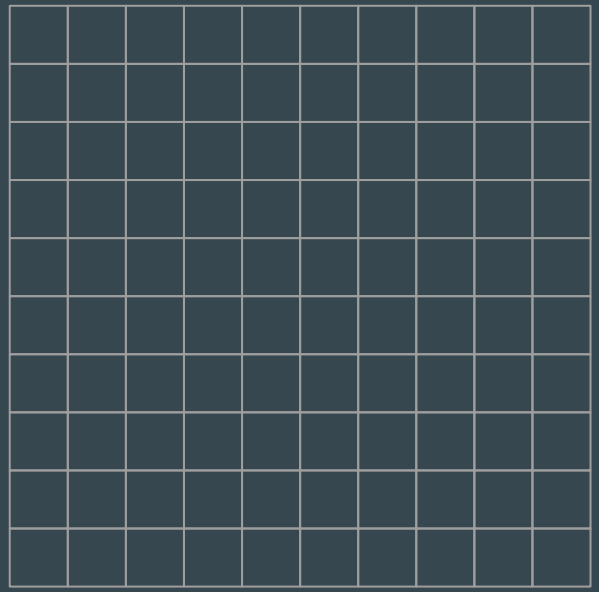
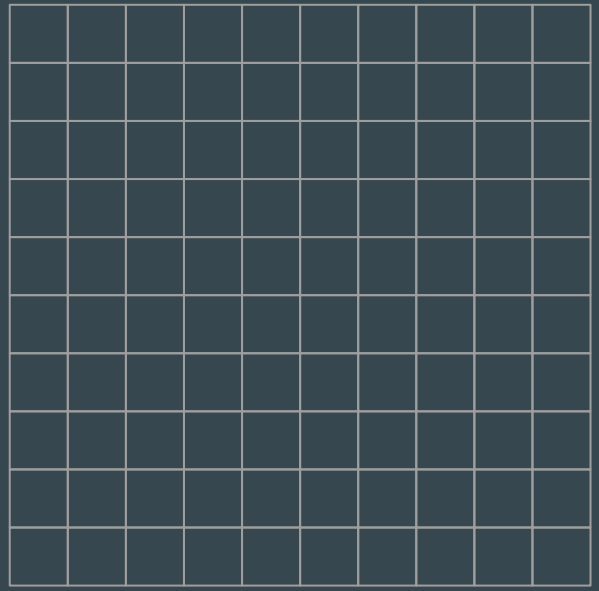
GPUs beyond deep learning and gaming

•••

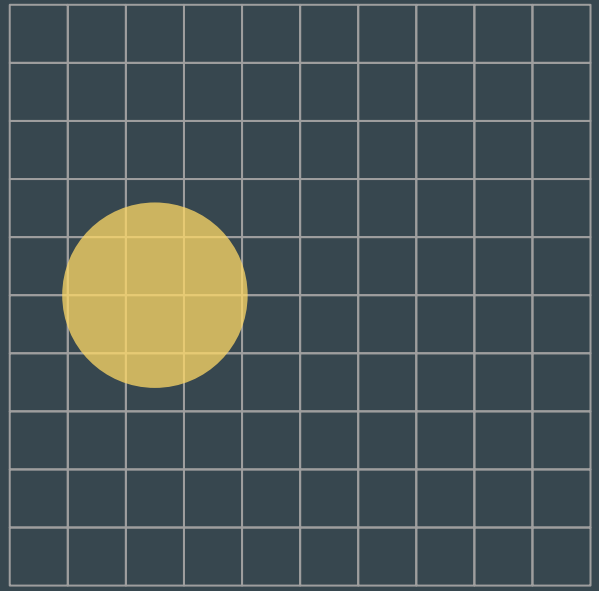
Solving hard problems with back propagation



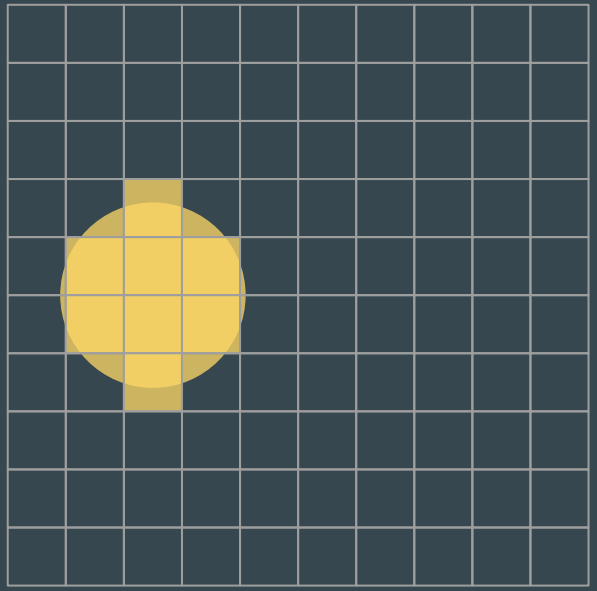
Given a radius, a color, and x,y coordinates, generate the resulting image.



