph[net]
```

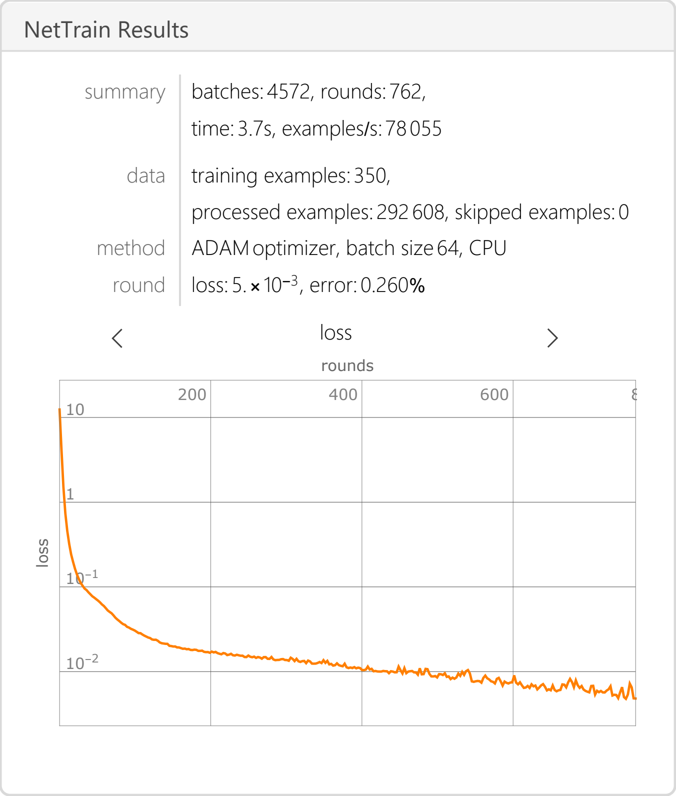
```
CompleteGraph: Single or list of positive machine–sized integers expected at position 1 of
   CompleteGraph[NetChain[<|Type –> Chain, Nodes –> <|1 –> <|Type –> Linear, Arrays –> <|<<2>>|>, <<2>>, Outputs –> <|Output –>
      NeuralNetworks`TensorT[{30}, NeuralNetworks`RealT]|>|>, <<2>>, 4 –> <|<<5>>|>|>, <<2>>, Outputs –> <|Output –> NetDecoder[{"Clas
      …}]|>|>, <|<<2>>|>]]

   .
```

Out[∘]=
```
CompleteGraph[NetChain[
  uninitialized       Input          vector (size: 4)
                    1 LinearLayer    vector (size: 30)
                    2 Ramp           vector (size: 30)
                    3 LinearLayer    vector (size: 2)
                    4 SoftmaxLayer   vector (size: 2)
                      Output         class
]]
```

Recapping

```
In[∘]:= results = NetTrain[net, trainingData, All]
        trained = results["TrainedNet"]
```

Out[∘]=



NetTrain Results

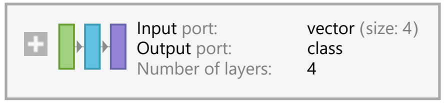
summary: batches: 4572, rounds: 762,
time: 3.7s, examples/s: 78 055

data: training examples: 350,
processed examples: 292 608, skipped examples: 0

method: ADAM optimizer, batch size 64, CPU

round: loss: 5. × 10^{-3}, error: 0.260%

Out[∘]= NetChain[



Input port: vector (size: 4)
Output port: class
Number of layers: 4

]

Recapping

```
In[∘]:= TableForm[Transpose@{trained /@ train[[351 ;; -1, 2 ;; 5]], train[[351 ;; -1, -1]]}]
```

```
Out[∘]//TableForm=
        1    1
        0    0
        1    1
        0    0
        0    0
        1    1
        0    0
        1    1
        0    0
        1    1
        1    1
        0    0
        0    0
        1    1
        0    0
        1    1
        0    0
        1    1
        1    1
        0    0
        1    1
        0    0
        0    0
        0    0
        0    0
        1    1
        1    1
        0    0
        0    0
        0    0
        1    1
        1    1
        0    0
        1    0
        1    1
        0    0
        1    1
        1    1
        1    1
```

