HW4: Artificial Neural Networks

Žan Pušenjak
MLDS 2024/25 , FRI, UL
zp4613@student.uni-lj.si

***Abstract*—We implemented artificial neural network models and tested them on a provided data set of square and doughnut points. We were able to achieve perfect prediction performance on both datasets. We extended the implementation to enable a regression model and custom regularization and per-layer activation function setup.**

I. Implementation

We implemented the classification network with the option of adding additional hidden layers specified in an array. We did not use stochastic gradient descent but rather computed the gradient on the full epoch. Backpropagation was checked by computing numerical gradients with

$$n\_grad(x) = \frac{f(x-\epsilon) - f(x+\epsilon)}{2\epsilon}$$

where $\epsilon = 10^{-6}$.
We changed each weight in the layer and computed the gradient than reset the changed weight to its original value and repeated that with the next weight. Such procedure was repeated for all weights and similarly for all gradients. Than we compared the weights gradients and bias gradients element-wise. The gradients matched up to the fifth decimal which was an indicator, that the backpropagation was computed properly.

II. Fitting the test data

We were able to perfectly fit the provided datasets. The parameters used can be observed in I, where

- Units - is an array of hidden layers. The numbers represent the size of the layer.
- γ - the learning rate.
- α - learning rate decay calculated with the formula $\gamma = \gamma_0 e^{-\alpha i}$, where i is the current epoch starting at 0 and γ_0 is the initial learning rate.
- Epochs - the number of epochs.
- λ - the regularization rate.

Note that since we wanted to overfit on the data we set the λ parameter to 0.

III. Extended implementation

A. Regression neural network

We further expanded the implementation to support a regression problem. The classification and regression networks differ in two places.

TABLE I
Parameters used to predict the provided datasets

Dataset	Units	γ	α	Epochs	λ
Squares	[5]	0.01	0.001	4100	0
Doughnut	[3]	0.01	0.001	1800	0

- We removed the use of activation function in the last layer
- We did not one-hot encode the `y` in the `fit(X,y)` function.

Note that the output layer of a regression network consists of only one neuron that has no activation function, but this information is implicitly given by not one-hot encoding the `y`.

B. Regularization

Support for regularization was added. By default the model uses `L2` regularizer but it supports other types (eg. `L1`), the user has to specify the regularizer when initializing the fitter. The strength of the regularization is determined with the λ parameter.

C. Activation functions

Lastly we enabled the user to specify activation functions for each layer except the last one, where the softmax function is used to obtain the class probabilities in the classification case or no function in case of regression.

We added support for `sigmoid`, `ReLU` and `TanH` activation functions. The default activation function is `Sigmoid`.

IV. Comparison

We compared our implementation with the existing PyTorch[1] implementation. The PyTorch implementation did not converge with as few epochs. The PyTorch network took more time for the same amount of training, but its weights and biases were smaller in comparison to our. Once both networks converged the probabilities were comparable differentiating only on the third decimal point.

We attribute the differences to the different approaches for the descent, PyTorch implementation uses stochastic approach while we used normal gradient descent. The

[1] https://pytorch.org/

initialization is another big factor, both implementations use the He initialization[2], but since the initializations are random, the results will vary.

[2]https://paperswithcode.com/method/he-initialization