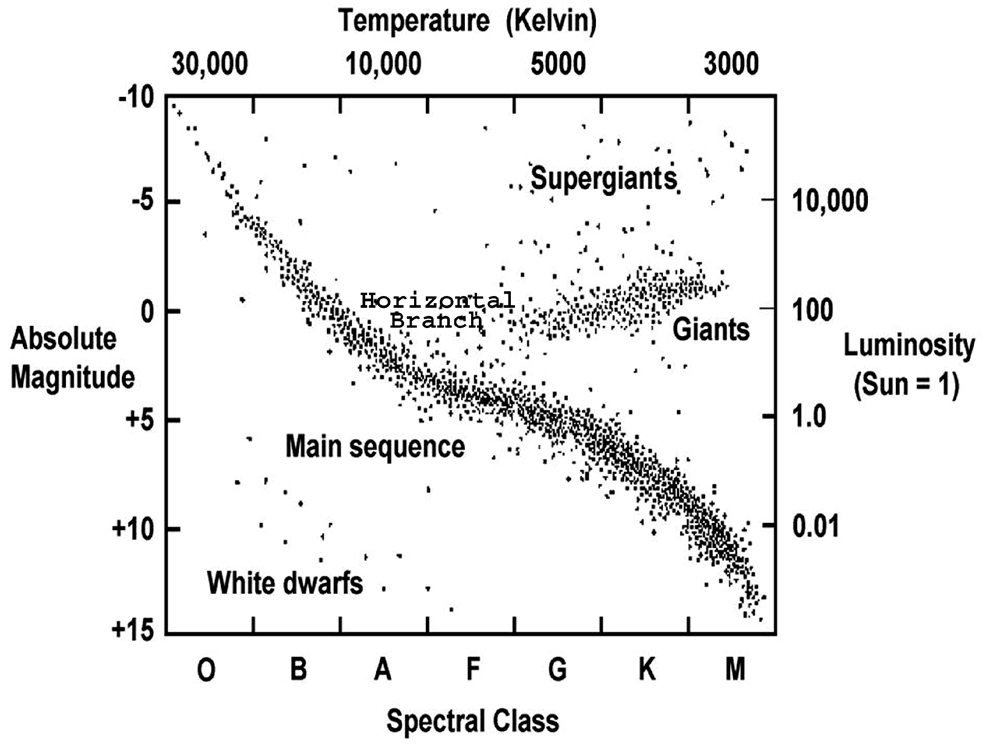
u7280249 ASTR4004 Assignment 4

Olivia Walters | Neural Networks

# Introduction

## Astrophysical Model

I’m going to create a simple astrophysical model for the classification of stars on a Herzsprung-Russel diagram:



The model will take two parameters: absolute magnitude and temperature (), and it should output the probability of it being a:

1. White Dwarf
2. Main Sequence Star
3. Giant

This should allow for a simple training set that I can pull from online. I can also use a small number of nodes.

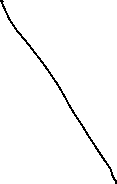
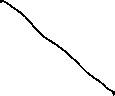
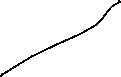
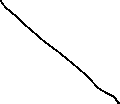
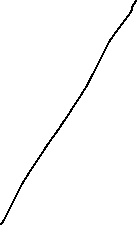
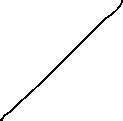
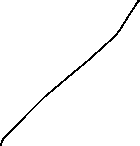
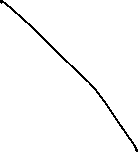
## Network Structure

The ideal network would follow what appears to look like a polynomial of degree three, with the model being able to differentiate between stars near, below, and above the main sequence central model line. So there only needs to be 3 parameters at most to have a good fit.

There will be two input neurons, three hidden neurons, and three output neurons.

Like the lecture example, the output layer will have an SoftMax activation function, and the hidden layer will use a sigmoid function.

Below is the diagram of the network structure.



