MODULE 3: RECOMMENDATION SYSTEMS

CASE STUDY ACTIVITY TUTORIAL

CASE STUDY 3.2 – DECISION BOUNDARY AND DEEP NETWORK

CASE STUDY ACTIVITY TUTORIAL

CASE STUDY 3.2 – DECISION BOUNDARY AND DEEP NETWORK

Description:

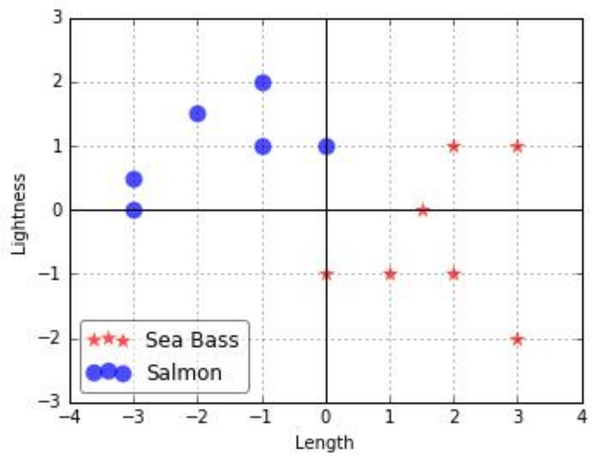
There's no getting around it. Layering perceptrons is mysterious. Why types of patterns can deep neural networks recognize? And in what ways are they more powerful than just a single perceptron? In this case study, we will explore these questions in two dimensions, so that we can visualize them easily. What happens in higher dimensions is much more complex, and an area of active research.

Part 1: When do perceptrons work, and when don't they?

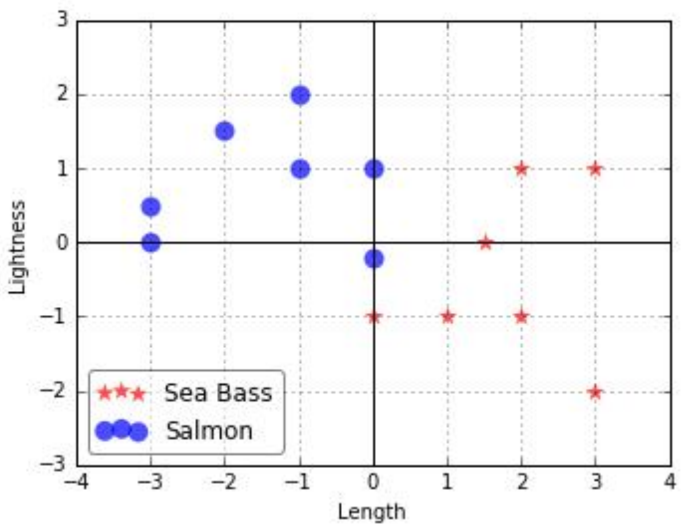
The story of fish classification is adapted from the book *Pattern Classification* by Richard O. Duda, Peter E. Hart and David G. Stork.

Suppose a fish packing plant wants to automate the process of sorting incoming fish on a conveyor belt according to species. As a pilot project, it has decided to try to separate sea bass from salmon using optical sensing. A camera is used to take pictures, and after some preprocessing, we obtain features of the fish such as length, lightness, width, number and shape of fins, and so on. After looking at some samples, we've found that it is enough to classify the two species of fish by looking only at the length and lightness.

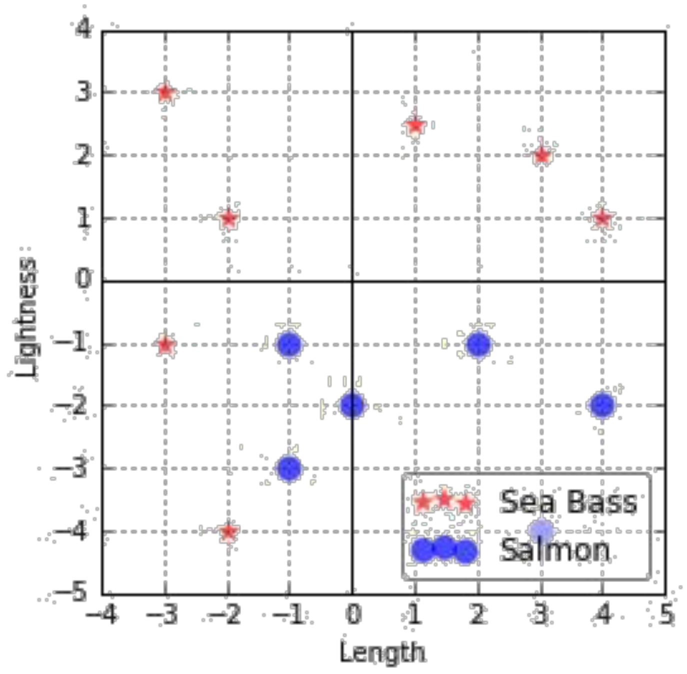
In the following, you are asked to help us to find a good fish classifier given a dataset of labeled fish samples. Specifically, for each fish, we collect its length and lightness, as a vector in $\mathbb{R}^2$. The vector is normalized by subtracting the mean length and mean lightness of fishes. We visualized the dataset in a 2D coordinate system shown below. Please find a line through the origin that separate the two types of fishes. Can you write down the weights of a perceptron whose threshold is set to zero that separate the two types of fish?



