HW4: Artificial Neural Networks

Aljaž Konec

I. Introduction

In this report, we present an implementation of Artificial Neural Networks (ANN) for classification and regression tasks. Both implementations are compared to the results of the same tasks using the scikit-learn library. All experiments are performed on the provided housing datasets.

II. Implementation Details

Both the classification and regression tasks share a common implementation of the ANN. Up to the last output layer, both implementations share the same structure where for the activation function we use the sigmoid. The main differences between the two tasks are:

- The number of neurons in the output layer for regression is 1, while for classification it is the number of classes.
- The output layer activation function is also different. For the classification task, we use the softmax function, while for the regression task, we use the identity function.
- The loss function is different. For the classification task, we use the cross-entropy loss, while for the regression task, we use the mean squared error. Both functions use L2 regularization of the weights only.

It turns out, when computing the gradients for loss functions with regars to the weights, both loss functions have the same form of the gradient in the last layer. Because of this, the implementation of the backpropagation algorithm is the same for both tasks. For computing the optimal weights, we use Scipy's implementation of the L-BFGS-B optimization algorithm.

III. Verification of compatibility of gradient and cost functions

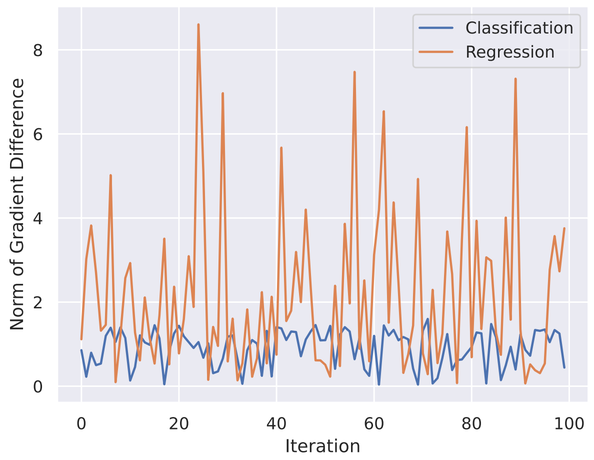

Figure 1: Norm of differences between analytical and numerical gradients for different starting weights.

To verify the compatibility of implemented gradient we compare the gradients to a numerical approximation. We instantiate the ANN with shape [3, 10, 10, 10, 2] and compute the analytical gradient via the backpropagation algorithm for one backward pass. We then compute the numerical gradient for weight w_{ij}^l between neurons i and j in layer l as:

$$\frac{\partial C}{\partial w_{ij}^l} = \frac{C(x, w_{ij}^l + \epsilon) - C(x, w_{ij}^l - \epsilon)}{2\epsilon} \tag{1}$$

where C is the cost function, x is the input to the network and $\epsilon = 1e-6$. Then we calculate the norm of differences between analytical and numerical gradients as:

$$\text{diff} = ||\nabla_{\text{analytical}} - \nabla_{\text{numerical}}|| \tag{2}$$

Figure 1 shows the norm of differences for different starting weights. To calculate the average difference we generate 100 random starting weights and compute the norm of differences for each starting weight. Then we used Bootstrap 1000 times to randomly sample 100 norms of differences. For classification task the norm of diffrences is 0.9140 ± 0.04475 and for the regression task it is 2.1775 ± 0.1848. We can see that for classification the norm is smaller and has lower variance than for regression.

IV. Predictions on the Housing Dataset

In this section we compare our implementation of ANN to multiple other models where we train the models on the provided housing datasets. All of the data is already in numerical form, so the only preprocessing that we did was to perform 80-20 split and normalize the data.

A. *Classification Task*

For our ANNClassifier we use hidden layers of size [20, 20] and regularization parameter $\lambda = 0.0001$. We compare our implementation of ANNClassifier to multiple other logistic regression models:

- **MLPClassifier** - Multi-layer Perceptron classifier from scikit-learn with same hidden layer structure but *relu* activation function.
- **MultinomialLogReg_HW3** - Multinomial Logistic Regression from HW3.
- **LogisticRegression** - Logistic Regression from scikit-learn.