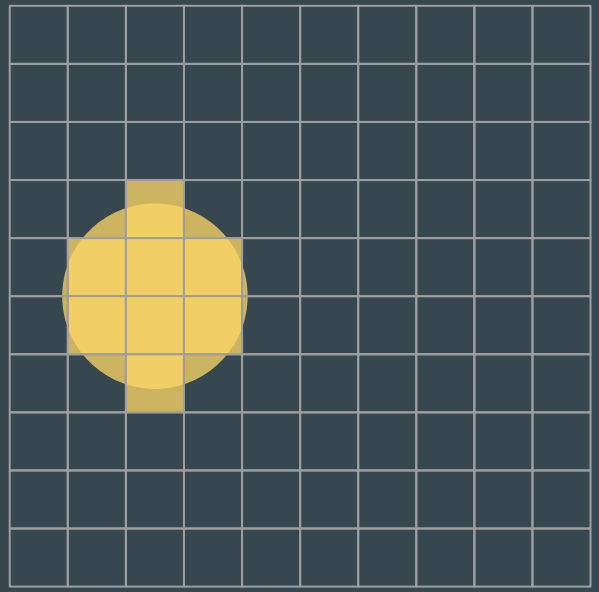
Given a radius, a color, and x,y coordinates, generate the resulting image.



Given a radius, a color, and x,y coordinates, generate the resulting image.

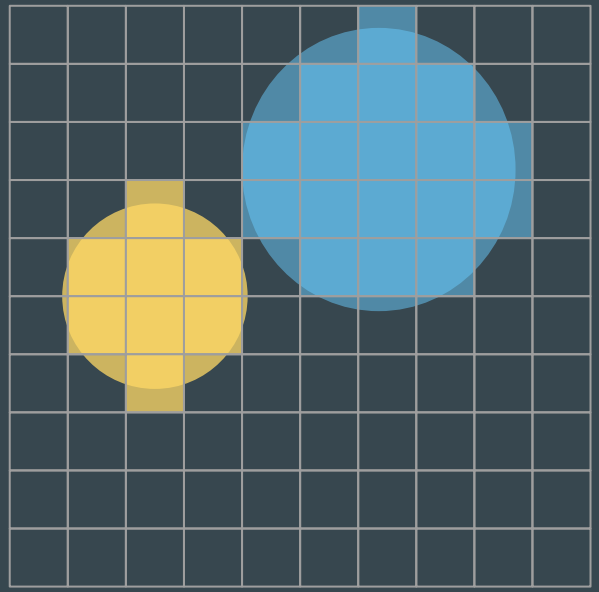


Given a radius, a color, and x,y coordinates, generate the resulting image.



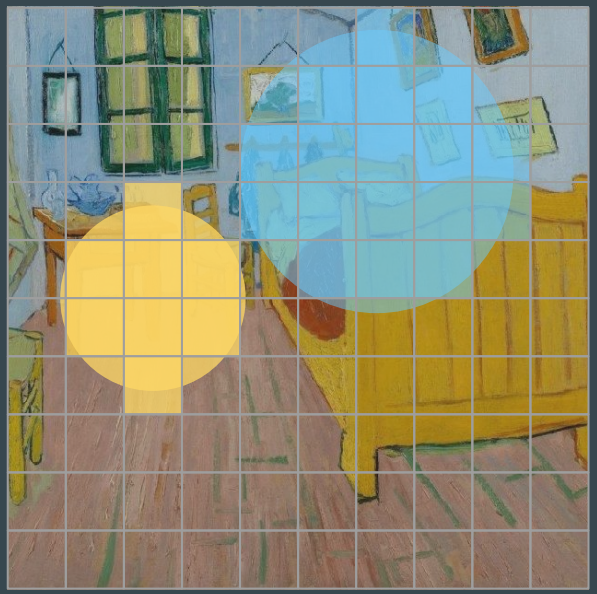
```
for pixel in canvas:
   if dist(pixel, (x,y)) <= radius:
      Pixel = color
```

Given a radius, a color, and x,y coordinates, generate the resulting image.



```
for n in N:
   for pixel in canvas:
      if dist(pixel, (x[n],y[n])) <= radius[n]:
         Pixel += color[n]
```

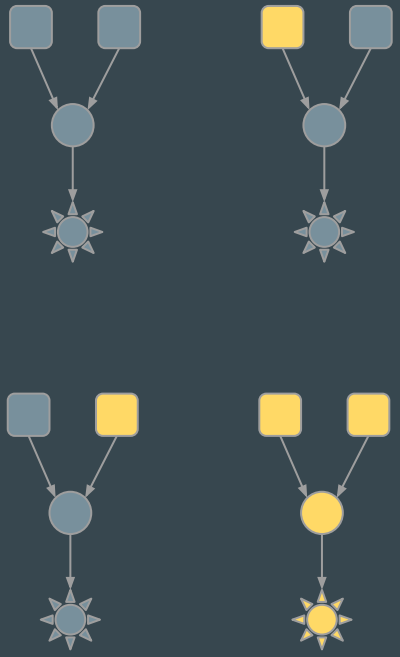
Given a radius, a color, and x,y coordinates, generate the resulting image.



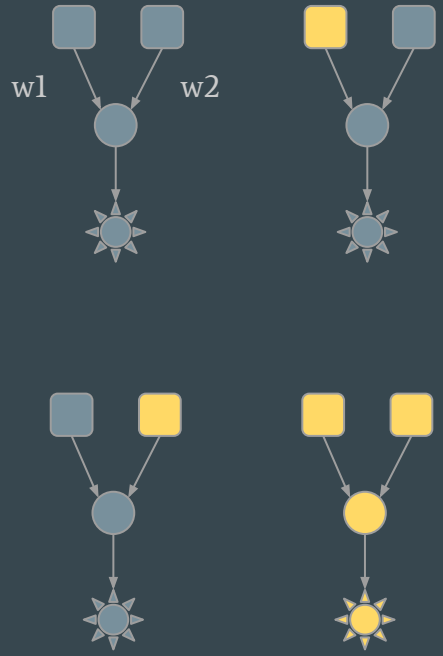
```
for n in N:
   for pixel in canvas:
      if dist(pixel, (x[n],y[n])) <= radius[n]:
         Pixel += color[n]
```