```
0    0
0    0
1    1
0    0
1    1
0    0
1    1
1    1
0    0
1    1
0    0
```

```
In[∘]:= display[train[[-17]]]
```

Out[∘]= {482, 18, 177, 233, 240, , False}

```
In[∘]:= trained@{18, 177, 233, 240}

Out[∘]= 1
```

Example: Object classification

Source: https://resources.wolframcloud.com/NeuralNetRepository/resources/YOLO-V2-Trained-on-MS-COCO-Data_ 1

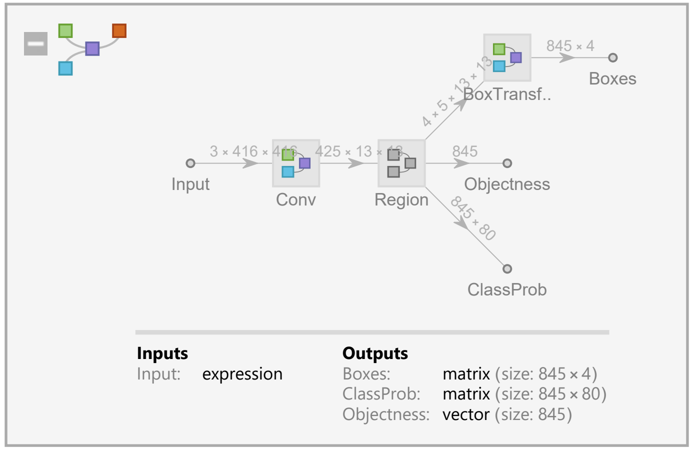
YOLO (You Only Look Once) Version 2 is an object detection model published by Joseph Redmon and Ali Farhadi in December 2016. It is a single-stage architecture that goes straight from image pixels to bounding box coordinates and class probabilities. Compared to its predecessor, it introduces batch normalization, raises the image resolution and switches from direct coordinate prediction to anchor boxes' offsets. On an NVIDIA Titan X, it processes images at 40-90 FPS.

Resource retrieval

Get the pre-trained net:

In[∘]:=
```
NetModel["YOLO V2 Trained on MS-COCO Data"]
```

Out[∘]= NetGraph[



Inputs		**Outputs**	
Input:	expression	Boxes:	matrix (size: 845 × 4)
		ClassProb:	matrix (size: 845 × 80)
		Objectness:	vector (size: 845)

]

Evaluation function

Write an evaluation function to scale the result to the input image size and suppress the least probable detections:

```
In[∘]:= nonMaxSuppression[overlapThreshold_][detection_] :=
        Module[{boxes, confidence},
            Fold[
                {list, new} ↦ If[NoneTrue[list[[All, 1]], IoU[#, new[[1]]] > overlapThreshold &], Append[list, new], list],
                Sequence @@ TakeDrop[Reverse@SortBy[detection, Last], 1]
            ]
        ]
        ClearAll[IoU]
        IoU := IoU =
        With[{c = Compile[
            {{box1, _Real, 2}, {box2, _Real, 2}},
            Module[{area1, area2, x1, y1, x2, y2, w, h, int},

                area1 = (box1[[2, 1]] - box1[[1, 1]]) (box1[[2, 2]] - box1[[1, 2]]);
                area2 = (box2[[2, 1]] - box2[[1, 1]]) (box2[[2, 2]] - box2[[1, 2]]);

                x1 = Max[box1[[1, 1]], box2[[1, 1]]];
                y1 = Max[box1[[1, 2]], box2[[1, 2]]];
                x2 = Min[box1[[2, 1]], box2[[2, 1]]];
                y2 = Min[box1[[2, 2]], box2[[2, 2]]];

                w = Max[0., x2 - x1];
                h = Max[0., y2 - y1];

                int = w * h;

                int / (area1 + area2 - int)
            ],
            RuntimeAttributes -> {Listable},
            Parallelization -> True,
            RuntimeOptions -> "Speed"
        ]},
            c @@ Replace[{##}, Rectangle → List, Infinity, Heads → True] &
        ]
```

```
In[•]:= netevaluate[img_, detectionThreshold_ : .2, overlapThreshold_ : .4] :=
  Module[
      {w, h, scale, coords, obj, classes, boxes, padding, classProb, bestClass, probable, finals},

