Star Classification with 4-Layer Neural Networks

1. Data Collection and Preprocessing

1.1 Raw Data

Star data is contributed by *Deepraj Baidya* on Kaggle, the world's largest data science community. There are 240 observations available.

Star Features are shown here.
Temperature (K): This column consists of the Surface temperatures of several stars.
Luminosity (L/Lo): This column consists of the Luminosity of several stars calculated with respect to sun (L/Lo).
Radius (R/Ro): This column consists of the Radius of several stars calculated with respect to sun (R/Ro).
Absolute magnitude (Mv): This column consists of the Absolute Visual magnitude (Mv) of several stars.
Star type: This column is the output class (6 classes ranging from 0-5) 0 -> Brown Dwarf 1-> Red Dwarf 2 -> White Dwarf 3-> Main Sequence 4 -> Supergiant 5 -> Hypergiant.
Star color: This column contains the info about the colors of each star after Spectral Analysis.
Spectral Class: This column contains info about the spectral classes of each star (O,B,A,F,G,K,M).

1.2 Data Cleaning and One Hot Encoding

I cleaned data and did one hot encoding for star color, spectral class. Star color had 9 types (Red, Yellow, White, Orange, Blue, Yellow+White, Blue+White, Orange+Red, Yellow+Orange). Spectral class had 7 types (O,B,A,F,G,K,M). For convenience, star types were rearranged to range from 1-6 after replacing 0 with 6.

1.3 Data Scaling

I used standardization to scale 4 numerical features (K, R/Ro, L/Lo, Mv).

1.4 Sampling

I randomly drew around 80% of data as train set and the rest of data was test set.

2. Methodology of Neural Networks

2. 1 Random Initialization

Initializing all theta weights to zero does not work with neural networks. When we backpropagate, all nodes will update to the same value repeatedly. Instead we can randomly initialize our weights for our matrices by initializing each to a random value between [-1,1].

2. 2 Neural Network Construction

1. pick a network architecture;
2. choose the layout of your neural network, including how many hidden units in each layer and how many layers in total you want to have.

Number of input units = dimension of features
Number of output units = number of classes
Number of hidden units per layer = usually more the better (must balance with cost of computation as it increases with more hidden units)
Defaults: 1 hidden layer. If you have more than 1 hidden layer, then it is recommended that you have the same number of units in every hidden layer.

2. 3 Neural Network Training

1. Randomly initialize the weights;
2. Implement forward propagation;
3. Implement the cost function;
4. Implement backpropagation to compute partial derivatives;
5. Use gradient checking to confirm that your backpropagation works and disable gradient checking;
6. Use gradient descent or a built-in optimization function to minimize the cost function with the weights in theta.

Details of cost function $J(\theta)$:

$$J(\Theta) = -\frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K}[y_k^{(i)}\log((h_\Theta(x^{(i)}))_k) + (1-y_k^{(i)})\log(1-(h_\Theta(x^{(i)}))_k)] + \frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}(\Theta_{j,i}^{(l)})^2$$

1. m: sample size of train or test;
2. K: number of output units/classes;
3. s_l: number of units (not counting bias unit) in layer l;
4. λ: regularization hyperparameter.

