README for Artificial Neural Network (ANN)

Group 2

02 12 2019

Corresponding R Script: ANN

1. Introduction

The goal of this project is to make a prediction about the likelilhood of College Football (CFB) players to be drafted into the professional football league (NFL). In order to achieve this, different aspects of a CFB player's college career are being used as features $p_1, ..., p_n$. These include game statistics such as rush attempts `Rush.Att` or yards ran after a pass `Pass.Yard`. To solve this problem, one of the methods employed is an artificial neural network (ANN). ANNs consist of connected nodes and are not too dissimilar to a human brain in that they derive their "intelligence" from the structure of their "neurons" and the strength of the "synapses" (edges) between these nodes. For this project, an ANN with one input layer with the size of the number of features in the underlying dataset, one hidden layer with ten nodes and an output layer of the size of one was used to solve the CFB player classification problem. The main advantages of ANN include its ability to model complex, non-linear relationships between input and output variables, its ability to infer generalized relationships on data it has never seen before, its robustness towards local minima in the cost function as well as its universality property, meaning it can fit any problem, provided it is sufficiently large.

2. ANN Setup and Backpropagation

The feedforward ANN used in this project is set up with three layers: an input layer of the size of the number of features in the data, one hidden layer consisting of ten nodes and an output layer containing a single node. Having one hidden layer was deemed to be sufficient due to the complexity of the underlying problem and the rule of thumb commonly used to determine the size of the hidden layer is to take the mean between the number of nodes in the input layer and the output layer. The output layer was constructed to have one single node because the classification problem at hand only determines between levels of one class (drafted or not). A more generalized approach to ANN setup with respects to the hidden layer would be to go through the process of pruning, meaning starting out with a number of nodes that is rather too large for the underlying problem and then subsequently eliminating acitvations with corresponding weights close to zero.

In order to train the ANN, a process called backpropagation is applied. This refers to the act of feeding labeled data into the network, analyzing the cost (squared difference between the desired and actual output) of the current combination of biases $b_1, ..., b_i$, activations $a_1, ..., a_i$ and weights $w_1, ..., w_i$ and adjusting according to their influence on any given node in the direction of greatest descent in the cost function $\nabla_a C$. Every acitvation a_L of layer L is given by the activation function $\sigma'(z_L)$ with $z_L = w_L a_{L-1} + b_L$. In this project a logistic function was chosen as the activation function:

$$\sigma'(z) = \frac{1}{1 + e^{-z}}$$

Subsequently, all changing effects for all neurons in the network are added together and propagated backwards through the network, hence the name backpropagation. This is repeated for all instances of training data resulting in an average change over all training data which is proportional to the negative gradient of the cost function:

$$\delta_L = \nabla_a C \odot \sigma'(z_L)$$

Because every activation is dependent on the bias b and weight w as well as the connected activations from the previous layer, the total change in the cost function is also dependent on the z which sums up these values for the previous layer (the layer index L is given as a superscript):

$$\frac{\partial C_0}{\partial w^L} = \frac{\partial z^L}{\partial w^L} \frac{\partial a^L}{\partial z^L} \frac{\partial C_0^L}{\partial a^L}$$

When combined with the sensitivity of the cost function to changes in bias $\frac{\partial C_0}{\partial b^L}$, this yields the combined change to be made (the previous layer is indexed by k and the current layer is indexed by j):

$$\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1}\delta_j^l \; and \; \frac{\partial C}{\partial b_j^l} = \delta_j^l, \; with \; \delta^l = ((w^{l+1})^T\delta^{l+1}) \odot \sigma'(z^l)$$

3. Data Standardization

Because an ANN posesses the ability to infer generalized relationships on the training data without imposing any limitations on distribution etc., it is not sensitive to differently scaled or centered features and therefore does not change depending on monotonic transformation of the input data. This means that there is no need to standardize the data used, event though it is differently scaled across features (see RM_DataHandling)

4. Tuning the model

As previously described, the ANN model could be tuned in terms of number of hidden layers and especially on the number of nodes in the hidden layer(s). Due to time and ressource constraints of this project, this is not done in the ANN script, which leaves room for future improvement. For the case at hand it is simply assumed to be sufficient to have one hidden layer and the number of hidden nodes is determined by the general rule of thumb descriped above. As the topic of ANNs is exceedingly deep, there are a multitude of optimization functions available to optimize ANN performance such as the Beale function, which, again due to project constraints, have been neglected in this project.

