BIS 557 Computational Statistics Final Project

Spiral Data Classification with Deep Neural Networks

MPH '21 Yijun Yang

Dec 2020

1 Introduction

Spirals are found in several natural domains, such as galaxies, shells, and double-helix DNA. Spiral structures are one of the most difficult patterns to classify because of their high levels of nonlinearity.

The two-spiral classification task for artificial neural networks was first proposed in the late 1980s by Lang and Whitbrock. Nowadays, there are several differnt packages available for this kind of classification task, but it is still important to understand the underlying logic of neural networks.

This project will be aimed at building a three-layer deep neural network and showing how to use neural networks for multi-classification. In addition, we will penalize the weights in the deep learning model and comparing the results under different choices of λ, to show how regularization solves the problem of overfitting.

2 Methods

Neural networks are an extension of the linear approaches applied to the problem of detecting non-linear interactions in high-dimensional data. Neural networks consist of a collection of objects, known as neurons, organized into an ordered set of layers. The picture below shows a 3-layer neural network.

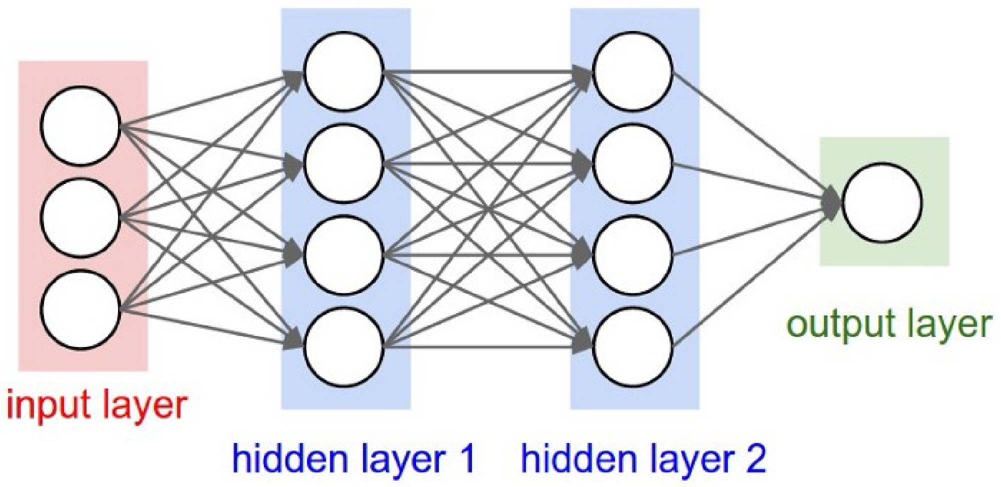

Figure1: neural network with two hidden layers

Back propagation

Training a neural network involves updating the parameters describing the connections in order to minimize some loss function over the training data. In the case of back propagation, the neural network first carries out a forward propagation, evaluates the loss of the model as per some defined cost function, adjusts the weights between the hidden layers accordingly, carries out another forward propagation and so on and so forth for some defined number of iterations.

Activation function

The use of activation functions is essential to the functioning of neural networks. Activation functions are mathematical functions that allow nueral networks to account for non-linear relationships. One of the popular choices is known as a *rectified linear unit (ReLU)*, which pushes negative values to 0 while returns positive values unmodified:

$$ReLU(x) = \begin{cases} x, & \text{if } x \geq 0 \\ 0, & \text{otherwise} \end{cases}$$

For classification tasks, we can convert a vector y of integer coded classes into a matrix Y containing indicator variables for each class. Treating the output layer as probabilities raises the concern that these values may not sum to 1 and could produce negative values or values greater than one depending on the inputs. Therefore, we need to apply a function to the output layer to make it behave as a proper set of probabilities. The activation we use is called the *softmax function*, defined as:

$$softmax(z_j^L) = \frac{e^{z_j}}{\sum_k e^{z_j}}$$

In the 3-layer neural network, *ReLU* will be used as activation function for the first 2 layers (hidden layers). *softmax* will be used for the last layer since we need to make the output layer behave as a proper set of probabilities.

Loss function

Regularization by adding a penalty term to the loss function will be employed intended to penalize the weights and address overfitting. Here, we choose to add an ℓ_2-norm as was done with ridge regression. The loss function is:

$$\mathcal{L} = -\frac{1}{n}\sum_{i=1}^{n} log\, p(y = y_i|X_i) + \frac{\lambda}{2}(\|W_1\|_2^2 + \|W_2\|_2^2 + \|W_3\|_2^2)$$

Application

To prove the effectivness of our model, a simulated spiral data will be used for classification. The input $X \in \mathbb{R}^{N\times 2}$ will be 2-dimension. The response will be $y \in \mathbb{R}^N$, which is the category indicator. The following figure intuitively show the data points and corresponding categories.

