Draft version November 10, 2019
Preprint typeset using LaTeX style emulateapj v. 12/16/11

CLASSIFICATION AND REGRESSION WITH NEURAL NETWORKS

Sigbjørn Foss

Draft version November 10, 2019

ABSTRACT

A data set of cancer cell samples is analysed using logistic regression and a neural net classifier. The neural net is also applied to a generated regression data set. The neural network is found to perform slightly worse at these tasks than the traditional methods, but the algorithm is more flexible when it comes to adapting it to other problems. Github repository with all source files can be found at this link.

1. INTRODUCTION

Machine learning algorithms have become ubiquitous across almost all aspects of modern society. Neural networks in particular are used in a wide range applications, and their importance cannot be understated. While the first neural network models as we recognize them today were proposed as early as in the 1960s [1], refinements and improvements made during the last two decades or so have made them into extremely powerful tools for data analysis. This paper applies a neural networks to solve a binary classification problem and a regression problem and compares the results with traditional solutions, logistic regression for classification and least squares regression for regression. The classification problem to be solved is a set of cell nucleus samples collected at the University of Wisconsin, which are labeled according to whether the samples were proven to be benign or malignant. The regression set is a generated sample of the Franke function with normally distributed noise. The methods section includes some background for logistic regression with gradient descent, and a more thorough explanation for the neural network algorithm with stochastic gradient descent. The results of the implemented algorithms are presented in the results section and further discussed in the conclusion.

2. METHODS

2.1. *Logistic Regression with gradient descent*

Logistic regression, despite its name, is a binary classification method. Given a feature matrix $\boldsymbol{X} \in \mathbb{R}^{n\times p}$, where p is the number of features and n is the number of samples, and a set of labels $\boldsymbol{y} \in \mathbb{R}^n$, the probability of sample i being in the target class, i.e., $y_i = 1$ is given by the sigmoid function,

$$\boldsymbol{p}(y_k = 1) = \frac{\exp(\boldsymbol{X}_{k,*}\boldsymbol{\beta})}{1 + \exp(\boldsymbol{X}_{k,*}\boldsymbol{\beta})},$$

and conversely,

$$\boldsymbol{p}(y_k = 0) = 1 - \boldsymbol{p}(y_k = 1).$$

The aim of logistic regression is to find a set of coefficients $\boldsymbol{\beta} \in \mathbb{R}^p$ such that the probabilities given by the sigmoid function with the features input produces a probability which aligns with the label for the samples. This is done by minimizing the cost function

$$C(\boldsymbol{\beta}) = \sum_i^n \left[y_i \boldsymbol{X}_{i,*}\boldsymbol{\beta} - \ln(1 + \exp(\boldsymbol{X}_{i,*}\boldsymbol{\beta}))\right],$$

which is derived using the maximum likelihood estimate principle [2]. Because the cost function is convex, it can be minimized by gradient descent. First, some random values for $\boldsymbol{b}$ is set. The gradient of the cost function with respect to $\boldsymbol{\beta}$, $\nabla_\beta C$ is calculated. This gives the direction of the steepest descent for the cost function. To bring the cost function closer to its minimum, the gradient is scaled by a the learning rate η and subtracted from $\boldsymbol{\beta}$,

$$\boldsymbol{\beta}_{i+1} = \boldsymbol{\beta}_i - \eta\ \nabla_{\beta_i} C(\boldsymbol{\beta}_i).$$

This step is repeated until $||\nabla_{\beta_i} C(\boldsymbol{\beta}_i)|| <$ tol for some tolerance. The gradient of the cost function is

$$\nabla_{\beta_i} C(\boldsymbol{\beta}_i) = -\boldsymbol{X}^T(\boldsymbol{y} - \boldsymbol{p})\ [2].$$

2.2. *Neural Network Architecture*

Neural nets are structures that can be used for either classification or regression. They are in essence complex functions with often thousands of parameters, and the challenge is to set these parameters. Figure 1 shows a

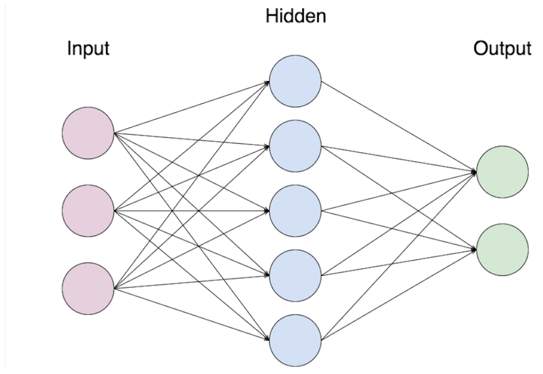

Fig. 1.— Neural network with 3 input nodes, 5 hidden nodes and 2 output nodes.