The shortest Intro to training Neural Networks

The Lightbulb

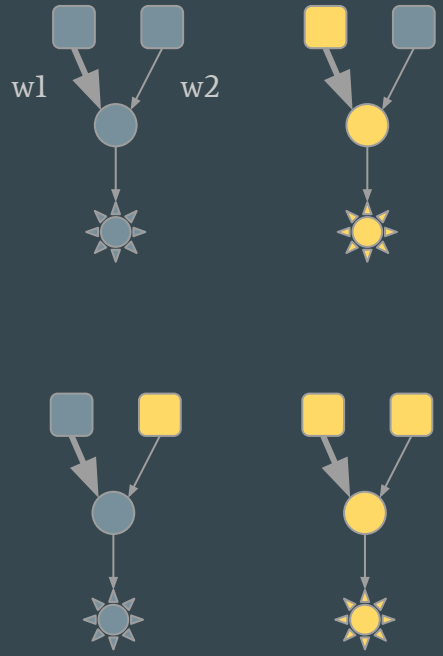


The Lightbulb




```
if ((w1*x) + (w2*y)) > threshold:
    return true
else:
    return false
```

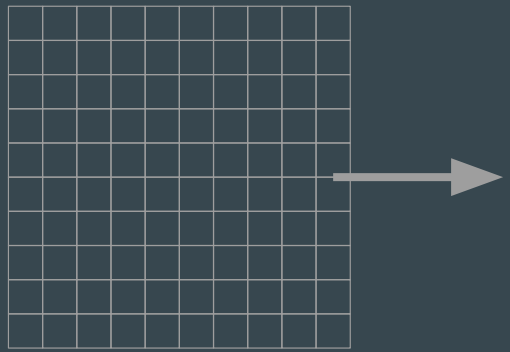
The Lightbulb




```
if ((w1*x) + (w2*y)) > threshold:
    return true
else:
    return false
```

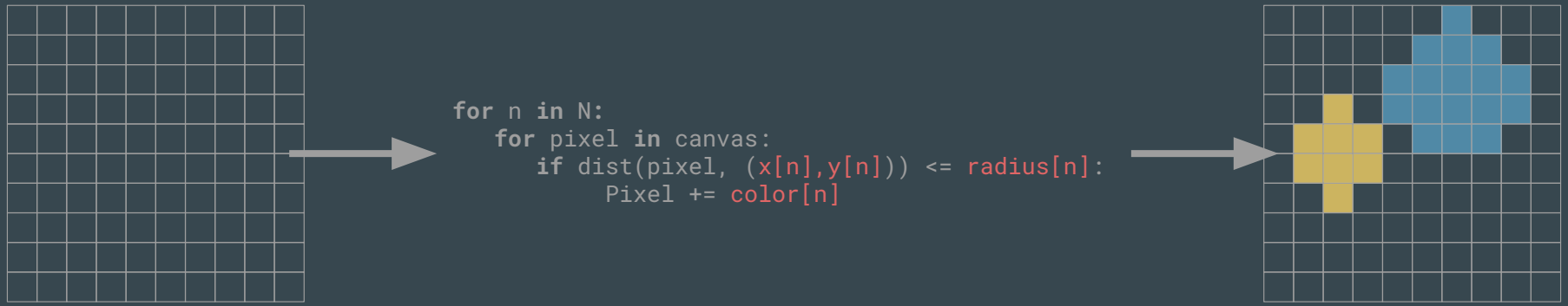
Can we put anything inside?

Back to the flashlights



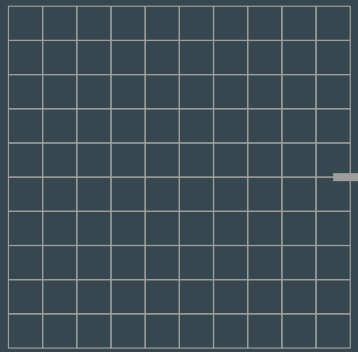
```
for n in N:
   for pixel in canvas:
      if dist(pixel, (x[n],y[n])) <= radius[n]:
           Pixel += color[n]
```

Back to the flashlights

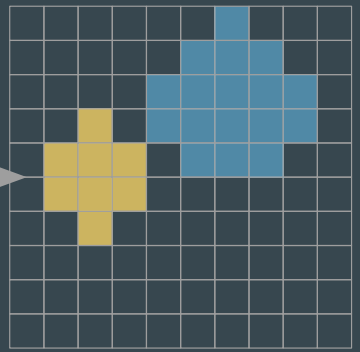


```
for n in N:
   for pixel in canvas:
      if dist(pixel, (x[n],y[n])) <= radius[n]:
           Pixel += color[n]
```