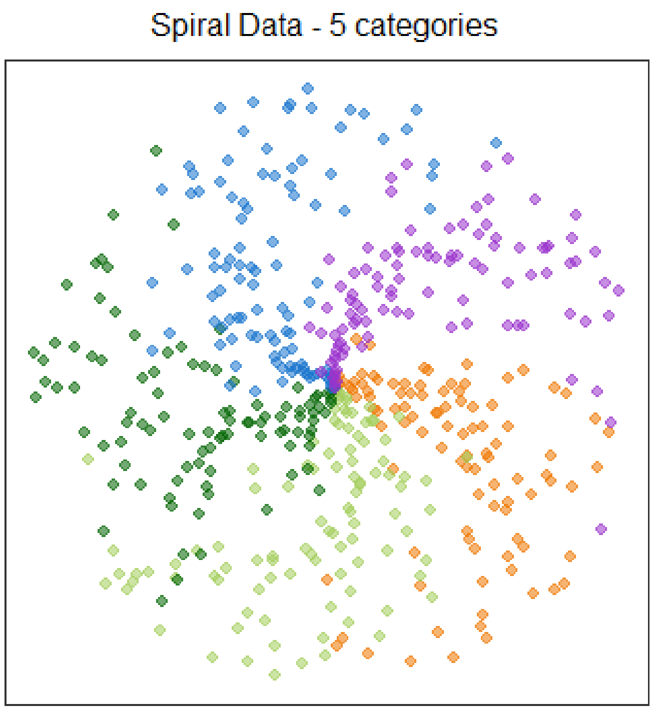

Figure2: simulated spiral data

3 Results

Math Derivation

From the above description, we construct a neural network model as below:

$$h_1 = ReLU(W_1 X + b_1)$$
$$h_2 = ReLU(W_2 h_1 + b_2)$$
$$p = softmax(W_3 h_2 + b_3)$$

where h_1 represents the results of the first hidden layer; h_2 represents the results of the second hidden layer; p represents the results of the output layer, which is a matrix of probability. W_1, W_2, W_3 and b_1, b_2, b_3 are the weights and biases for the corresponding layer.

Then, we need to get the formulas for the derivatives for each layer of the network:

$$\frac{\partial \mathcal{L}}{\partial W_3} = \frac{1}{N} h_2^T (P - Y) + \lambda W_3$$

$$\frac{\partial \mathcal{L}}{\partial b_3} = \begin{pmatrix} \frac{1}{N} \\ \vdots \\ \frac{1}{N} \end{pmatrix} (P - Y)$$

$$\frac{\partial \mathcal{L}}{\partial W_2} = \frac{1}{N} h_1^T (P - Y) W_3^T \, \mathbf{1}\{h_2 > 0\} + \lambda W_2$$

$$\frac{\partial \mathcal{L}}{\partial b_2} = \begin{pmatrix} \frac{1}{N} \\ \vdots \\ \frac{1}{N} \end{pmatrix} (P - Y) W_3^T \, \mathbf{1}\{h_2 > 0\}$$

$$\frac{\partial \mathcal{L}}{\partial W_1} = \frac{1}{N} X^T (P - Y) W_3^T \, \mathbf{1}\{h_2 > 0\} W_2^T \, \mathbf{1}\{h_1 > 0\} + \lambda W_1$$

$$\frac{\partial \mathcal{L}}{\partial b_1} = \begin{pmatrix} \frac{1}{N} \\ \vdots \\ \frac{1}{N} \end{pmatrix} (P - Y) W_3^T \, \mathbf{1}\{h_2 > 0\} W_2^T \, \mathbf{1}\{h_1 > 0\}$$

Coding Implementation

Coding are implemented with R version 4.0.3. All the relavent resources are introduced in Appendix. Parameters are fine-tuned based on the loss and convergence, and remain fixed. In each hidden layer, there are 100 nodes; the maximum iteration numbers are 2,000; step size is 0.01.

The following figure intuitively shows how the classifier works under different λ.

