deepk

Loading data

To load some data, you have to create a DataLoader object and use its methods.

`DataLoader::LoadCSV`

Read a CSV file and write the data in a 2 dimension vector of strings.
parameters :

- std::filesystem::path path -> path to the CSV file
- bool shuffle -> shuffle the dataset if true

std::vector<std::vector<std::string>> LoadCSV(std::filesystem::path path, bool shuffle = true);

`DataLoader::SplitDataset`

Split a 2D vector of strings and split it accordingly to the wanted percentage in 2 2D vectors of strings.
parameters :

- std::vector<std::vector<std::string>> dataset -> dataset to split.
- double percentage -> percentage -> proportion to split the dataset.
- std::vector<std::vector<std::string>> *set_one -> new 2D vector that will contain “percentage” of the dataset content.
- std::vector<std::vector<std::string>> - > new 2D vector that will contain the rest of the dataset content.

void SplitDataset(std::vector<std::vector<std::string>> dataset, double percentage, std::vector<std::vector<std::string>> *set_one, std::vector<std::vector<std::string>> *set_two);

`DataLoader::RemoveColumn`

Remove a column from a 2D vector dataset.
parameters :

- std::vector<std::vector<std::string/double>> &dataset -> dataset in which the column will be removed.
- int index -> index of the column to remove

building a neural network

To build a neural network, you have to create a SequentialModel object and use its methods.

`SequentialModel::SetLossFunction`

select the loss function that will be used in the neural network.
The available loss functions are:

- Mean squared error

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(\hat{Y}_i - Y_i)^2$$

- Binary cross entropy

$$H_p(q) = -\frac{1}{N}\sum_{i=1}^{N} y_i \cdot log(p(y_i)) + (1 - y_i) \cdot log(1 - p(y_i))$$

- Hinge

$$\mathcal{L}(Y, \dot{Y}) = \frac{1}{N}\sum_{n=1}^{N} \max(0, 1 - \dot{y}_n \cdot y_n)$$

parameters:

- std::string l -> name of the loss function to use (“mean_squared”, “binary_cross_entropy” or “hinge”).

`SequentialModel::SetLearningRate`

Set the learning rate used during the training of the neural network.
parameters:

- double lr -> learning rate.

`SequentialModel::AddLayer`

Add a layer in the neural network. The layers can use the following activation functions:

- Logistic

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

- Rectified linear unit

$$RELU(x) = \begin{cases} 0 \; if \; x < 0 \\ x \; if \; x >= 0 \end{cases}$$

- hyperbolic tangent

$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

parameters:

- int neurons_nbr -> number of neurons in the layer.
- std::string activation_function -> name of the activation function that the layer will use (“Logistic”, “ReLU” or “Tanh”).

`SequentialModel::UniDistribInit`

Initialize the weights of the neural network with a uniform distribution between -1 and 1.

`SequentialModel::UniDistribInit`

Initialize the weights of the neural network with the He method.

`SequentialModel::Train`

Use the forward and backward propagation algorithm to train the neural network by updating the synaptic weights through the training iterations.

parameters:

- vector<vector<double>> training set -> dataset used to train the neural network
- vector<vector<double>> train_label_set -> dataset containing the label of the training set .
- vector<vector<double>> test_set -> dataset used to evaluate the neural network.
- vector<vector<double>> test_labels_set -> dataset containing the label of the test set.

`SequentialModel::Predict`

Take input data to make a prediction with the trained neural network.

parameters:

- vector<double> input -> input data.

Displaying training performances histories

`SequentialModel::DisplayAccuraciesHistory`
Create and save a graph showing the evolution of the accuracies obtained at each iteration of the training.

`SequentialModel::DisplayPrecisionHistory`
Create and save a graph showing the evolution of the precisions obtained at each iteration of the training.

`SequentialModel::DisplayRecallHistory`
Create and save a graph showing the evolution of the recall scores obtained at each iteration of the training.

`SequentialModel::DisplayRecallHistory`
Create and save a graph showing the evolution of the F1 scores obtained at each iteration of the training.

`SequentialModel::DisplayLossesHistory`
Create and save a graph showing the evolution of the losses obtained at each iteration of the training.