Artificial Neural Networks

Matej Miočić, 63180206, mm9520@student.uni-lj.si

I. Introduction

In this homework we were tasked with implementation of multi-layer fully-connected artificial neural networks (ANN) for classification and regression with backpropagation. First we show how we performed backpropagation and which activation and cost functions we used for both types of ANN. Then we numerically verify that the gradient and cost are compatible for ANN. In the end we show results with our produced model and compare it to various other models.

II. Models

A. Backpropagation

Backpropagation is an algorithm for supervised learning of artificial neural networks using gradient descent. Given an artificial neural network and a cost function, the method calculates the gradient of the cost function with respect to the neural network's weights. Starting from the final layer calculation of the gradient proceeds backwards through the network. This backwards flow of the error information allows for efficient computation of the gradient at each layer. Gradient computation begins by applying the chain rule to the cost function partial derivative which depends on output of previous layer, weights and activation function. For weighted sum of inputs for a neuron at layer l we used the following equation (1):

$$z^{(l)} = \sum_{i=1}^{n} w_i^{(l)} a_i^{(l-1)} b_i^{(l)}, \tag{1}$$

where n is the number of neurons at previous layer, w_i is the weight of i-th neuron, a_i is the activation of i-th neuron and b_i is the bias of i-th neuron.

For all layers except the output layer, we used a standard logistic activation function (2):

$$f(x) = \frac{1}{1+e^{-x}}. \tag{2}$$

Since we implemented ANN for classification and regression, we used a different cost function and a different activation function for the output layer.

1) Regression: Since we are dealing with real numbered target variable, it makes sense to compare the difference between our output and the true value. This is why we used mean squared error (MSE) as our cost function for regression (3):

$$J = \frac{1}{2m}((\sum_{i=1}^{m}(a_i^{(L)} - \hat{y}_i)^2) + \lambda\|w\|^2), \tag{3}$$

where J is the cost function, m is the number of instances, $a_i^{(L)}$ is the model output of i-th instance, $\hat{y}_i$ is the true value of i-th instance, λ regularization parameter and w is the weight vector (does not include weights for bias).

For the activation function for the last layer for regression we used a linear activation function which means we do not change the output.

In the following we show derivations of partial derivatives for regression. We show equations for the last layer (4):

$$\frac{\partial J}{\partial W^{(L)}} = \frac{\partial Z^{(L)}}{\partial W^{(L)}} \times \frac{\partial A^{(L)}}{\partial Z^{(L)}} \times \frac{\partial J}{\partial A^{(L)}}, \tag{4}$$

where J is our cost function, W is our matrix of weights, Z is the weighted input and A is our activation function of Z. We denote last layer with L.

We show derivation of each componenet individually (5):

$$\frac{\partial Z^{(L)}}{\partial W^{(L)}} = A^{(L-1)}, \quad \frac{\partial A^{(L)}}{\partial Z^{(L)}} = 1, \quad \frac{\partial J^{(L)}}{\partial A^{(L)}} = A^{(L)} - Y. \tag{5}$$

And every other layer (6):

$$\frac{\partial J}{\partial W^{(l)}} = \frac{\partial Z^{(l)}}{\partial W^{(l)}} \times \frac{\partial A^{(l)}}{\partial Z^{(l)}} \times \frac{\partial Z^{(l)}}{\partial A^{(l-1)}} \times \frac{\partial A^{(l+1)}}{\partial Z^{(l+1)}} \times \frac{\partial J}{\partial A^{(l+1)}}, \tag{6}$$

where we denote any layer but last with l. We show derivation of new components (7):

$$\frac{\partial Z^{(l)}}{\partial A^{(l-1)}} = W^{(l)}, \quad \frac{\partial A^{(l)}}{\partial Z^{(l)}} = A^{(l)}(1 - A^{(l)}). \tag{7}$$

2) Classification: In contrast to regression, we are not trying to predict a real number but a class. We do not pick the most probable class, but we output probabilities for each class. We could again use MSE as our cost function, but we want to make our model more certain at predicting the right class. This is why we decided to use cross-entropy as our cost function (8):

$$J = \frac{1}{m}((-\sum_{i}^{m}\sum_{j}^{C} \hat{y}_{ij} \log(a_{ij}^{(L)})) + \frac{\lambda}{2}\|w\|^2), \tag{8}$$

where J is the cost function, m is the number of instances, C is the number of classes, $a_{ij}^{(L)}$ is the predicted probability for i-th instance, $\hat{y}_{ij}$ is the true binary value of i-th instance and λ as regularization parameter.