Back to the flashlights

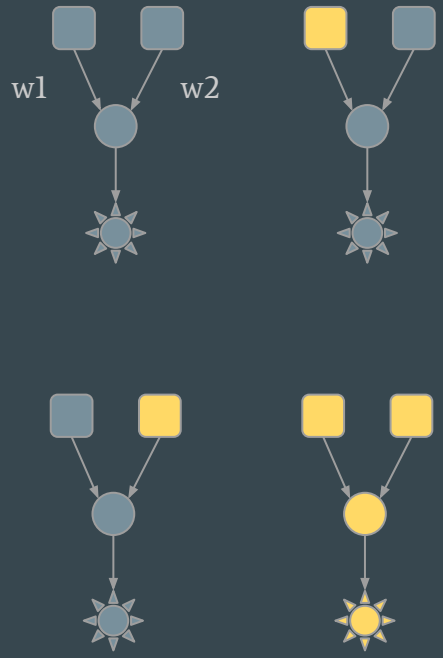


```
for n in N:
   for pixel in canvas:
      if dist(pixel, (x[n],y[n])) <= radius[n]:
           Pixel += color[n]
```



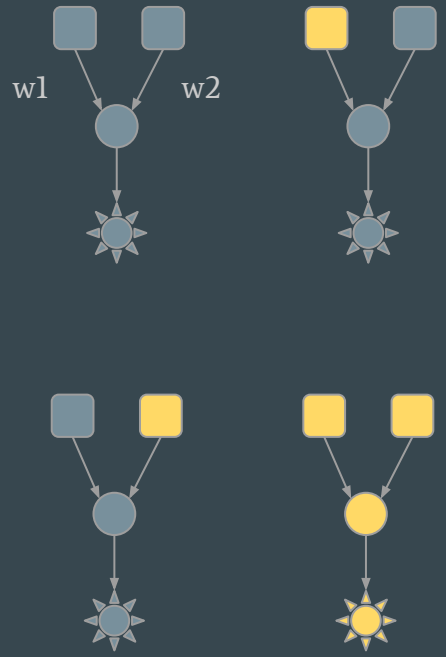
Live Coding

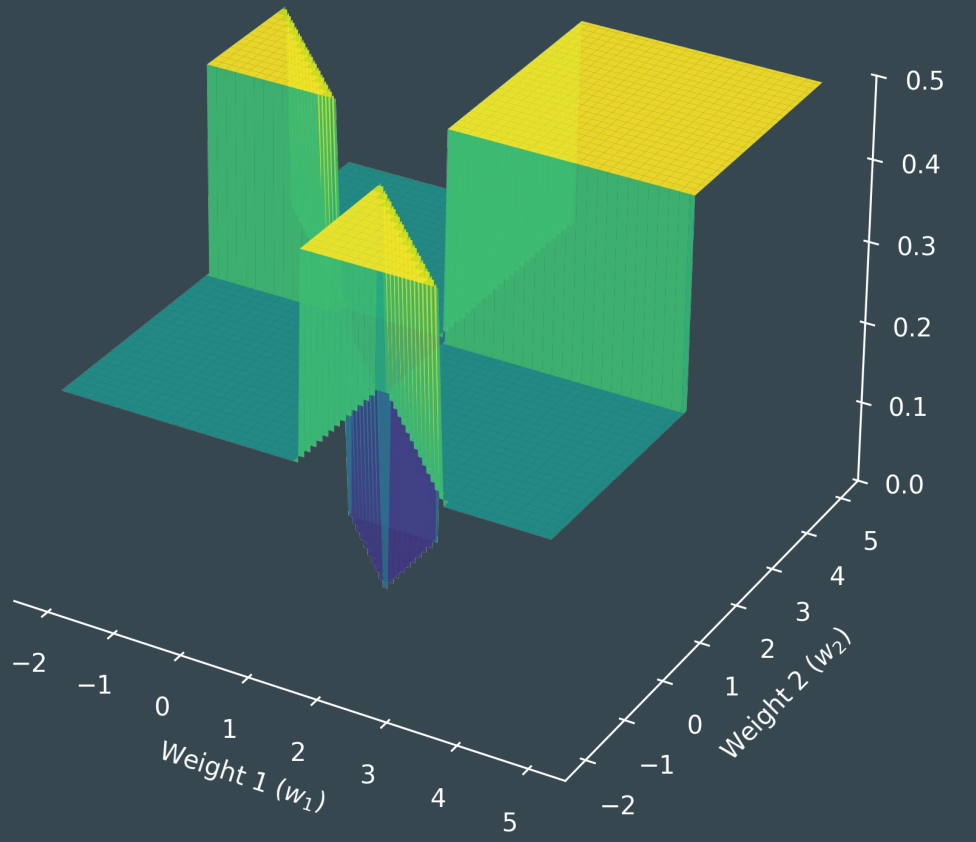
Remember the Lightbulb?



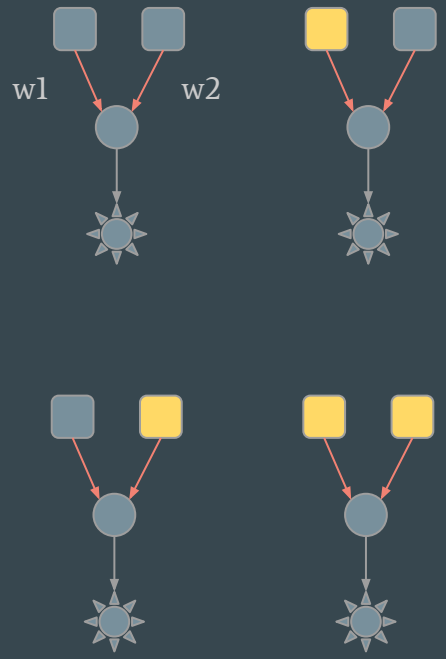

```
if ((w1*x) + (w2*y)) > threshold:
    return true
else:
    return false
```