model of simple neural network. The nodes depicted can be thought of as just holding a value, and are denoted $\boldsymbol{a}^l$, where l is the layer, and the vector holds all the values in the nodes. The first (leftmost) layer contains the input values, i.e. one instance of the feature matrix from the previous section ,

$$\boldsymbol{a}^0 = \boldsymbol{X}^T_{0,*}.$$

The layers are connected by a set of weights, connecting each node of the previous layer to each node of the current layer, so if the previous layer contains n_{l-1} nodes and the current layer contains n_l nodes, the weights form a matrix $\boldsymbol{w}^l \in \mathbb{R}^{n_{l-1}\times n_l}$. For every layer after the input layer, a bias value is added in each node, $\boldsymbol{b}^l \in \mathbb{R}^{n_l}$. For each forward step in the network, the activations are fed through an activation function. If the inputs were fed through linearly, the network could not work as anything more than a linear prediction algorithm. The typical example here is the sigmoid function

$$\sigma(x) = \frac{1}{1-e^{-x}},$$

which will be used in the following. Explicitly, the feedforward algorithm consists of

1. setting the input layer, $\boldsymbol{a}^0 = \boldsymbol{X}^T_{k,*}$,
2. feeding the input through the network by $\boldsymbol{a}_l = \sigma(\boldsymbol{a}_{l-1}\boldsymbol{w}_l + \boldsymbol{b}_l)$.

The last activation $\boldsymbol{a}_L$ is then the output of the network. In practice, these operations work when inputting the whole feature matrix. The the activations $\boldsymbol{z}_l$ and $\boldsymbol{a}_l$ are then matrices with second dimension equal to numbers of samples in the data set.

2.3. *Training the network*

Similarly to regular classification and regression techniques, neural network parameters are tuned by minimization of a cost function $C(\boldsymbol{a}_L, \boldsymbol{y})$, where $\boldsymbol{y}$ are the targets. The standard minimization method is batch stochastic gradient descent. It works in essentially the same way as regular gradient descent, but because there are now potentially thousands of weights and biases, the gradient is approximated by using only a subset of the data set in each iteration. For each iteration a subset $\boldsymbol{X}_b \in \mathbb{R}^{n_b \times p}$ is fed through the algorithm, and the weights and biases are updated in a procedure which will be explained shortly. After the network has seen all the training examples, it has competed an 'epoch' of training. To update the weights and biases, the so-called back-propagation algorithm is used. It is derived in Hjort-Jensen(2019) [1], invoking the chain rule. The output error is found by

$$\delta_L = \nabla_{a^L} C \odot \sigma'(\boldsymbol{z}^L),$$

where $\odot$ denotes the Hadamard product, i.e. component-wise multiplication. Note the that $\sigma(x)$ here denotes the activation function used in the last layer. The errors in the other layers are given by

$$\delta_l = \left[(\boldsymbol{w}^L)^T \delta^{l+1}\right] \odot \sigma'(\boldsymbol{z}^l).$$

The cost function partial derivatives w.r.t the weights and biases are then given by

$$\nabla_{w_l} C = (\boldsymbol{a}^{l-1})^T \delta^l$$
$$\nabla_{b_l} C = \delta^l,$$

and the weights and biases are updated by

$$\boldsymbol{w}^l_{i+1} = \boldsymbol{w}^l_i - \eta \nabla_{w_l} C$$
$$\boldsymbol{b}^l_{i+1} = \boldsymbol{b}^l_i - \eta \nabla_{b_l} C.$$

To prevent overfitting, a regularization term is often added to the cost function. l_2-regularization is most common, adding $\lambda||\boldsymbol{w}^l||^2$ to the cost function. Back-propagating this term through the network leads to the weights being updated by

$$\boldsymbol{w}^l_{i+1} = \boldsymbol{w}^l_i - \eta(\nabla_{w_l} C + \lambda \boldsymbol{w}^l_i).$$

3. RESULTS

The results referenced here can be found in the Ipython notebook file 'main.ipynb' in the repository.

3.1. *Analysing cancer data using logistic regression*

The data set used to test the classification methods is a set of measurements done on cancer cell nuclei at the University of Wisconsin. The data set contains 569 samples with 30 features and each sample is labeled with 1 or 0 according to whether it was found to be benign or malignant respectively. The data set was sectioned into a training and testing set. The logistic regression algorithm was set up as described in section 2.1, with a learning rate of 10^{-7}. Higher learning rates were tested, but the cost function gradient (CFG) failed to converge at a sufficiently low values. The tolerance for the CFG was set at 0.1, but the algorithm was terminated after 10^7 iterations, and so none of the tests reached this tolerance. At this tolerance, the script ran for upwards of 8 minutes. While this data set is quite small, some preprocessing was applied to test their effect. The classification was first performed on the raw data, and then with preprocessing. The preprocessing consisted of scaling the data by its standard deviation and centering, and principal component analysis (PCA). PCA is a dimensionality reduction technique, where the feature data is projected onto an orthogonal subspace that retains most of the variance in the data. This has the effect of reduction the amount of features, while still allowing for accurate prediction. Without any preprocessing, the accuracy for logistic regression on the test set was 96 %. After preprocessing the accuracy was 94% but the algorithm was 10 % faster.