To obtain probabilities of each class we used a softmax activation function on the last layer(9):

$$f(x_i) = \frac{e^{x_i}}{\sum_j^C e^{x_{ij}}}. \tag{9}$$

In the following we show derivations of partial derivatives for classification. We show equations for the last layer (10):

$$\frac{\partial J}{\partial W^{(L)}} = \frac{\partial Z^{(L)}}{\partial W^{(L)}} \times \frac{\partial J^{(L)}}{\partial Z^{(L)}}, \tag{10}$$

and for each component (11):

$$\frac{\partial Z^{(L)}}{\partial W^{(L)}} = A^{(L-1)}, \quad \frac{\partial J^{(L)}}{\partial Z^{(L)}} = A^{(L)} - Y. \tag{11}$$

For the other layers it is very similar like we discussed for regression.

B. Numerical verification

In this section we show how we verified that our backpropagation gradients and cost are compatible for both types of ANN. We used XOR dataset and for each iteration we computed gradients numerically and with backpropagation. We calculated gradients numerically as (12):

$$f(x)' = \frac{f(x + he_i) - f(x)}{h}, \tag{12}$$

where f is our cost function, x is weights vector, h is a very small number 10^{-8} and e is a unit vector with 1 on i-th position.

In Figure 1 we show norm of the differences between numerical and backpropagation gradients for 55 iterations using 1 hidden layer with 2 nodes. We can see that the differences are extremely low, which means that backpropagation gradients and cost are compatible.

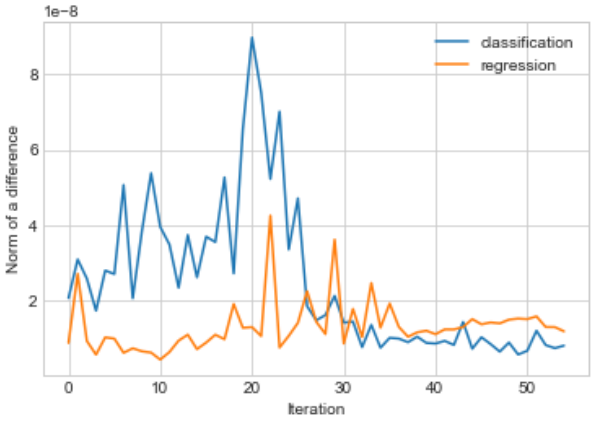

Figure 1. Norm of the differences between numerical and backpropagation gradients for 55 iterations for classification and regression.

C. Results

1) Regression: We tested regression on *housing2r.csv* dataset. We split the first 160 instances into train set and the rest 40 instances into test set. We used 1 hidden layer and used 5-fold cross-validation repeated 10 times on training set to obtain the optimal numbers of nodes and λ parameter. For each iteration we also standardized the dataset using information from only training folds.

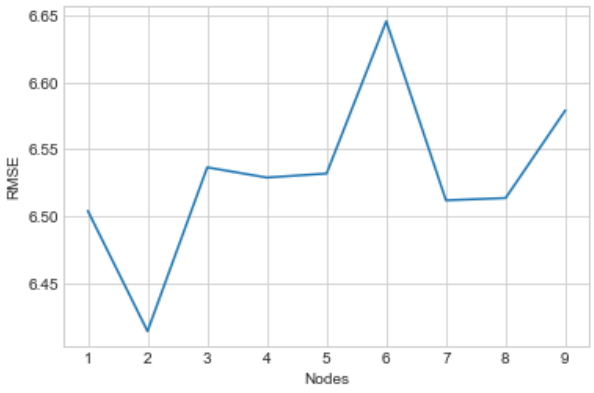

Figure 2. RMSE for prediction of test set of *housing2r.csv* dataset with 1 hidden layer with different number of nodes.

In Figure 2 we show obtained RMSE on the test set of *housing2r.csv* dataset for different number of nodes with obtained optimal λ on training folds. We show that we obtained the lowest RMSE when using only 2 nodes. Because we have only 160 training instances, increasing the number of nodes makes the neural network perform worse. We already performed evaluation on this dataset with kernelized ridge regression and support vector regression. We show results from previous homework in Figure 3, where we observe similar RMSE when using support vector regression with RBF kernel.