We collected another set of labeled fish, as shown below. Can we still find a line *through the origin* that separate the two types? How about a line that is not necessarily through the origin? Can you write down the weights for a perceptron that could do this? Do you need a non-zero threshold, or can you still get away with setting the threshold to zero?



Now for the following dataset, can you find a line to separate the two types of fish? Is there any perceptron that could do this? Why or why not?

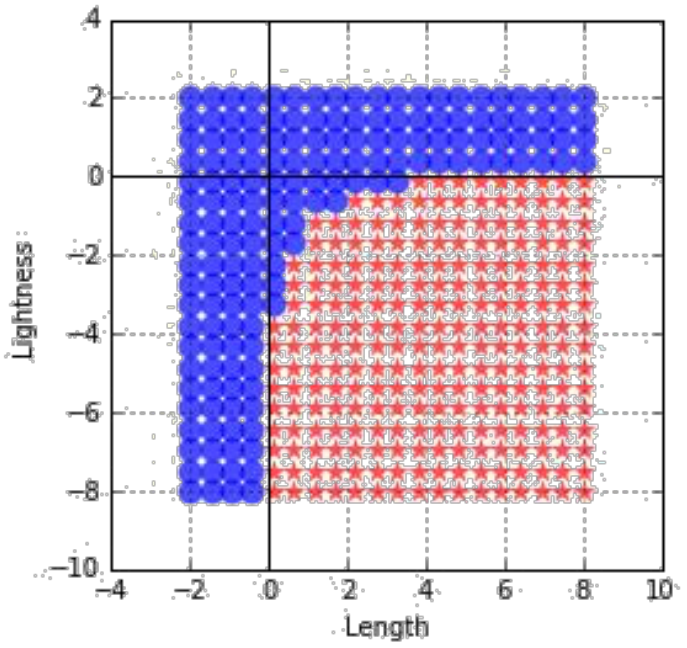


Can you come up with a different example of points in $\mathbb{R}^2$ that are labeled with +1 and -1, that no perceptron could separate?

Part 2: Some 2-Layer Neural Networks

In this part we are going to explore how adding more layers gives more expressive power to a perceptron. Namely, a multi-layer perceptron can have a much more complicated decision boundary than a single perceptron.

One simple way to decide the decision boundary of a complex function is to evaluate the function on a grid, and color each vertex according to the function value. The following figure shows an example of the decision boundary of a 2-layer perceptron visualized in this way. The *sigmoid* nonlinearity $\sigma(x)=1/(1+\exp(-x))$ non-linearity is used in this perceptron, given by $f(x)=\text{sign}(\sigma(x1)+\sigma(-x2)-1.5)$. Note how the decision boundary becomes non-linear.



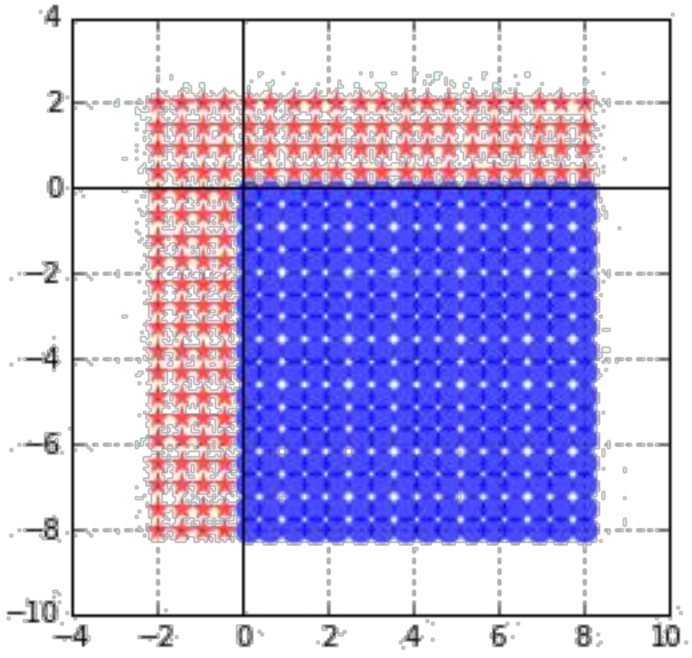
However, this simple method is quite tedious to carry out manually. Even with a computer it becomes prohibitively costly when the dimension becomes large because there are just too many points to check, and it's also very difficult to visualize what is going on.

