Deep Learning Methods
Project 1

Jakub Kała & Krzysztof Spaliński

March 2020

The goal of the first project for Deep Learning Methods class is to create a low level implementation of multi layer perceptron in programming language of choice (in our case it is Python with NumPy package).

Table of contents

1 Documentation

Our design was to create a set of classes for a user with basic knowledge of neural network architectures. There are two main classes made for users to interact with:

- `NeuralNetworkWrapper` – allows user to create a neural network. It includes some tools for the user, including learning in batches, split for training and validation set, drawing a loss function, etc.
- `LearningProcessVisualisation` – allows user to visualise the learning process on a 2D dataset.

There is also `Optimizer` class with optimizers to use for neural network training.

1.1 `NeuralNetworkCore` class

This class allows user to create a multi layer perceptron. Upon initialisation, user has to specify number of neurones in each layer (including input layer), activation function for each layer, loss function, learning rate, optimizer and whether the there should be bias in linear transformation part. Instances of `NeuralNetworkCore` class have two methods: train and predict.

1.1.1 `NeuralNetworkWrapper` initialization

- `input_dim` [`int`]: a dimension of input observations.
- `neuron_numbers` [`list` of ints]: numbers of neurons in each layer, length of the list specifies a number of layers in the network.
- `activation_functions` [`list` of strings]: names of activation function for each layer. Available names are: `sigmoid`, `relu`, `leaky_relu`, `tanh` and `softmax`.
- `loss_function` [`str`]: name of the loss function that is optimized during the training. Available names are: `logistic_loss` and `max_likelihood_loss`
- `learning_rate` [`float`]: learning rate used to update the weights during backpropagation.
- `optimizer` [`Optimizer`, optional]: optimizer used for updating weights. Default is stochastic gradient descent. More information on the subject can be found it `Optimizer` class documentation.
- `batch_size` [`int`, optional]: batch size for training. By default set to 128.
- `bias` [`boolean`, optional]: if `False` then bias is set to zero and is not updated in the process of backpropagation. By default set to `True`.

1.1.2 `train` method

Input:

- `X_train` [`numpy.ndarray`]: 2 dimensional array containing training observations (in rows).
- `y_train` [`numpy.ndarray`]: 2 or 1 dimensional array containing training labels. If labels are real numbers, then it should be a vector ($1 \times n$ matrix), if labels are in higher dimensions then they should be in rows ($m \times n$ matrix, where m is dimension of a label).
- `epochs` [`int`]: numbers of epochs of neural network. One epoch means going forward and backwards in the network with whole training dataset.
- `validation_split` [`int`, optional]: fraction of the original training set that is taken to make validation set. By default set to 0.1.
- `verbosity` [`boolean`, optional]: If set to `True`, learning progress will be printed on the go. By default set to `True`.
- `cache_weights_on_epoch` [`boolean`, optional]: weight cache is used to create visualisation of learning state. By default set to `False`.

Output:

- `loss_on_iteration` [`list` of floats]: value of loss function on training dataset after each iteration.

1.1.3 `predict` method

Input:

- `X` [`numpy.ndarray`]: a dataset to predict on (observations in rows).

Output:

- `y_hat` [`numpy.ndarray`]: array of predictions for input dataset.

1.1.4 `plot_loss` method

Input:

- `None`

Output:

- Plot of the loss function (y-axis) to the number of epochs (x-axis) is displayed. Includes loss on validation loss (if present).

1.2 `LearningProcessVisualisation` class

1.3 `Optimizer` class

2 Testing on classification datasets

3 Testing on regression datasets

4 Digit recognition Kaggle competition