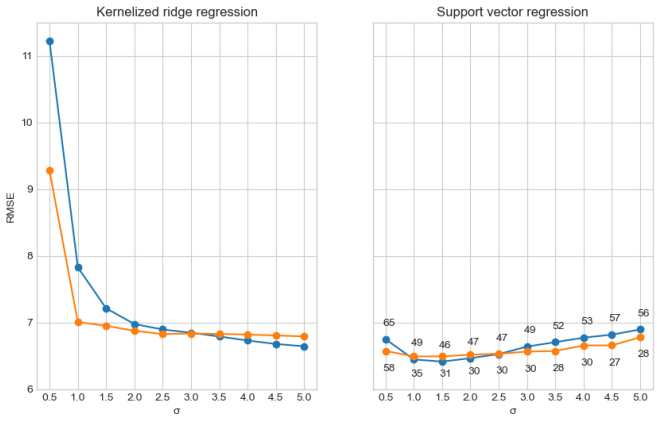

Figure 3. RMSE for prediction of test set with kernelized ridge regression (left) and support vector regression (right) using **RBF** kernel for parameters $\sigma \in \{0.5, 1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0, 4.5, 5.0\}$, with $\lambda = 1$ (blue line) and tuned λ (orange line). For SVM we also display number of support vectors (for $\lambda = 1$ above and tuned λ below).

2) Classification 1: For this task we used *housing3.csv* dataset. This time we had more instances and just like before, we split the training set into first 400 instances and the test set into remaining 100 instances. Just like for regression we did 5-fold cross-validation repeated 10 times on the training set with 1 hidden layer to obtain the optimal numbers of nodes and λ parameter.

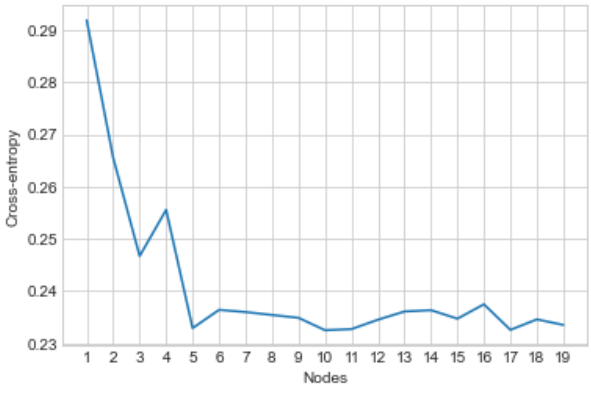

Figure 4. Cross-entropy for prediction of test set of *housing3.csv* dataset with 1 hidden layer with different number of nodes.

In Figure 4 we show obtained cross-entropy on the test set for different number of nodes with obtained optimal λ on training folds. We can see that cross-entropy decreases with increasing number of nodes and subsides after using less than 5 nodes.

We conclude that this is because we have more instances in the training set which can train more number of parameters.

From previous homework we could use trees for classification, but we did not implement it such that the model returns probabilites, so we would not be able to calculate cross-entropy. This is why we decided to use multinomial logistic regression as comparison. When using the same training and test set, we obtained 2.85 as the value of cross-entropy. We observe major improvements when using the neural network.

3) Classification 2: For the final task we recieved a dataset containing 9 classes and 50,000 instances. This time training took a bit longer, so we performed 5-fold cross-validation without repetition, to obtain best λ and only for a neural network with 1 hidden layer and 20 nodes. We also limited the fmin_l_bfgs_b function with 500 iterations, since we noticed that cross-entropy change was minimal after that point. We then performed 100 bootstrap iterations and evaluated the model on the out of bag samples. We computed 95% confidence interval for cross-entropy which is: [0.572, 0.574]. Our model beats a random classifier for which cross-entropy mean is 2.48 and a classifier where all probabilies for each class is 1/9 for which cross-entropy mean is 2.86. We also measured time elapsed for each iteration for which mean is 229 seconds, which is almost 4 minutes. Then we predicted the test dataset containing almost 12,000 instances and saved the results to *final.txt*.

III. Conclusion

We learned how to implement artificial neural networks with backpropagation. We learned which activation and cost functions to use to obtain desired results. We showed that backpropagation gradients are close to numerical ones. Then we showed that compared to models from previous homeworks, neural network outperformed them all. For further work we could try other activation functions such as ReLU for non output layers.