      {coords, classes, obj} = Values@NetModel["YOLO V2 Trained on MS-COCO Data"][img];

      (* transform coordinates into rectangular boxes *)
      {w, h} = ImageDimensions[img];
      scale = Max[{w, h}] / 416;
      padding = 416 (1 - {w, h} / Max[{w, h}]) / 2;
      boxes = Apply[
           Rectangle[
                scale {#1 - #3/2 - padding[[1]], 416 - #2 - #4/2 - padding[[2]]},
                scale {#1 + #3/2 - padding[[1]], 416 - #2 + #4/2 - padding[[2]]}
            ] &,
           416 coords,
       1];

      (* each class probability is rescaled with the box objectivness *)
      classProb = classes * obj;
      (* filter by probability*)
      (* very small probability are thresholded *)
      probable = Position[Max /@ classProb , p_ /; p - 10^-2 > detectionThreshold];
      If[Length[probable] == 0, Return[{}]];

      (* gather the boxes of the same class and perform non-max suppression *)
      bestClass = Last@*Ordering /@ classProb;
      finals = Join @@ Values @ GroupBy[
            MapThread[
                 {#1, #2, #3[[#2]]} &,
                 {Extract[boxes, probable], Extract[bestClass, probable], Extract[classProb, probable]}
            ],
            #[[2]] &, nonMaxSuppression[overlapThreshold]
        ]
  ]
```

Label list

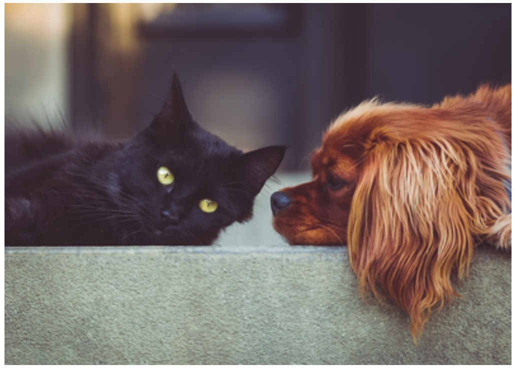
Define the label list for this model. Integers in the model’s output correspond to elements in the label list:

In[ ]:=
```
labels = {"person", "bicycle", "car", "motorcycle", "airplane", "bus", "train", "truck", "boat", "traffic light", "fire hydrant",
   "stop sign", "parking meter", "bench", "bird", "cat", "dog", "horse", "sheep", "cow", "elephant", "bear", "zebra", "giraffe",
   "backpack", "umbrella", "handbag", "tie", "suitcase", "frisbee", "skis", "snowboard", "sports ball", "kite", "baseball bat",
   "baseball glove", "skateboard", "surfboard", "tennis racket", "bottle", "wine glass", "cup", "fork", "knife", "spoon",
   "bowl", "banana", "apple", "sandwich", "orange", "broccoli", "carrot", "hot dog", "pizza", "donut", "cake", "chair", "couch",
   "potted plant", "bed", "dining table", "toilet", "tv", "laptop", "mouse", "remote", "keyboard", "cell phone", "microwave",
   "oven", "toaster", "sink", "refrigerator", "book", "clock", "vase", "scissors", "teddy bear", "hair drier", "toothbrush"};
```

Basic usage

Obtain the detected bounding boxes with their corresponding classes and confidences for a given image:

```
In[]:= testImage =
```



```
;
```

```
In[]:= detection = netevaluate[testImage]

Out[]= {{Rectangle[{199.925, 22.0704}, {399.521, 209.118}], 17, 0.84133},
        {Rectangle[{7.75462, 88.1155}, {216.116, 235.609}], 16, 0.926401}}
```

Evaluation function

Inspect which classes are detected:

```
In[ ]:=  classes = DeleteDuplicates@detection[[All, 2]]
Out[ ]=  {17, 16}
```

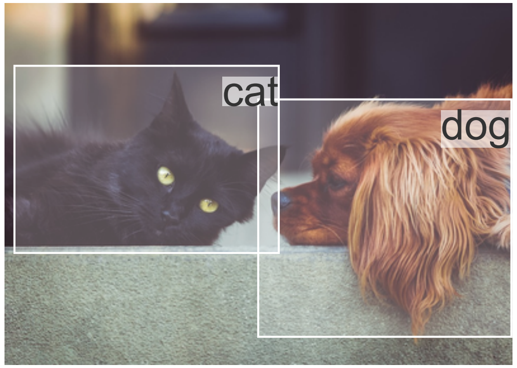
```
In[ ]:=  labels[[classes]]
Out[ ]=  {dog, cat}
```

Evaluation function

Visualize the detection:

In[ ]:=
```
HighlightImage[testImage,
 MapThread[{White, Inset[Style[labels[[#2]], Black, FontSize → Scaled[1/12], Background → GrayLevel[1, .6]],
     Last[#1], {Right, Top}], #1} &, Transpose@detection]]
```

Out[ ]=



Network result

The network computes 845 bounding boxes, the probability of having an object in each box and the conditioned probability that the object is of any given class:

```
In[]:= res = NetModel["YOLO V2 Trained on MS-COCO Data"][testImage]
```

```
Out[]= <|Boxes → {{0.0271295, 0.0479492, 0.0325754, 0.0351086}, … 843 …, {0.948957, 0.947914, 0.741345, 0.866989}},
        … 1 …, Objectness → {0.0000290822, … 843 …, … 22 …}|>

        large output | show less | show more | show all | set size limit...
```

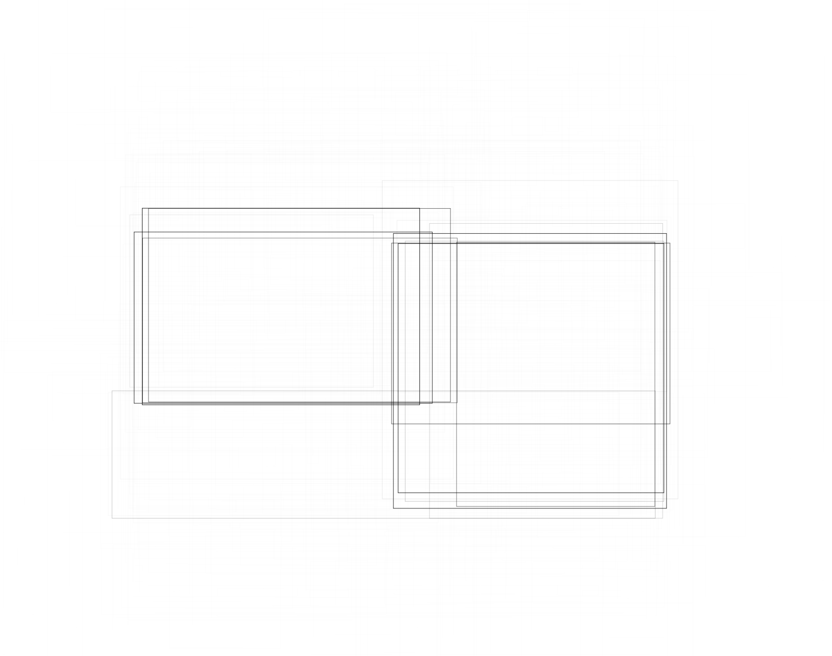
Network result

Visualize all the boxes predicted by the net scaled by their “objectness” measures:

```
In[ ]:= rectangles = Apply[Rectangle[{#1 - #3 / 2, 1 - #2 - #4 / 2}, {#1 + #3 / 2, 1 - #2 + #4 / 2}] &, res["Boxes"], 1];
```

In[∘]:=
```
Graphics[
 MapThread[{EdgeForm[Opacity[#1 + .01]], #2} &, {res["Objectness"], rectangles}],
 BaseStyle → {FaceForm[], EdgeForm[{Thin, Black}]}
]
```

Out[∘]=



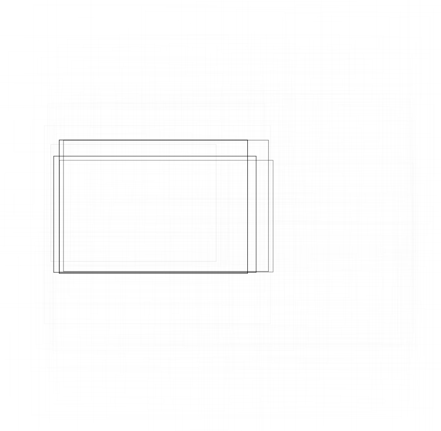
Network result

Visualize all the boxes scaled by the probability that they contain a cat:

```
In[•]:= idx = Position[labels, "cat"][[1, 1]]

Out[•]= 16
```

```
In[•]:= Graphics[
          MapThread[{EdgeForm[Opacity[#1 + .01]], #2} &, {res["Objectness"] × Extract[res["ClassProb"], {All, idx}], rectangles}],
          BaseStyle → {FaceForm[], EdgeForm[{Thin, Black}]}
         ]
```

Out[•]=

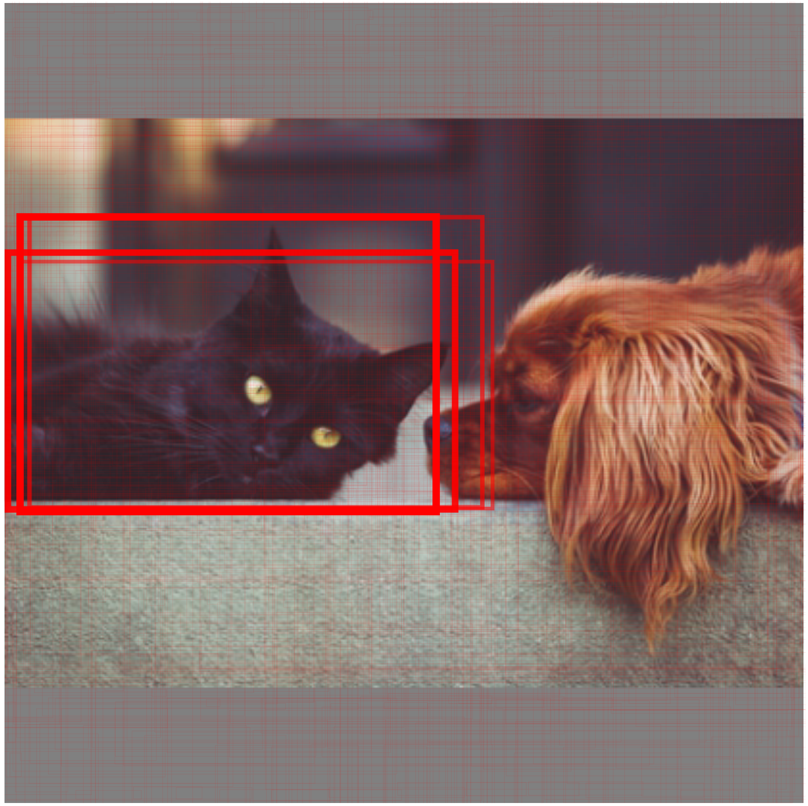


Network result

Superimpose the cat prediction on top of the scaled input received by the net:

In[ ]:=
```
HighlightImage[
 Image[NetExtract[NetModel["YOLO V2 Trained on MS-COCO Data"], "Input"][testImage], Interleaving → False],
 MapThread[{EdgeForm[{Thickness[#1 / 100], Opacity[#1 + .1]}], #2} &,
  {res["Objectness"] × Extract[res["ClassProb"], {All, idx}], rectangles}],
 BaseStyle → {FaceForm[], EdgeForm[{Thin, Red}]},
 DataRange → {{0, 1}, {0, 1}}
]
```

Out[ ]=



Advanced visualization

Write a function to apply a custom styling to the result of the detection:

