UNIVERSITÀ
DI TORINO

Deep Learning

Francesco Marchisotti

`MFNET` – A simple Machine Learning Library

Professor:
Matteo Osella

academic year 2024/2025

Abstract

This document presents `mfnet`, a simple machine learning library developed as a final project for the Deep Learning course. The key feature is the implementation of the backpropagation algorithm, enabling the training of neural networks through gradient descent.

The library is then compared with PyTorch on two simple tasks: a regression (on the California Housing dataset) and a classification (on the MNIST dataset).

Contents

Signal Flow in Neural Networks

A Feedforward Fully Connected Neural Network (FCNN) is a Neural Network architecture where each neuron in one layer is connected to every neuron in the subsequent layer.

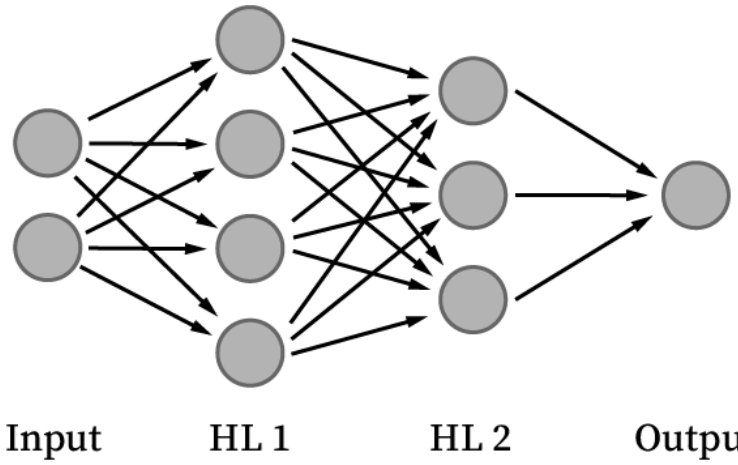

Figure 1: A Feedforward Fully Connected Neural Network (FCNN).

A FCNN learns by iteratively performing two main steps: the forward pass and the backward pass.

Forward pass The input data is propagated through the network, layer by layer, to produce an output prediction. This prediction is then compared to the true target values using a loss function, which quantifies the prediction error.

Backward pass The network uses the computed loss to adjust its internal parameters. This is done by propagating the error backward through the network and updating the weights to minimize the loss. The process of forward and backward passes is repeated for multiple iterations, gradually improving the model's performance.

This step is the heart of the learning process, as it allows the network to learn from its mistakes and improve its predictions over time.

Part I
Implementation of `mfnet`

1 Basic Data Structure

The fundamental data structure in `mfnet` is the Tensor. In this context, a Tensor is simply a `numpy` array with a fixed data type of `numpy.float64`.

Tensors are used throughout `mfnet` to represent inputs, outputs, intermediate activations, weights, and gradients within the neural network.

2 Layer

In `mfnet`, a layer is a fundamental building block of the neural network. Each layer consists of a set of neurons, and it performs a specific transformation on the input data. The two main types of layers implemented in `mfnet` are Linear layers and Activation layers.

Layers are stacked together to form a complete neural network, with the output of one layer serving as the input to the next layer. Each layer is responsible for maintaining its own parameters and computing gradients during the backpropagation process.

Each of the layers in `mfnet` expects the input to be in the form of a matrix with shape $(n_{\text{features}} + 1, n_{\text{samples}})$, where n_{features} is the number of input features and n_{samples} is the number of input samples. The first row of this matrix is reserved for a bias term, which is always set to 1.

2.1 Linear Layer

2.1.1 Forward pass

The Linear layer in `mfnet` applies a linear transformation to the input data, performing a change in dimensionality from $n_{\text{in\_features}}$ to $n_{\text{out\_features}}$. Mathematically, this can be represented as:

$$y = b + Wx \tag{1}$$

where:

- x is the $(n_{\text{in\_features}}, 1)$ input vector,
- W is the $(n_{\text{out\_features}}, n_{\text{in\_features}})$ weight matrix,
- b is the $(n_{\text{out\_features}}, 1)$ bias vector, and
- y is the $(n_{\text{out\_features}}, 1)$ output vector.

Implementation Details The actual implementation is a bit different: the first key difference is that the bias term is absorbed inside the weights matrix, and the input vector is augmented with an additional constant value of 1. The layer expects this "bias feature" to be already present in the input data, and propagates it to the next layer by adding a row of $\begin{pmatrix}1 & 0 \cdots 0\end{pmatrix}$ to the weights matrix. This allows us to rewrite the equation as:

$$\begin{pmatrix}1\\ y\end{pmatrix} = \begin{pmatrix}1 & 0 \cdots 0\\ b & W\end{pmatrix} \begin{pmatrix}1\\ x\end{pmatrix} \tag{2}$$

The second key difference is that, instead of feeding one data point at a time to the network and heavily relying on inefficient for loops, we can feed a batch of data points at once, and leverage efficient matrix operations. This means that the input x is actually a matrix X where each column represents a different data point, and the output y is also a matrix Y where each column corresponds to the output for each input data point. The equation then becomes:

$$\begin{pmatrix} 1 \cdots 1 \\ Y \end{pmatrix} = \begin{pmatrix} 1 & 0 \cdots 0 \\ b & W \end{pmatrix} \begin{pmatrix} 1 \cdots 1 \\ X \end{pmatrix} \tag{3}$$

Switching to standard backpropagation notation, we can summarize the forward pass of a Linear layer as:

$$Z^{[l]} = W^{[l]} A^{[l-1]} \tag{4}$$

where:

- $A^{[l-1]}$ is the input of the Linear layer l, with $A^{[0]}$ being the input data (shape $(n_{\text{features}} + 1, n_{\text{samples}})$),
- $W^{[l]}$ is the weights matrix of the Linear layer l (shape $(n_{\text{out\_features}} + 1, n_{\text{in\_features}} + 1)$), and
- $Z^{[l]}$ is the output of the Linear layer l (shape $(n_{\text{out\_features}} + 1, n_{\text{samples}})$).

All these Tensors already include all the necessary additions to correctly handle the bias.

The Linear layer forward method also stores its input $A^{[l-1]}$ for use in the backward pass.

2.1.2 Backward pass

The Linear layer's responsibility is to compute the gradients of the loss with respect to its weights during the backward pass.