

Programming Test:

Objective: Use a 1-hidden layer neural network model that adapts to the most suitable activation function according to the data set. The ANN model can learn for itself the best AF to use by exploiting a flexible functional form, $k_0 + k_1 x$ with parameters k_0 ; k_1 being learned from multiple runs.

Datasets :Data Sets for experiments and results: Bank-Note, Iris, Wisconsin Breast Cancer, and MNIST.

EXPECTED RESULTS: A technical report containing implementation details (algorithm, initial settings such as sampling the parameters k_0 ; k_1 from some distribution, parameter updates on epochs, final parameter values at the end of the training, train vs test loss, train and test accuracy, F1-Score, plot of the loss function vs. epochs, Codebase hosted on github linked in the report)–Maximum 3 pages Grid search/brute force NOT allowed Data Sets for experiments and results: Bank-

Report:

The entire objective was to test out different ANN models with most suitable activation function depending on the datasets used.

I made an ANN network from Scratch with 2 layers one with the input and activation function, this case relu. Then the output layer with the desired dimension.

Following functionality is needed,

1. Initialize the parameters.
2. Choose an optimization algorithm.

Repeat these steps:

1. Forward propagate an input.
2. Compute the cost function.
3. Compute the gradients of the cost with respect to parameters using backpropagation.

4. Update each parameter using the gradients, according to the optimization algorithm.

The primary focus was to find out the best possible Activation Function depending on the datasets given as an input. After the model initialization, we call the model using a loop of Activation functions. Iteration of 150 epochs works better with dropout. The Model has a Flatten and a dropout layer which defines the functionality of the Hidden layer and Input layer but, does not act as an individual layer as I was supposed to take only 1 Hidden layer.

Few of the datasets needed pre-processing. So depending on the distribution of these pre-processed data, features have been selected for future use. A comparative Loss vs Epochs for different AF is given at the end of the iteration.

We then call the parameters initial value using `get_config()` feature. The the required results are produced subsequently.

Based on the Math indicated in the PDF we can come to a conclusion that for a ANN model below are the requirements,

1. Minimize the Cost function to find good parameters.
2. Good Initialization.
3. According to the gradient descent formula above, the direction and magnitude of the parameter update are given by the learning rate multiplied by the slope of the cost function at a certain point W .
4. Adaptive learning-rate algorithms such as Momentum Adam and RMSprop help adjust the learning rate during the optimization process.
5. Batch size is the number of data points used to train a model in each iteration. Choosing the right batch size is important to ensure convergence of the cost function and parameter values, and to the generalization of your model.
6. $W = W - \alpha dW$ -- (Stochastic) Gradient Descent
 1. Gradient descent can use parallelization efficiently, but is very slow when the data set is larger the GPU's memory can handle. The parallelization wouldn't be optimal.

2. Stochastic gradient descent usually converges faster than gradient descent on large datasets, because updates are more frequent. Plus, the stochastic approximation of the gradient is usually precise without using the whole dataset because the data is often redundant.
3. Of the optimizers profiled here, stochastic gradient descent uses the least memory for given batch size.

Needed Packages as follows,

1. pip install pydot
2. pip install keras, tensorflow.
3. Assuming that other packages like numpy, pandas as pre-installed.

Note:

For hidden layers the best option to use is ReLU, and the second option you can use is SIGMOID. For output layers the best option depends, so we use LINEAR FUNCTIONS for regression type of output layers and SOFTMAX for multi-class classification.






Some questions that can be answered by bringing in more complexity to the models

1. I understand that after epoch 150, my model is likely overfitting, but besides Dropout, what else can I do to reduce this in such a small network?
2. Also, given the limited amount of training data, how much better can I expect to perform even with more regularization?
3. Is adding more depth guaranteed to increase accuracy (as long as I regularize each layer with, e.g., Dropout)?

The MNIST model loss for both training and testing(Validation) justifies that few complexity in the models are required to get a desired results, even though its produced a fascinating Accuracy and F1 Scores

For both Wisconsin Breast Cancer and Bank-Note desired F1 score are not produced because of the lack of data and complexity in the model. Sometimes the best models are linear like regression or decision tree.

Comparison of Scores

<u>Aa</u> Datasets	 F1 Score	 Accuracy	 Loss	 Precision	 Recall
<u>MNIST</u>	0.9823	0.9824	0.1654	0.9824	0.9823
<u>Wisconsin Breast Cancer</u>	0.5484	0.7692	9.4291	0.3799	1.0
<u>Iris</u>	0.9843	0.9736	0.0397	0.9843	0.9843
<u>Bank-Note</u>	0.6117	1.0	0.0015	0.4429	1.0

Note:

Loss entropy is chosen based on the output's number of classes.

Following pre-requisites are attached,

1. Datasets excluding MNIST
2. Model and its weights for future use