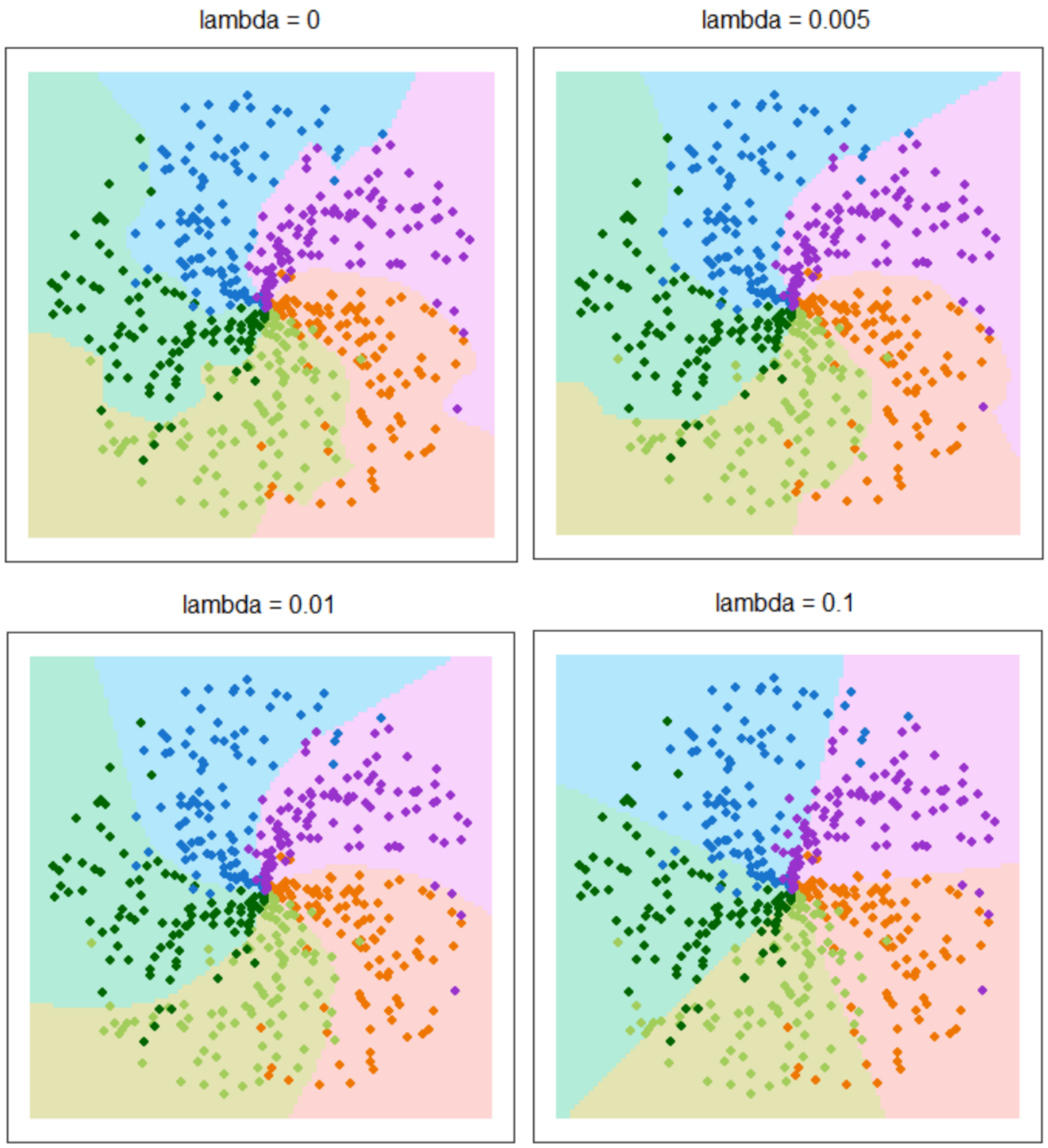

Figure3: classification boundary under different λ

lambda	accuracy
0.000	0.924
0.005	0.900
0.010	0.880
0.100	0.744

Table1: classification accuracy under different λ

4 Discussion

Overfitting is a modeling error that occurs when a function is too closely fit to a limited set of data points. It captures the noise in the data set, and may not fit new incoming data. A neural network with large weights can be a sign of an unstable network. Large weights tend to cause sharp transitions in the node functions and thus large changes in output for small changes in the inputs.

To overcome this issue, one of the most common methods is regularization. Regularization is a technique used for tuning the function by adding an additional penalty term in the error function. In this case, we add an ℓ_2-norm (calculating the sum of the squared values of the weights) as was done with ridge regression to penalize weights.

According to the above results, it is obvious that when $\lambda = 0$ (no penalization), although the classification accuracy is high, the decision boundary is so curvy and causes an overfitting problem. When $\lambda = 0.01$, the decision boundary seems to be more reasonable. However, When λ is too large, for example, $\lambda = 0.1$, the decision boundary tends to be like that of a linear classifier, which results in low accuracy.

In the furture research, more improvements can be done. For example, instead of only using the ℓ_2-norm, elastic net that uses both ℓ_1 and ℓ_2 penalties can be a useful approach to try.

Reference

[1] Arnold, T., Kane, M., & Lewis, B. W. (2019). A computational approach to statistical learning. CRC Press.

[2] A Neural Network Playground

[3] Build your own neural network classifier in R

[4] Coding Neural Network — Forward Propagation and Backpropagtion

[5] Explain the softmax function and related derivation process in detail

[6] Phyllotaxis - Draw flowers using mathematics

[7] Stephan K. Chalup & Lukasz Wiklendt (2007) Variations of the two-spiral task, Connection Science, 19:2, 183-199

[8] Understanding Regularization in Logistic Regression

[9] Use Weight Regularization to Reduce Overfitting of Deep Learning Models

Appendix

All the code used for this project are stored at GitHub: yijunyang/bis557 repository.

- Code for spiral data simulation: `project/spiral_simulation.R`
- Code for functions created for this project: `R/...`
    - `relu.R`: the ReLU function
    - `relu_p.R`: derivative of ReLU function
    - `softmax.R`: the softmax function
    - `make_weights.R`: initialize weights and biases for neural networks
    - `forward_propagation.R`: implement forward propagation
    - `neural_networks.R`: implement back propagation
    - `neural_networks_pred.R`: prediction of the neural network classifer
- Original `.rmd` file for this project: `project/final_report.rmd`