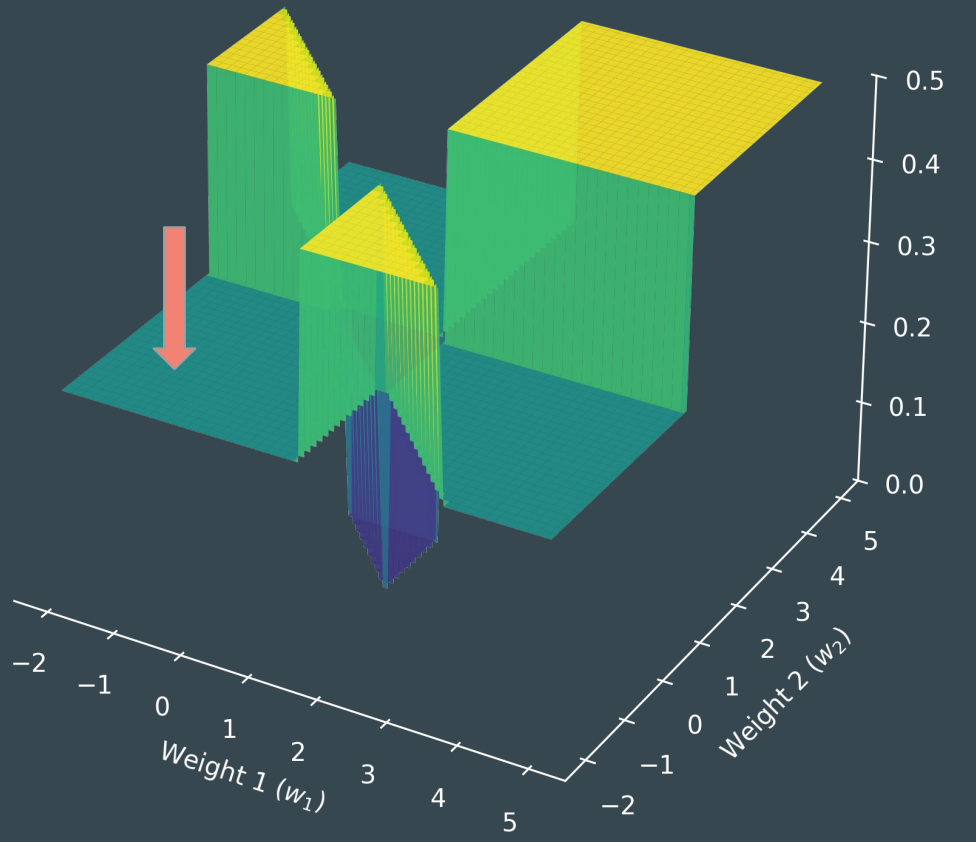
Discrete Logic



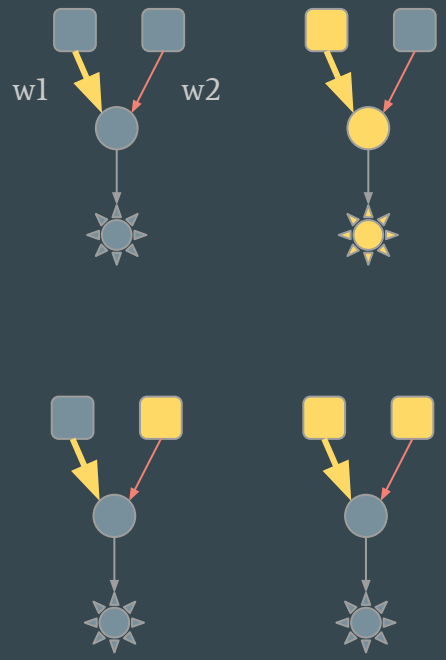


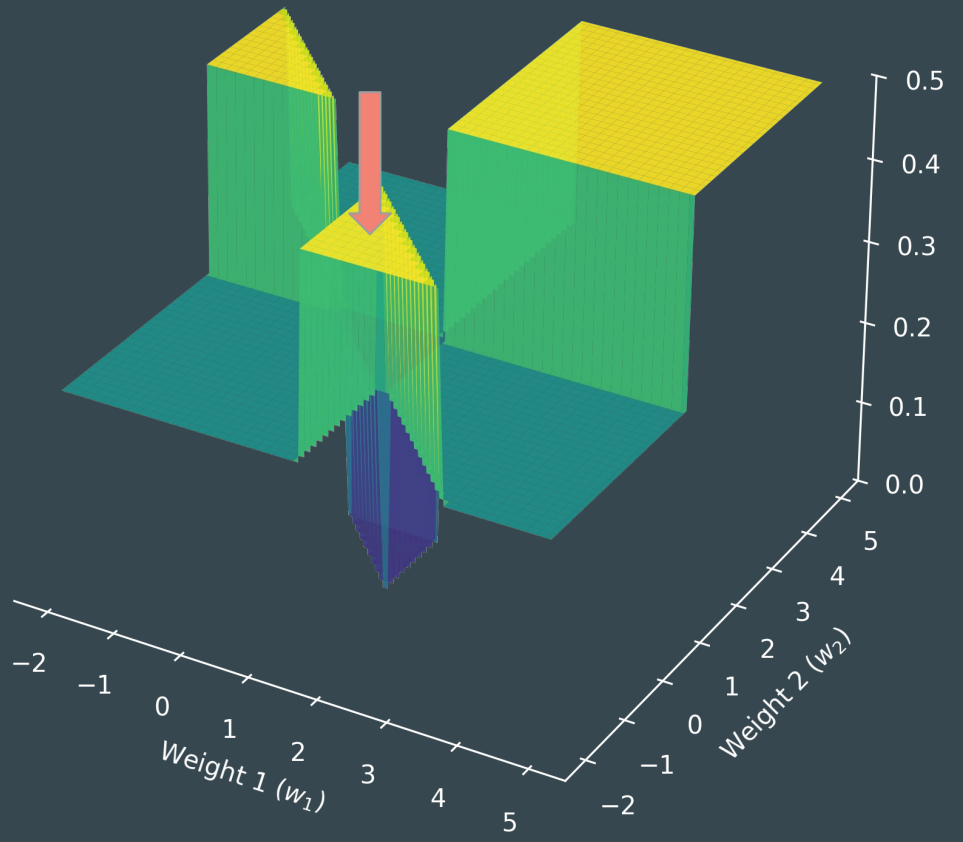
Discrete Logic



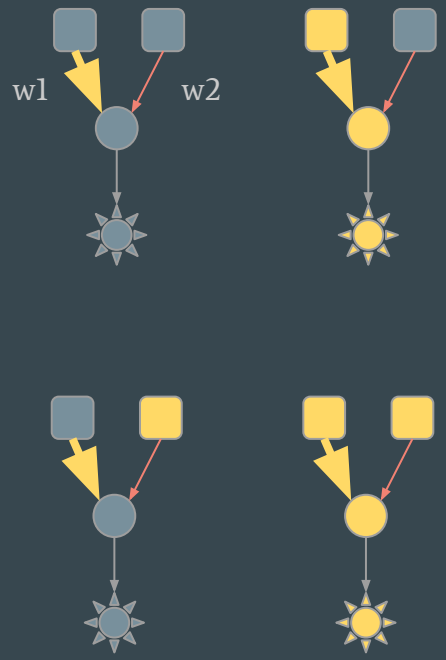


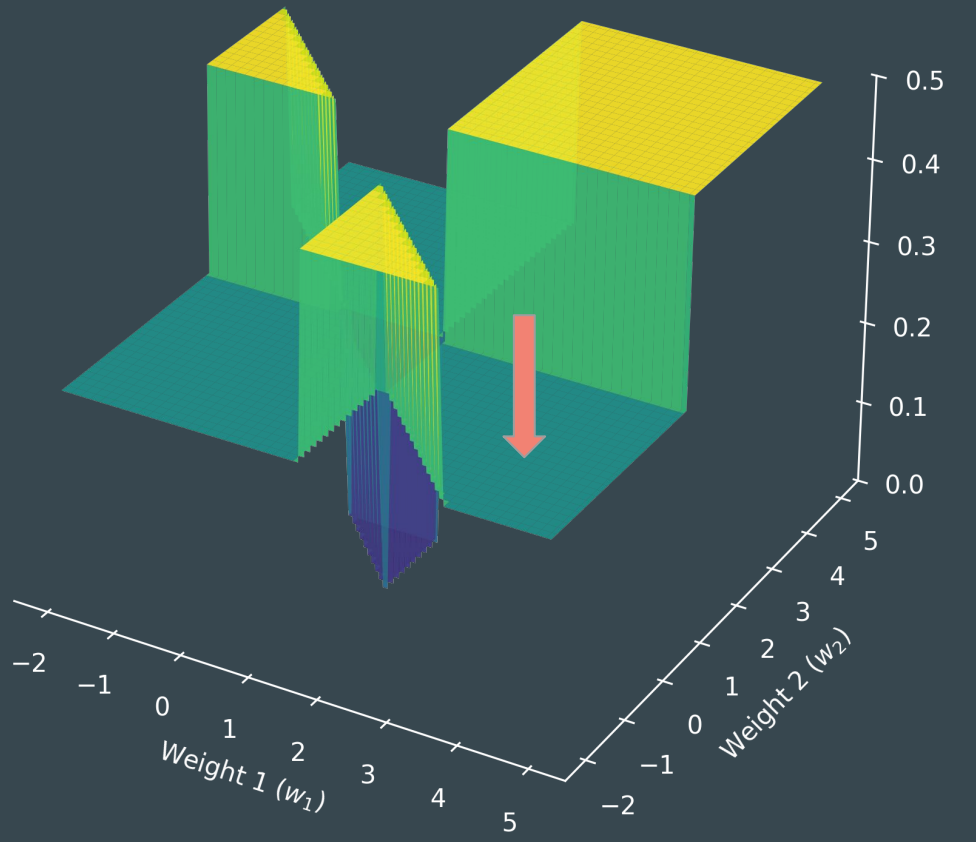
Discrete Logic



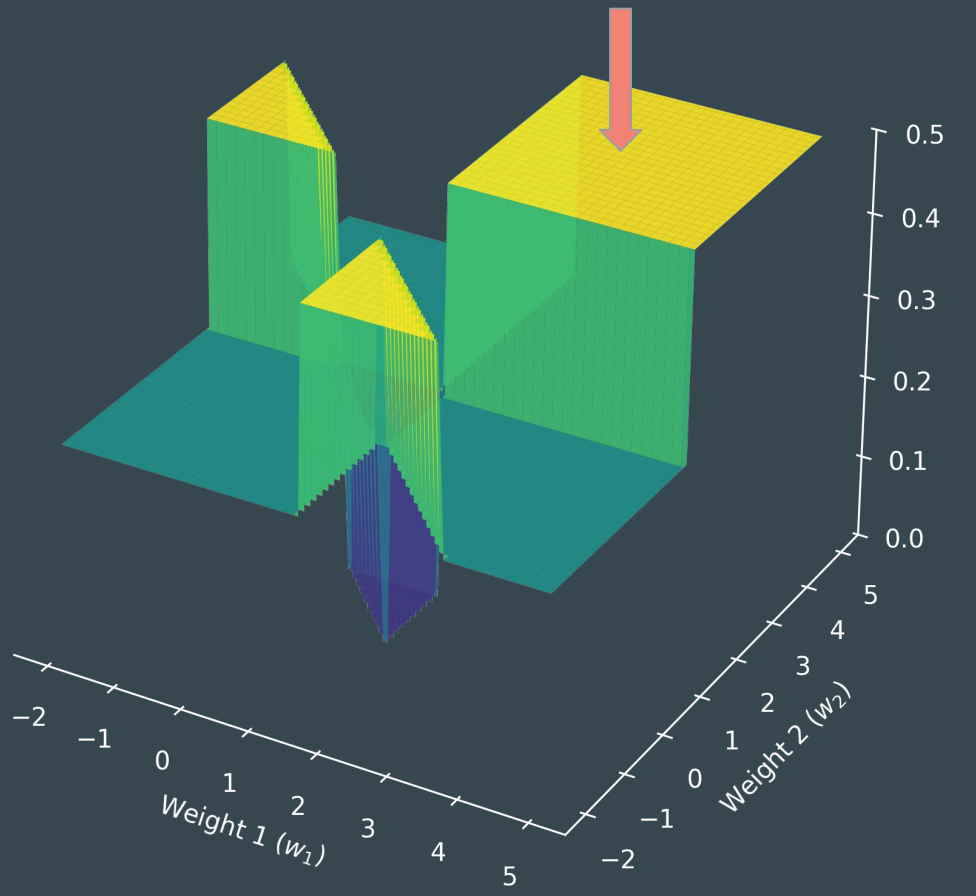