2. 4 Neural Network Evaluation and Fine-tuning

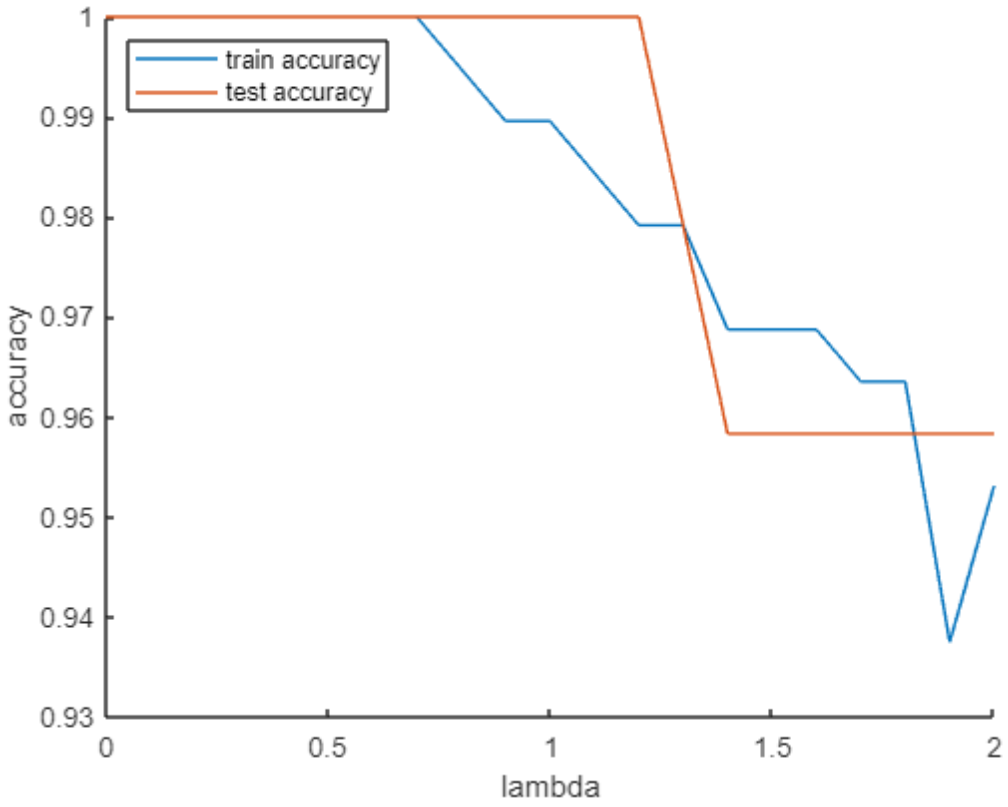
I used accuracy to measure neural network performance instead of precision or recall. Train accuracy and test accuracy are calculated. I fine-tuned the model with combinations of the number of hidden neurons in each hidden layer and regularization hyperparameter λ.

3. Empirical Results

3.1 Run 1

Settings:

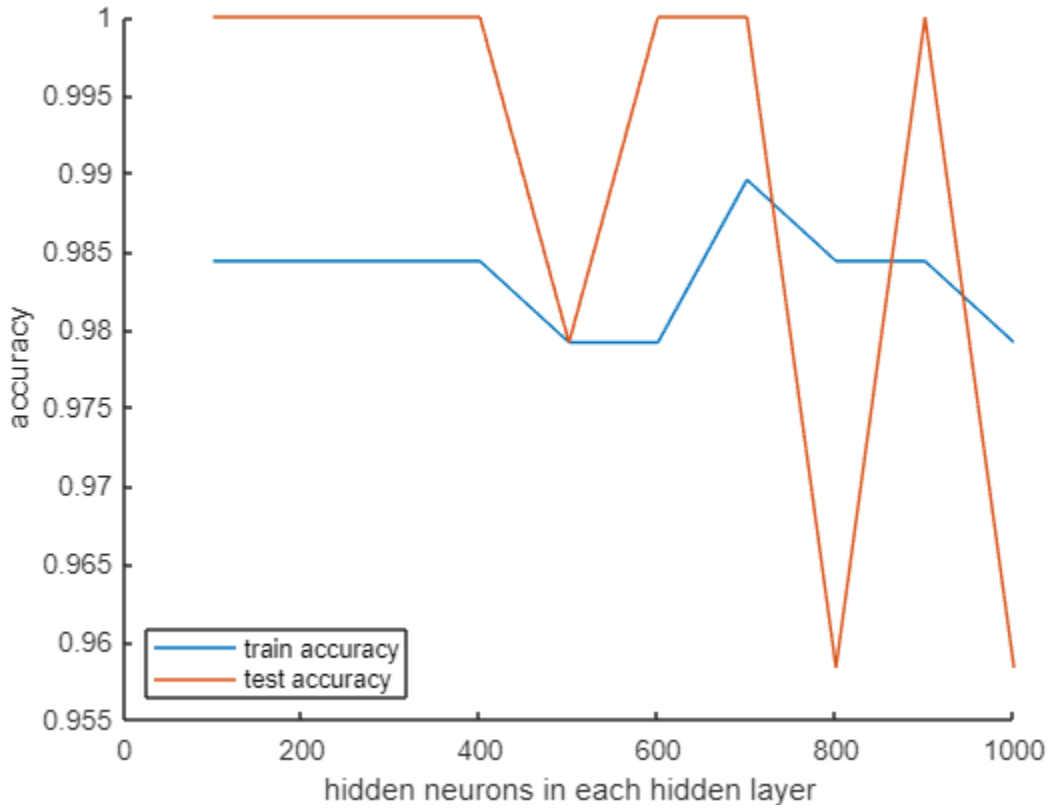
1. 4 layers of neural networks (1 input layer+2 hidden layers+1 output layer);
2. 1000 maximum iterations in gradient descent;
3. 1000 hidden neurons in each hidden layer ($s_2 = s_3 = 1000$);
4. Different values of regularization hyperparameter ($\lambda \epsilon [0,2]$).