3.2. *Repeating the analysis with a neural network*

The neural network was implemented with a log likelihood cost function and the softmax activation function for the last layer. The log likelihood used here is analogous to the cost function in logistic regression, but allows for more classes in the prediction. To produce outputs that can be interpreted as probabilities the softmax function is used as the output layer activation function,

$$a^L_i = \text{Softmax}(z^L_i) = \frac{e^{z^L_i}}{\sum_j^{n_L} e^{z^L_j}}$$

which results in the property that $\sum_j^{n_L} a^L_i = 1$. In principle, the classification can be performed with a single output neuron using sigmoid activation, but because the softmax method works as well in binary classification and can be used for multiclass classification as well, its was chosen for better flexibility. To use softmax for binary classification, the network needs two output neurons. This means that the labels in the training set had to be transformed using one hot encoding. This entails mapping the test labels onto one-hot vectors, i.e.

$$1 \rightarrow \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \; 0 \rightarrow \begin{pmatrix} 0 \\ 1 \end{pmatrix}.$$

Most of the hyperparameters were roughly tuned by hand. This includes the number of layers, number of neurons in the hidden layers, batch sizes, number of training epochs and activation function parameter. Because of the small number of samples, networks with more than one hidden layer were distinctly ineffective. The network used for classification was then initialized with 20 hidden neurons, as numbers above this did not yield significantly better results. The batch size was set to 10, as the network tended to encounter overflow errors with higher batch sizes, and was trained for 50 epochs, again because longer training periods did not yield better results. The network was first tested on the Wisconsin cancer data using sigmoid activation functions in the hidden layers. This resulted in the network predicting all the samples as malignant in nearly every attempt using a wide range of hyperparameters. The network was modified to use exponential linear units ,

$$\mathrm{ELU} = \begin{cases} x & , x \geq 0 \\ \alpha(e^x - 1) & , x < 0 \end{cases}.$$

The ELU activation function does not flatten for high activations, ensuring a higher learning rate. The network using ELU activations still suffered the same problem, but at a much less often. To find the best learning rate and regularization parameter a grid search was performed over $\lambda, \eta \in [10^{-5}, 10^{-4}, .., 10^{1}]$. Again the network only made predictions that weren't all zeros for $\eta > 10^{-4}$, so the grid search was refined to only search for learning rates between 10^{-5} and 10^{-4}. After all these considerations, using $\eta = 5.05 \cdot 10^{-5}$ and $\lambda = 0.01$ resulted in a classification accuracy of 92 % on the test set.

3.3. *Neural network regression*

The regression data set is the Franke function with normally distributed noise used in project 1. The data set consists of 1875 samples with 36 features, with labels $\boldsymbol{y} = f\boldsymbol{X}$. Re-purposing the neural network for classification is just a matter of swapping the cost function. The cost is now the mean squared error

$$C(\boldsymbol{a}^L, \boldsymbol{y}) = \frac{1}{2n} \sum (a_i^L - y_i)^2,$$

and the activation in the last layer is the sigmoid function. For each sample there is one label, the function output, so the network is initialized with one output node. Using ELU activation functions for the hidden layer, a batch size of 20, and 200 epochs, a grid search is performed over the regularization parameters and learning rates. For high learning rates and regularization parameters, the network tends to encounter divide by zero errors. For the rest of the values, the predictions are overall quite good and for $\eta = 0.001, \lambda = 10^{-5}$ the network achieves a MSE of 0.017, and $R^2 = 0.81$ on the test set. The standard ridge regression achieves somewhat better predictions with an MSE of 0.013 and an R^2 of 0.87.

4. CONCLUSION

For these two tasks the neural net did not outperform the traditional methods of classification and regression. There are however many possible ways to improve the network that could most likely improve performance. If grid search was performed over more of the parameters, we would have a much better chance of finding parameters that would increase success rate a few percentage points. Additionally the data sets used were quite small, and neural nets are known to be very data hungry, so it is really no great surprise that sample sizes of 569 and 1875 are not sufficient to truly demonstrate the potential of the algorithm. What we do get to showcase is the flexibility of the method, and herein lies the great strength of neural nets. With only minor modifications, it can be tailored to the application of a wide range of problems. Most of the shortcomings of the results in this report stem from a failure to properly explore the many possibilities for enhancing the neural network algorithm. This is the result of me spending an inordinate amount of time grasping the basic concepts of neural networks, a subject that I had great difficulties with. Most of the time was spent just making the basic network structure work, so I ended up not really having time to develop the algorithm beyond its basic functionality. Because of this I also chose to work on a very basic data set that did not need a great amount of preprocessing. However, as previously stated, other data sets with more samples would most likely be better suited for analysis with neural nets.

5. REFERENCES

1. Michael A. Nielsen (2015). *Neural Networks and Deep Learning*, Determination Press.

2. Morten Hjort-Jensen (2019). *Data Analysis and Machine Learning: Logistic Regression*