It turns out that for a particular simple case, we can work out the decision boundary without resorting to checking over a grid of points. Can you decide the decision boundary of the following 2-layer neural network with a nonlinearity called Rectified Linear Unit (ReLU)? ReLU is defined as $\text{ReLU}(x)=\max(x,0)$. Now for $x=(x1, x2)\in\mathbb{R}^2$, consider the following network:
$f(x)=\text{sign}(\text{ReLU}(-x1)+\text{ReLU}(x2)-0.1)$

What is its decision boundary?

Hint: Notice that each of the two perceptrons only depends on one of the coordinates, length or lightness. You could try fixing one of the coordinates of a point and seeing what happens when you move along the other coordinate. When does the output f(x) change?

Answer: probably not shown to the students.



Part 3: The Power of Depth

Now that we have a handle on some types of patterns that can be recognized with a depth two neural network, and also on some types of patterns that cannot be recognized by *any* single perceptron, we can explore the benefits of depth. Let's combine what we've learned. Using the two previous parts, can you come up with an example of a set of points in $\mathbb{R}^2$ with assigned sea bass / salmon labels so that they could be separated by a 2-layer perceptron but not with a single perceptron?

Now, you may be wondering about another natural question: Are depth three neural networks more powerful than depth two? This question is much subtler than what we explored here. For starters, when we compared depth two and depth one, there were patterns that depth one just couldn't recognize. However, it turns out that *every* pattern can be recognized by a depth two network. What's the catch? Well, you might need a huge network to do it. It's conjectured that there are patterns that can be recognized with small depth three networks that cannot be recognized with small depth two networks. It certainly seems true, but no one knows how to prove it!

Appendix

This appendix is optional. If you are curious of how those figures are produced and knows a little bit Python. Here are the codes.

```
import matplotlib.pyplot as plt import numpy as np
def plot_dataset(x, y, legend_loc='lower left'): fig, ax = plt.subplots()
ax.scatter(x[y==1, 0], x[y==1, 1], c='r', s=100, alpha=0.7, marker='*', label='Sea Bass',
linewidth=0)
ax.scatter(x[y==- 1, 0], x[y==-1, 1], c='b', s=100, alpha=0.7, marker='o', label='Salmon',
linewidth=0)
ax.axhline(y=0, color='k') ax.axvline(x=0, color='k') ax.set_xlabel('Length')
ax.set_ylabel('Lightness') ax.set_aspect('equal')
if legend_loc: ax.legend(loc=legend_loc,fancybox=True).get_frame().set_alpha(0.5)
ax.grid('on')

# For the three figures in part 1
x = np.array([[2, 1], [0, -1], [1.5, 0], [0, 1], [-1, 1], [-3, 0],
[1, -1], [2, - 1], [3, -2], [3, 1], [-2, 1.5], [-3, 0.5], [-1, 2]]) y = np.array([1, 1, 1, -1, -1, -1,
1, 1, 1, 1, -1, - 1, -1]) plot_dataset(x, y)
x2 = np.vstack([x, np.array([0, -0.2])])
y2 = np.hstack([y, np.array([-1])]) plot_dataset(x2, y2)
x3 = np.array([[4, 1], [-2, 1], [ -2, - 4], [-1, -1], [2, -1], [-1, -3], [3, 2], [1, 2.5], [-3, -1], [-3, 3], [0,
-2], [4, -2], [3, -4]])
y3 = np.array([1, 1, 1, -1, - 1, -1, 1, 1, 1, 1, -1, -1, -1])
plot_dataset(x3, y3, legend_loc='lower right')

# For the sigmoid network in part 2 def sigmoid(inputs):
return 1.0 / (1.0 + np.exp(-inputs))
def nn_2layer(inputs):
return np.sign(sigmoid(inputs[:, 0]) + sigmoid(-inputs[:, 1]) - 1.5)
def plot_decision_boundary(network):
x0v, x1v = np.meshgrid(np.linspace(-2, 8, 20), np.linspace(-8, 2, 20)) x4 =
np.hstack([x0v.reshape((-1,1)), x1v.reshape((-1,1))])
y4 = network(x4)
plot_dataset(x4, y4, legend_loc=None)
plot_decision_boundary(nn_2layer)

# For the ReLU network in Part 2 def relu(inputs):
return np.maximum(0, inputs)
def nn_2layer_relu(inputs):
return np.sign(relu(-inputs[:, 0]) + relu(inputs[:, 1]) - 0.1)
plot_decision_boundary(nn_2layer_relu)
```