```
In[∘]:= styleDetection[detection_] :=
          Values[GroupBy[detection, #1⟦2⟧ &, With[{c = RandomColor[]}, ({c, #1⟦1⟧, Inset[Style[labels⟦#1⟦2⟧⟧, FontSize → Scaled[1/28],
                    Black], Last[#1⟦1⟧], {Right, Top}, Background → GrayLevel[1, .8]]} &) /@ #1] &]];
```

Visualize multiple objects, using a different color for each class:

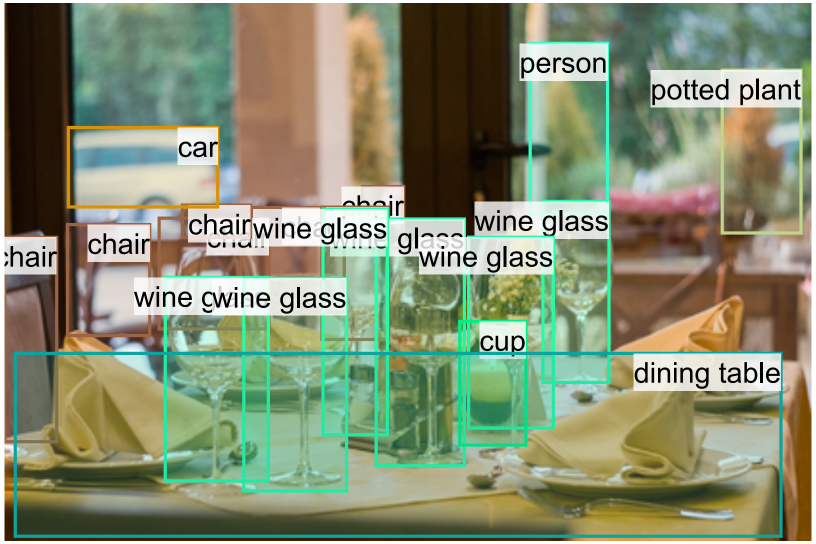
```
In[∘]:= image =
```

```
;
```

Advanced visualization

In[◦]:=
```
HighlightImage[image, styleDetection[netevaluate[image, 0.2, .4]]]
```

Out[◦]=



Net information

Inspect the number of parameters of all arrays in the net:

```
In[•]:= NetInformation[NetModel["YOLO V2 Trained on MS-COCO Data"], "ArraysElementCounts"]
```