5. Implementation in R

The application of ANN in R is explained below. All code is included in the ANN script and step-by-step comments are provided.

5.1 Training the ANN Model

For training the data from years 2007 to 2013 of all unsampled and sampled datasets are used respectively. The ANN model is trained using the `nlminb()` function of the `h2o` package which implements quasi-Newton optimization methods. These are used in order to reduce computational power that would be needed for full Newton's methods in order to find the twice differentiable global minimum of a complex non-linear function.

```
BP_pred <- nlminb(start = ANN_par,
                  objective = ANN_cost,
                  gradient = ANN_grad,
                  hessian = NULL,
                  L_i_size = L_i_size,
                  L_h_size = L_h_size,
                  L_o_size = L_o_size,
                  x = x, y = y,
                  lambda = 1,
                  control = options)
```

As the `start` parameter, the combined thetas containing only 1s is passed. The `ANN_cost` and `ANN_grad` functions have been previously defined as the cost and gradient function to optimize. The `L_i_size`, `L_h_size` and `L_o_size` arguments define the dimensions of the input, hidden and output layers as described above.

The resulting vetor of the two thetas is then saved to be used for prediction on unlabeled testing data later on:

```
BP_par <- BP_pred$par
```

5.2 Variable Importance

Largely due to the universality property of ANNs, variable importance and the construction of hypermodels including preliminary methods for variable selection have not been implemented. This is following the logic that an ANN should be capable of dealing with a mixture of features of differing importance by assigning very small weights to the corresponding nodes. Because it is not clear if the neural network is tuned optimally, it is not possible to give statements about the universality of the specific ANN used in this project. Because there is a possibility that the underlying ANN is oversimplified in respects to the problem at hand, this leaves room for future improvement.

5.3 Model Performance

In order to gain an oversight of the model performance, the ANN models are tested on their training (using the years 2007 through 2013) and testing (using the year 2014) fit for all positions and all sampling methods. Both are based on unsampled data, allowing for the model's predictive ability on imbalanced class sizes, as they occur in the underlying business case, to be analyzed. The training and testing data performance can be viewed below (also see the between model comparison):

Table 1: training data performance

	No Sampling	Oversampling	Undersampling	Rose Both	Smote
QB_TP	32	73	73	73	72
QB_TN	316	180	176	172	199
QB_FP	41	0	0	0	1
QB_FN	21	157	161	165	138
WR_TP	81	159	158	159	162
WR_TN	1096	768	761	746	708
WR_FP	83	5	6	5	2
WR_FN	40	368	375	390	428
RB_TP	38	84	83	84	84
RB_TN	517	344	347	351	334
RB_FP	52	6	7	6	6
RB_FN	21	194	191	187	204
Together_TP	140	313	311	314	314
Together_TN	1924	1269	1301	1263	1229
Together_FP	187	14	16	13	13
Together_FN	87	742	710	748	782

Table 2: testing data performance

	No Sampling	Oversampling	Undersampling	Rose Both	Smote
QB_TP	8	10	11	11	10
QB_TN	98	56	52	55	58
QB_FP	3	1	0	0	1

	No Sampling	Oversampling	Undersampling	Rose Both	Smote
QB_FN	7	49	53	50	47
WR_TP	11	22	22	21	22
WR_TN	312	216	213	211	190
WR_FP	11	0	0	1	0
WR_FN	24	120	123	125	146
RB_TP	9	12	12	12	12
RB_TN	171	107	107	105	103
RB_FP	5	2	2	2	2
RB_FN	11	75	75	77	79
Together_TP	27	44	44	44	44
Together_TN	584	372	386	377	358
Together_FP	20	3	3	3	3
Together_FN	39	251	237	246	265