Figure 2 shows the ROC curves for all tested models. The best performing models are ANNClassifier and MLPClassifier, with ANNClassifier having a slightly lower AUC score. Both logistic regression models perform worse at around 0.8 AUC score.

Model	Precision	Recall	f1	Log-Loss
ANN	0.86 ± 0.03	0.86 ± 0.03	0.86 ± 0.03	0.34 ± 0.05
MLPClassifier	0.86 ± 0.03	0.86 ± 0.03	0.86 ± 0.03	0.37 ± 0.1
MultinomialLogReg_HW3	0.62 ± 0.23	0.6 ± 0.07	0.48 ± 0.1	0.66 ± 0.01
LogisticRegression	0.74 ± 0.04	0.74 ± 0.04	0.73 ± 0.04	0.52 ± 0.3

Table I: Classification metrics for all tested models.

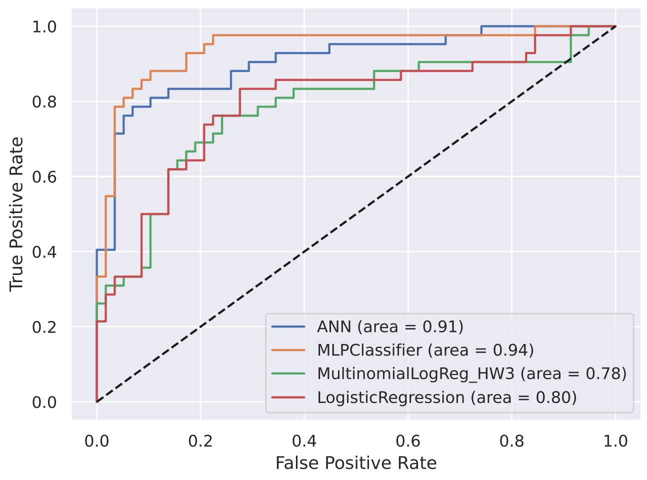

Figure 2: ROC curves for all models.

Table I shows the classification metrics for all models. The best socring models are again ANN and MLPClassifier, with ANN having a slightly lower precision and recall. This simple example shows how even a small ANN, with 2 hidden layers, can learn deep representation of the data and outperform basic Logistic Regression models.

B. Regression Task

Similarly to Classification task we build a ANNRegressor model with the hidden layers [20, 20] and compare it to multiple other models:

- **MLPRegressor** - Multi-layer Perceptron regressor from scikit-learn with same hidden layers but *relu* activation function.
- **LinearRegression** - Linear Regression from scikit-learn.

Table II shows Root Mean Squared Error (RMSE) for all tested models. To accurately test performance we performed 500 iterations of 80-20 split and calculated the mean and standard deviation of RMSE. Here we can see that all models perform similarly. Trying different number of hidden layers and neurons did not improve the performance of the ANNRegressor. This is due to the very small regression dataset, where ANNs do not have enough data to learn deep representations of the features.

Model	Mean
ANNRegression	6.04 ± 1.16
MLPRegressor	6.10 ± 1.30
LinearRegression	5.86 ± 1.127

Table II: RMSE for all tested models.

V. Implementation of final predictions

The final part of this Homework requires us to build and evaluate an ANN model on the provided train dataset. The dataset consists of 93 features and 1 calssification targert variable with 9 classes. Similarly as before we perform 80-20 split and normalize the data. The best performing model is ANNClassifier with hidden layers [128] and regularization parameter $\lambda = 0.0001$. We train the model on the training dataset and evaluate it on the test dataset. Since the training data set contains 40 000 samples, we perform iterative optimization on smaller batches with 64 samples at once. Using this, the training time is around 25 minutes. This model achieves an average per sample cross-entropy loss of 0.5882.