Artificial Neural Networks

Biljana Vitanova
MLDS1 2024/25 , FRI, UL
bv7063@student.uni-lj.si

I. Introduction

In this project, we implemented an artificial neural network (ANN) for classification and verified it by comparing analytical and numerical gradients. We extended the model to regression tasks and different activation functions. Additionally, we trained model for a tissue classification competition.

II. Model Implementation

A. Methodology

In the first part, we implemented a fully connected neural network for a classification task, with support for an arbitrary number of hidden layers, each using the sigmoid activation function. During training, we focus on four steps: forward propagation, loss computation, backpropagation, and parameter update.

Forward propagation is defined as:

$$Z^{(l)} = W^{(l)} A^{(l-1)} + b^{(l)}$$
$$A^{(l)} = \sigma(Z^{(l)})$$

where $\sigma(z) = \frac{1}{1+e^{-z}}$ is the sigmoid activation function.

For loss computation, we use the cross-entropy loss.

Backpropagation is defined as:

$$dZ^{(L)} = \frac{1}{m}(A^{(L)} - Y)$$
$$\nabla_{W^{(L)}} = dZ^{(L)}(A^{(L-1)})^T$$
$$\nabla_{b^{(L)}} = \sum_{i=1}^{m} dZ_i^{(L)}$$

and for hidden layers:

$$dA^{(l)} = (W^{(l+1)})^T dZ^{(l+1)}$$
$$dZ^{(l)} = dA^{(l)} \odot \sigma'(Z^{(l)})$$
$$\nabla_{W^{(l)}} = dZ^{(l)}(A^{(l-1)})^T$$
$$\nabla_{b^{(l)}} = \sum_{i=1}^{m} dZ_i^{(l)}$$

where $\sigma'(z) = \sigma(z)(1 - \sigma(z))$ and $\odot$ denotes element-wise multiplication.

Weight and bias update is defined as:

$$W^{(l)} \leftarrow W^{(l)} - \eta \nabla_{W^{(l)}}$$
$$b^{(l)} \leftarrow b^{(l)} - \eta \nabla_{b^{(l)}}$$

where η is the learning rate.

B. Results and Discussion

In order to verify the correctness of the computed gradients, we calculated them numerically by approximating:

$$\frac{\partial \mathcal{L}}{\partial \theta} \approx \frac{\mathcal{L}(\theta + \varepsilon) - \mathcal{L}(\theta - \varepsilon)}{2\varepsilon}$$

where ε is a small constant.

We then compared the numerical gradients to the analytical gradients computed by backpropagation, and evaluated their relative difference to ensure that the implementation is correct. The relative difference between the numerical and analytical gradients for the first layer was 6×10^{-10}.

To further verify that our implementation can successfully learn, we tested the network on two simple datasets: `doughnut.tab` and `squares.tab`. A network with one hidden layer of 5 neurons, a learning rate of 0.1, and a maximum number of epochs set to maximum of 50000 was able to achieve perfect fitting on both datasets.

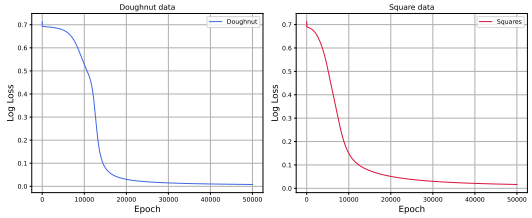


Fig. 1: Fitting on doughnut and square datasets.

Specifically, the network reached an accuracy of 1.0 and a log loss smaller than 0.01 on both training sets.

III. Extension of the Model

A. Methodology

In the second part, we extended the neural network to support regression tasks. The key changes are that the output layer now uses a linear activation function instead of softmax, and the loss function is mean squared error (MSE) instead of cross-entropy. Additionally, the targets

are treated as continuous scalar values rather than one-hot encoded vectors, which means that the output layer contains one neuron.

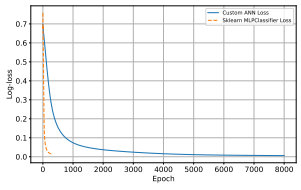
We also introduced L_2 regularization, controlled by the `lambda_` parameter, to prevent overfitting. The regularization is applied by adding an additional penalty term proportional to the sum of squared weights to the loss function, and by modifying the gradient descent updates to include the derivative of this penalty.

Furthermore, the network was extended to support multiple activation functions, such as ReLU, leaky ReLU, tanh, and sigmoid, with the possibility to specify a different activation function for each hidden layer individually.

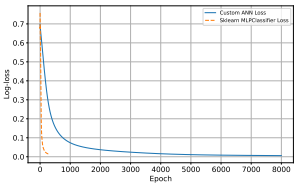
B. Results and Discussion

To further verify the proper implementation of the regression and classification ANNs, we compared our custom network against the `MLPClassifier` and `MLPRegressor` implementations from the `scikit-learn`. We initialized both models with the same number of epochs, layers, and activation functions. The only difference was the gradient optimization method: since `scikit-learn` does not provide a plain gradient descent option, we used stochastic gradient descent (SGD) instead.

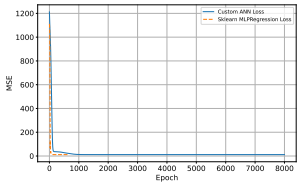
For the classification task, we used the `doughnut.tab` dataset, and for regression synthetically generated. From Figure 2, it can be observed that both the scikit-learn and the custom models converge similarly, although the custom implementation requires a larger number of epochs to achieve comparable results.



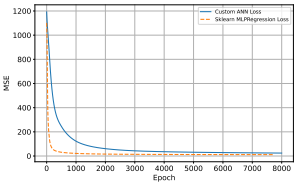
(a) ReLU activation - classification



(b) Tanh activation - classification



(c) ReLU activation - regression



(d) Tanh activation - regression

Fig. 2: Comparison of custom and scikit-learn models on classification and regression tasks.

To evaluate the effect of regularization, we compared the classification accuracy with and without L_2 regularization. From Table I we can see that without regularization, the network overfits the training data, achieving perfect accuracy, while with regularization, does not fully fit the training data.

Model	Accuracy
Without regularization ($\lambda = 0$)	1.0000
With regularization ($\lambda = 1$)	0.8519

TABLE I: Impact of L_2 regularization on classification accuracy.

IV. Competition

For the competition, where the goal was to predict the class of each tissue sample, I used only the spectral information. The final model was a simple one-layer network with 10 neurons and the `tanh` activation function. On the partially available test set, the model achieved a log-loss of 0.61. Using 5-fold cross-validation, the performance was 0.5493 ± 0.2162.

I tried different feature transformations, such as using PCA, FFT features, and adding extra information like segmenting the signal and computing the mean and variance. However, none of these approaches improved the performance. In particular, PCA was surprising, as the first 10 components explained most of the variance, and plotting the first two principal components showed clear clustering.

I also tried building more complex models with more layers, longer training, and splitting the data into training and validation sets. In most cases, although both the training and validation losses decreased, the final score on the partially available test set remained low.

V. Conclusion

In this project, artificial neural networks (ANNs) were successfully implemented. The custom ANN achieved similar performance to library models but required more epochs to converge. For the competition, the best results were obtained with a simple one-layer model using only spectral features.