Discrete Logic

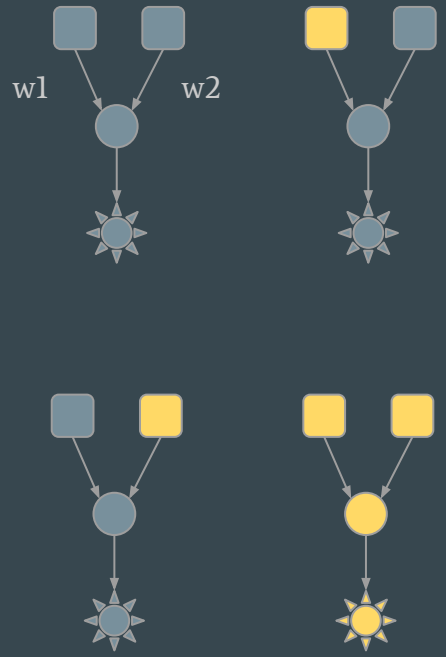


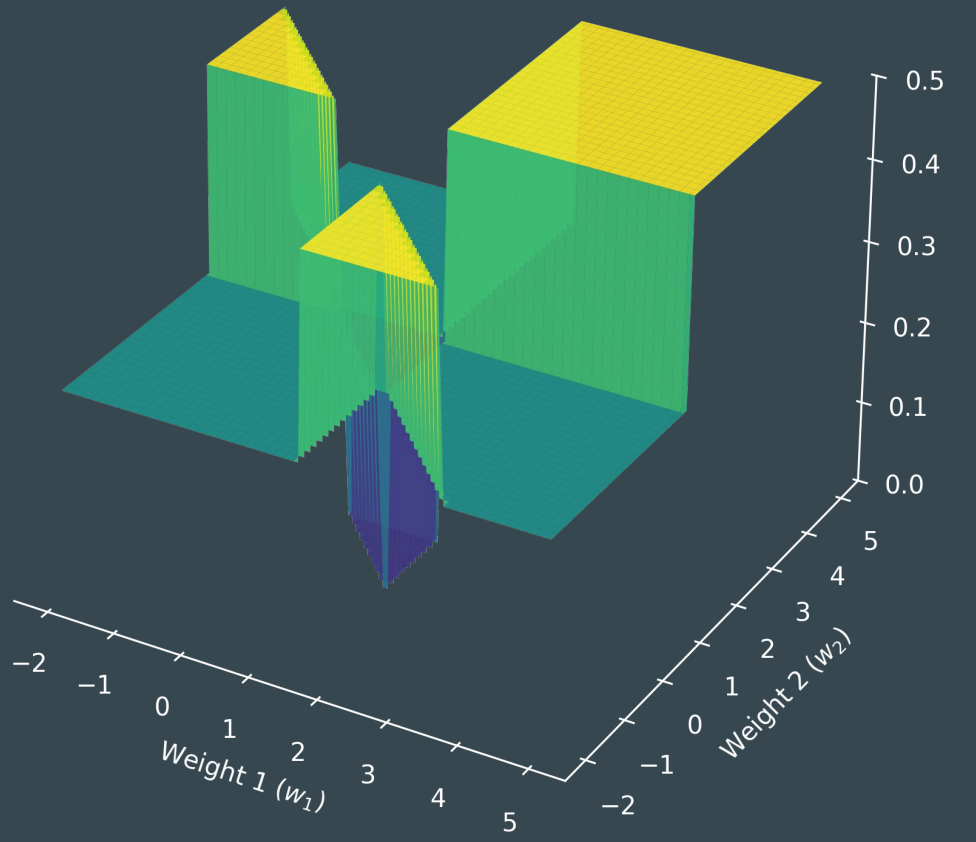


Discrete Logic

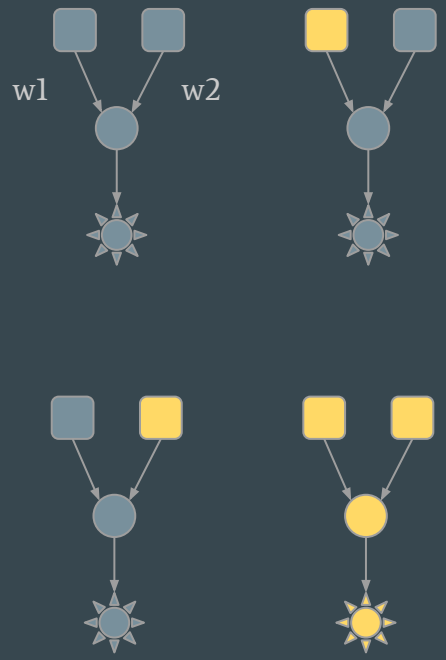


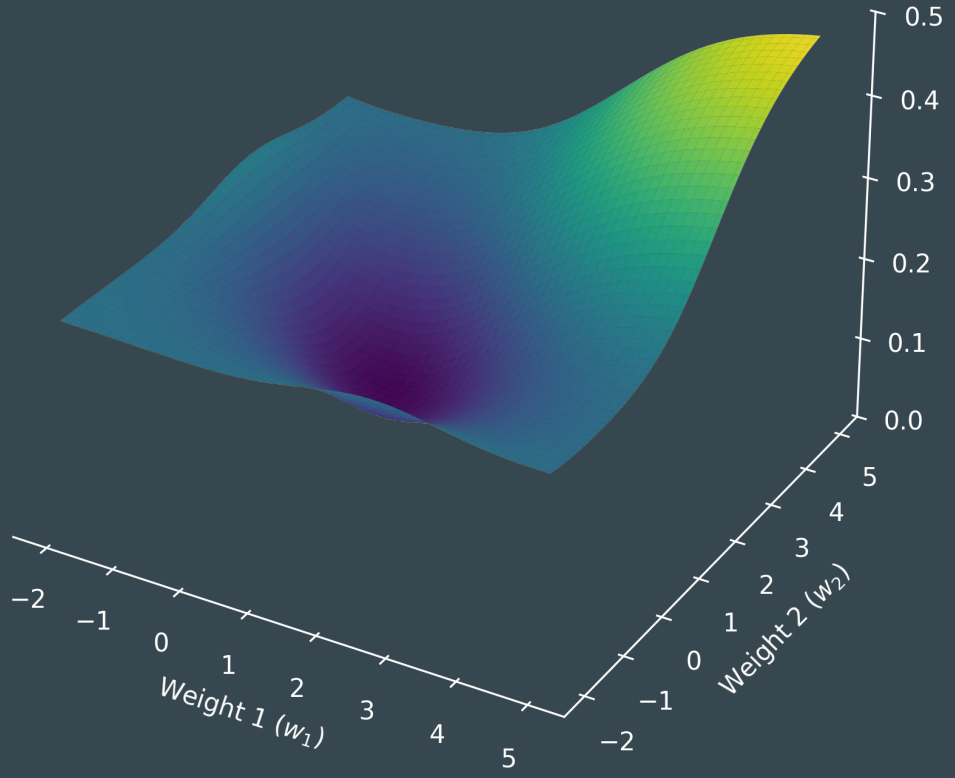
Discrete Logic



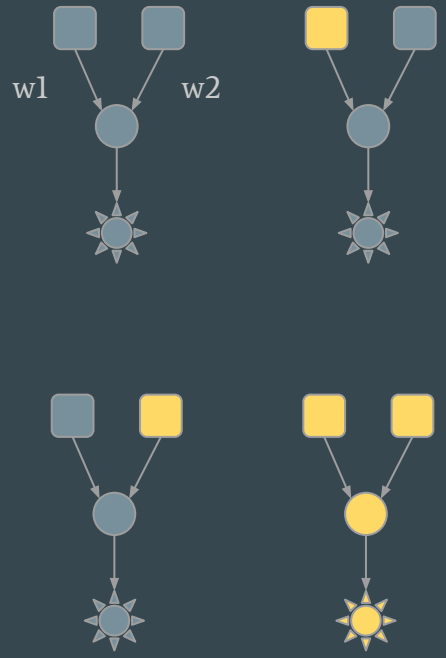


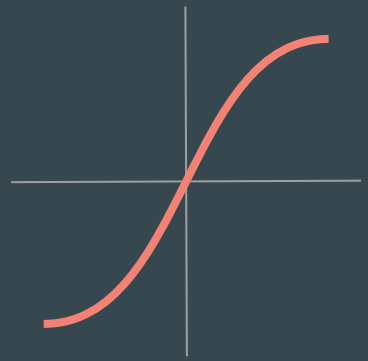
Continuous Logic





Continuous Logic





```
return sigmoid(((w1*x) + (w2*y)) + threshold)
```

Live Coding

Differentiable Programming

- Solve an optimization through **gradient descent**
- Must be a **continuous** problem

- Neural networks are just one example of differentiable programming
- Weather modeling
- Robotics
- ...

Future Work

- Vectorize the flashlight loop
- Use minimal number of flashlights
- Not all colors are equal: Use a vibrance-aware error function
- Use image recognition in error function: Focus on the important bits

Thanks

Try it for yourself on GitHub: https://github.com/sweigert/flashlights