```
Out[•]= <|{BoxTransformation, Anchors, Array} → 1690, {BoxTransformation, Grid, Array} → 1690,
 {BoxTransformation, Scale, Array} → 3380, {Conv, conv1024, 10, Biases} → 1024, {Conv, conv1024, 10, MovingMean} → 1024,
 {Conv, conv1024, 10, MovingVariance} → 1024, {Conv, conv1024, 10, Scaling} → 1024, {Conv, conv1024, 12, Biases} → 512,
 {Conv, conv1024, 12, Weights} → 524 288, {Conv, conv1024, 13, Biases} → 512, {Conv, conv1024, 13, MovingMean} → 512,
 {Conv, conv1024, 13, MovingVariance} → 512, {Conv, conv1024, 13, Scaling} → 512, {Conv, conv1024, 15, Biases} → 1024,
 {Conv, conv1024, 15, Weights} → 4 718 592, {Conv, conv1024, 16, Biases} → 1024, {Conv, conv1024, 16, MovingMean} → 1024,
 {Conv, conv1024, 16, MovingVariance} → 1024, {Conv, conv1024, 16, Scaling} → 1024, {Conv, conv1024, 3, Biases} → 1024,
 {Conv, conv1024, 3, Weights} → 4 718 592, {Conv, conv1024, 4, Biases} → 1024, {Conv, conv1024, 4, MovingMean} → 1024,
 {Conv, conv1024, 4, MovingVariance} → 1024, {Conv, conv1024, 4, Scaling} → 1024, {Conv, conv1024, 6, Biases} → 512,
 {Conv, conv1024, 6, Weights} → 524 288, {Conv, conv1024, 7, Biases} → 512, {Conv, conv1024, 7, MovingMean} → 512,
 {Conv, conv1024, 7, MovingVariance} → 512, {Conv, conv1024, 7, Scaling} → 512, {Conv, conv1024, 9, Biases} → 1024,
 {Conv, conv1024, 9, Weights} → 4 718 592, {Conv, conv128, 10, Biases} → 128, {Conv, conv128, 10, MovingMean} → 128,
 {Conv, conv128, 10, MovingVariance} → 128, {Conv, conv128, 10, Scaling} → 128, {Conv, conv128, 3, Biases} → 128,
 {Conv, conv128, 3, Weights} → 73 728, {Conv, conv128, 4, Biases} → 128, {Conv, conv128, 4, MovingMean} → 128,
 {Conv, conv128, 4, MovingVariance} → 128, {Conv, conv128, 4, Scaling} → 128, {Conv, conv128, 6, Biases} → 64,
 {Conv, conv128, 6, Weights} → 8192, {Conv, conv128, 7, Biases} → 64, {Conv, conv128, 7, MovingMean} → 64,
 {Conv, conv128, 7, MovingVariance} → 64, {Conv, conv128, 7, Scaling} → 64, {Conv, conv128, 9, Biases} → 128,
 {Conv, conv128, 9, Weights} → 73 728, {Conv, conv256, 10, Biases} → 256, {Conv, conv256, 10, MovingMean} → 256,
 {Conv, conv256, 10, MovingVariance} → 256, {Conv, conv256, 10, Scaling} → 256, {Conv, conv256, 3, Biases} → 256,
 {Conv, conv256, 3, Weights} → 294 912, {Conv, conv256, 4, Biases} → 256, {Conv, conv256, 4, MovingMean} → 256,
 {Conv, conv256, 4, MovingVariance} → 256, {Conv, conv256, 4, Scaling} → 256, {Conv, conv256, 6, Biases} → 128,
 {Conv, conv256, 6, Weights} → 32 768, {Conv, conv256, 7, Biases} → 128, {Conv, conv256, 7, MovingMean} → 128,
 {Conv, conv256, 7, MovingVariance} → 128, {Conv, conv256, 7, Scaling} → 128, {Conv, conv256, 9, Biases} → 256,
 {Conv, conv256, 9, Weights} → 294 912, {Conv, conv32, 1, Biases} → 32, {Conv, conv32, 1, Weights} → 864,
 {Conv, conv32, 2, Biases} → 32, {Conv, conv32, 2, MovingMean} → 32, {Conv, conv32, 2, MovingVariance} → 32,
 {Conv, conv32, 2, Scaling} → 32, {Conv, conv512, 10, Biases} → 512, {Conv, conv512, 10, MovingMean} → 512,
 {Conv, conv512, 10, MovingVariance} → 512, {Conv, conv512, 10, Scaling} → 512, {Conv, conv512, 12, Biases} → 256,
 {Conv, conv512, 12, Weights} → 131 072, {Conv, conv512, 13, Biases} → 256, {Conv, conv512, 13, MovingMean} → 256,
 {Conv, conv512, 13, MovingVariance} → 256, {Conv, conv512, 13, Scaling} → 256, {Conv, conv512, 15, Biases} → 512,
 {Conv, conv512, 15, Weights} → 1 179 648, {Conv, conv512, 16, Biases} → 512, {Conv, conv512, 16, MovingMean} → 512,
 {Conv, conv512, 16, MovingVariance} → 512, {Conv, conv512, 16, Scaling} → 512, {Conv, conv512, 3, Biases} → 512,
 {Conv, conv512, 3, Weights} → 1 179 648, {Conv, conv512, 4, Biases} → 512, {Conv, conv512, 4, MovingMean} → 512,
{Conv, conv512, 4, MovingVariance} → 512, {Conv, conv512, 4, Scaling} → 512, {Conv, conv512, 6, Biases} → 256,
{Conv, conv512, 6, Weights} → 131 072, {Conv, conv512, 7, Biases} → 256, {Conv, conv512, 7, MovingMean} → 256,
{Conv, conv512, 7, MovingVariance} → 256, {Conv, conv512, 7, Scaling} → 256, {Conv, conv512, 9, Biases} → 512,
{Conv, conv512, 9, Weights} → 1 179 648, {Conv, conv64, 3, Biases} → 64, {Conv, conv64, 3, Weights} → 18 432,
{Conv, conv64, 4, Biases} → 64, {Conv, conv64, 4, MovingMean} → 64, {Conv, conv64, 4, MovingVariance} → 64,
{Conv, conv64, 4, Scaling} → 64, {Conv, conv_final_1, 1, Biases} → 1024, {Conv, conv_final_1, 1, Weights} → 9 437 184,
{Conv, conv_final_1, 2, Biases} → 1024, {Conv, conv_final_1, 2, MovingMean} → 1024, {Conv, conv_final_1, 2, MovingVariance} → 1024,
{Conv, conv_final_1, 2, Scaling} → 1024, {Conv, conv_final_1, 4, Biases} → 1024, {Conv, conv_final_1, 4, Weights} → 9 437 184,
{Conv, conv_final_1, 5, Biases} → 1024, {Conv, conv_final_1, 5, MovingMean} → 1024, {Conv, conv_final_1, 5, MovingVariance} → 1024,
{Conv, conv_final_1, 5, Scaling} → 1024, {Conv, conv_final_2, 1, Biases} → 64, {Conv, conv_final_2, 1, Weights} → 32 768,
{Conv, conv_final_2, 2, Biases} → 64, {Conv, conv_final_2, 2, MovingMean} → 64, {Conv, conv_final_2, 2, MovingVariance} → 64,
{Conv, conv_final_2, 2, Scaling} → 64, {Conv, conv_final_3, 1, Biases} → 1024, {Conv, conv_final_3, 1, Weights} → 11 796 480,
{Conv, conv_final_3, 2, Biases} → 1024, {Conv, conv_final_3, 2, MovingMean} → 1024, {Conv, conv_final_3, 2, MovingVariance} → 1024,
{Conv, conv_final_3, 2, Scaling} → 1024, {Conv, end, 1, Biases} → 425, {Conv, end, 1, Weights} → 435 200|⟩
```

Net information

Obtain the total number of parameters:

```
In[•]:= NetInformation[NetModel["YOLO V2 Trained on MS-COCO Data"], "ArraysTotalElementCount"]
Out[•]= 51 000 657
```

Net information

Obtain the layer type counts:

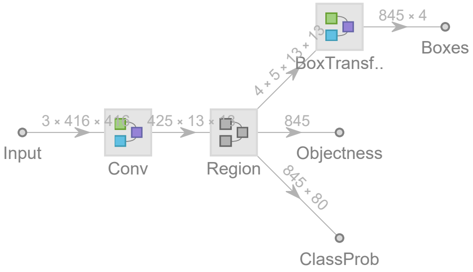
```
In[∘]:= NetInformation[NetModel["YOLO V2 Trained on MS-COCO Data"], "LayerTypeCounts"]

Out[∘]= ⟨|ConvolutionLayer → 23, BatchNormalizationLayer → 22, ElementwiseLayer → 25,
         PaddingLayer → 5, PoolingLayer → 5, ReshapeLayer → 3, TransposeLayer → 4, CatenateLayer → 3, PartLayer → 6,
         SoftmaxLayer → 1, FlattenLayer → 3, ConstantArrayLayer → 3, TotalLayer → 1, ThreadingLayer → 2|⟩
```

Net information

Display the summary graphic:

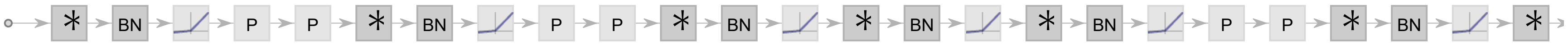
In[◦]:=
```
NetInformation[NetModel["YOLO V2 Trained on MS-COCO Data"], "SummaryGraphic"]
```

Out[◦]=



In[◦]:=
```
NetInformation[NetModel["YOLO V2 Trained on MS-COCO Data"], "FullSummaryGraphic"]
```

Out[◦]=



Export to MXNet

`Export` the net into a format that can be opened in MXNet:

```
In[∘]:= jsonPath = Export[FileNameJoin[{$TemporaryDirectory, "net.json"}], NetModel["YOLO V2 Trained on MS-COCO Data"], "MXNet"]
Out[∘]= /private/var/folders/pz/94mxs33x21512z6wtjbthvy0000_ck/T/net.json
```

`Export` also creates a *net.params* file containing parameters:

```
In[∘]:= paramPath = FileNameJoin[{DirectoryName[jsonPath], "net.params"}]
Out[∘]= /private/var/folders/pz/94mxs33x21512z6wtjbthvy0000_ck/T/net.params
```

Get the size of the parameter file:

```
In[∘]:= FileByteCount[paramPath]
Out[∘]= 204 012 145
```

The size is similar to the byte count of the resource object:

```
In[∘]:= ResourceObject["YOLO V2 Trained on MS-COCO Data"]["ByteCount"]
Out[∘]= 204 595 856
```

Additional Resources

Nice interactive demonstration: http://playground.tensorflow.org

Adventures learning Neural Nets and Python

● Init