## Training Data

I’m only going to include one main sequence star per spectral class, and an additional 5 white dwarfs and 6 red giants/supergiants. Along with this I’m including the Sun. This makes a total of 17 test data points per evaluation.

Both temperature and absolute magnitude will be normalized across the whole data set. Note that we use base 10 logarithmic scales because it linearises the input parameters for the neural network. For example, it’s much easier to backpropagate when you’re dealing with single digit numbers.

As an example, here’s how the input/output would work for the star Sirius B, a white dwarf:

Here’s the list of all stars used, with data gathered from Wikipedia:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **#** | **Name** | **Class** | **Type** | **Abs. Mag.** | **T eff.** | **logT** | **Y-Type** |
| 1 | The Sun | G | Main Sequence | 4.83 | 5,778 | 3.762 | 1 |
| 2 | 10 Lacertae | O | Main Sequence | −4.17 | 34,550 | 4.538 | 1 |
| 3 | Alkaid | B | Main Sequence | −0.67 | 15,540 | 4.191 | 1 |
| 4 | Sirius A | A | Main Sequence | 1.43 | 9,845 | 3.993 | 1 |
| 5 | 78 Ursae Majoris A | F | Main Sequence | 2.84 | 6,908 | 3.839 | 1 |
| 6 | 61 Ursae Majoris | G | Main Sequence | 5.53 | 5,488 | 3.738 | 1 |
| 7 | Epsilon Eridani | K | Main Sequence | 6.19 | 5,049 | 3.703 | 1 |
| 8 | Barnard's Star | M | Main Sequence | 13.21 | 3,195 | 3.504 | 1 |
| 9 | Sirius B |  | White Dwarf | 11.18 | 25,000 | 4.398 | 2 |
| 10 | Van Maanen 2 |  | White Dwarf | 14.21 | 6,130 | 3.787 | 2 |
| 11 | WD 0343+247 |  | White Dwarf | 16.8 | 4,197 | 3.623 | 2 |
| 12 | Procyon B |  | White Dwarf | 13 | 7,740 | 3.889 | 2 |
| 13 | LP 658-2 |  | White Dwarf | 15.44 | 4,270 | 3.630 | 2 |
| 14 | IK Pegasi |  | White Dwarf | 7.04 | 35,500 | 4.550 | 2 |
| 15 | Alnitak Aa | O | Blue Supergiant | -6.0 | 29,500 | 4.470 | 3 |
| 16 | Aldebaran | K | Red Giant (Branch) | -6.041 | 3,900 | 3.591 | 3 |
| 17 | Pollux | K | Red Giant (Clump) | 1.08 | 4,586 | 3.661 | 3 |
| 18 | Rho Persei | M | Red Giant (Asym.) | -1.7 | 3,479 | 3.541 | 3 |
| 19 | Arcturus | K | Giant | -0.30 | 4,286 | 3.632 | 3 |
| 20 | Antares | M | Giant | −5.28 | 3,660 | 3.563 | 3 |

# Equations of Gradient

Like what was provided in the lectures, due to the similarity of the network, the same gradient functions are ultimately used (but I will derive them in more detail to show my understanding).

We will use a cross-entropy loss function:

Note that the vector corresponds to the unit vector expected output. I’m also using a more formalised einstein notation, hence the use of superscripts/subscripts without the Σ.

This makes the backpropagated correction written as follows, for completeness:

For . The transpose as described is accounted for in lecture code.

Due to the use of SoftMax, this makes conveniently makes the pre-activation derivative of the output neurons as follows:

Using the chain rule, and , we can get the equations for the weight matrix and bias vector (using einstein summation notation):

For the hidden layer nodes, the activation function being a sigmoid make calculating the derivative easy as well:

We can use the chain rule again:

Then, with , we can find our hidden layer gradient parameters:

And thus, we have the gradient values for each of the parameters we want to adjust. The einstein notation makes it look more complicated, but it is effectively a mathematical formalisation of what was provided in the lecture code.

# Results of the Training

## Learning Rate =0.1

E

## Learning Rate >0.1

Es