3.2 Run 2

Settings:

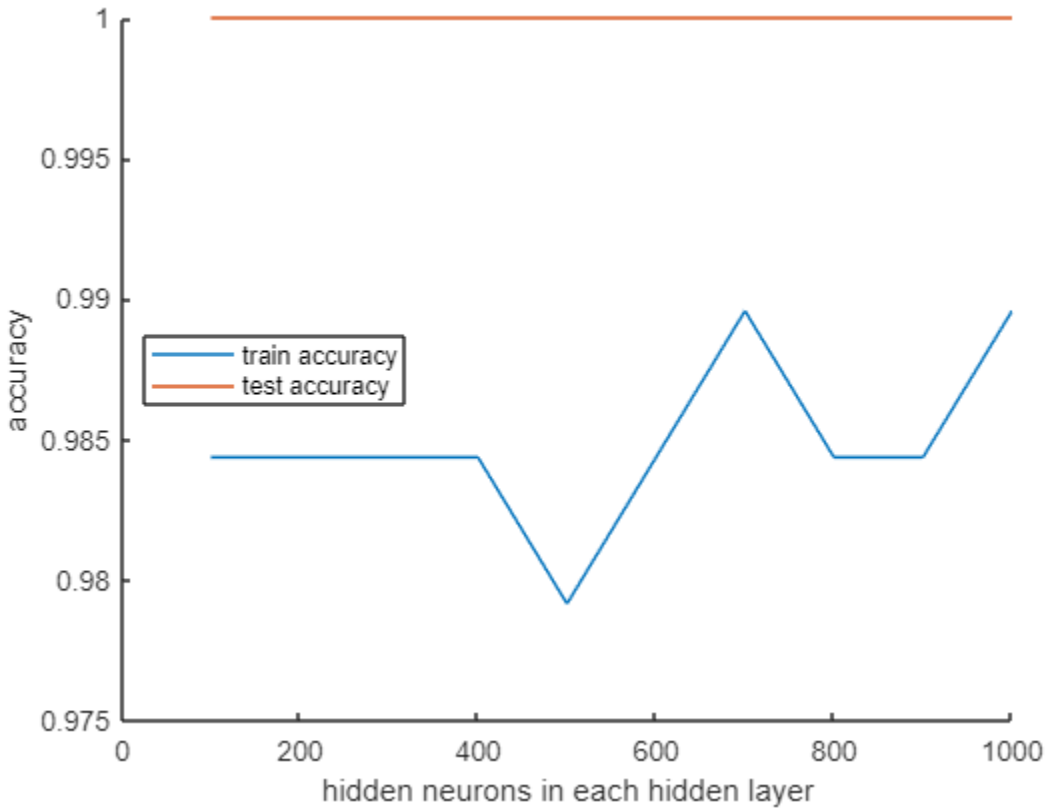
1. 4 layers of neural networks (1 input layer+2 hidden layers+1 output layer);
2. 300 maximum iterations in gradient descent;
3. $\lambda = 1$;
4. Different values of number of hidden neurons in each hidden layer ($s_2 = s_3 \ \epsilon$ [100, 1000]).



3.3 Run 3

Settings:

1. 4 layers of neural networks (1 input layer+2 hidden layers+1 output layer);
2. 500 maximum iterations in gradient descent;
3. $\lambda = 1$;
4. Different values of number of hidden neurons in each hidden layer ($s_2 = s_3 \ \epsilon$ [100, 1000]).



3.4 Run 4

Settings:

1. 4 layers of neural networks (1 input layer+2 hidden layers+1 output layer);
2. 1000 maximum iterations in gradient descent;
3. $\lambda = 1$;
4. Different values of number of hidden neurons in each hidden layer ($s_2 = s_3 \ \epsilon$ [100, 1000]).