STATISTICAL MACHINE LEARNING (SML2020)
2. COMPUTER LAB
Backpropagation

Jan Drchal

1 Overview

The goal of this laboratory lab is to implement backpropagation for MultiLayer Perceptron (MLP). Backpropagation is a method computing the gradient $\frac{\partial \mathcal{L}}{\partial \boldsymbol{w}}$, which is in turn used by optimization methods such as gradient descent in order to learn the network.

In backpropagation we divide the network into layers (modules). Each layer l is specified by the three following messages:

- *forward message* ($\boldsymbol{z}^{l+1} = \boldsymbol{f}(\boldsymbol{z}^l)$), describing the function realized by the layer,
- *backward message* ($\frac{\partial \boldsymbol{z}^{l+1}}{\partial \boldsymbol{z}^l}$), Jacobian matrix giving sensitivities of all layer outputs w.r.t. to all of its inputs and
- *parameter message* ($\frac{\partial \boldsymbol{z}^{l+1}}{\partial \boldsymbol{w}^l}$), giving sensitivities of all layer outputs w.r.t. to all layer parameters.

Let $\boldsymbol{\delta}^l = \frac{\partial \mathcal{L}}{\partial \boldsymbol{z}^l}$ be the sensitivity of the loss w.r.t. the module input for layer l, then

$$\delta_i^l = \frac{\partial \mathcal{L}}{\partial z_i^l} = \sum_j \delta_j^{l+1} \frac{\partial z_j^{l+1}}{\partial z_i^l}. \tag{1}$$

Parameter gradient can be then computed using:

$$\frac{\partial \mathcal{L}}{\partial w_i^l} = \sum_j \delta_j^{l+1} \frac{\partial z_j^{l+1}}{\partial w_i^l}. \tag{2}$$

Your task can be summarized by the following steps:

1. Implement all messages for linear, ReLU, and softmax layers as well as for the multinominal cross-entropy loss.
2. Implement the backpropagation.
3. Train MLP classifier using Stochastic Gradient Descent (SGD) on an artificial spiral dataset (see Figure 1).
4. Modify your layer implementations so these can cope with numerical instabilities (e.g., underflow, overflow, or division by zero). This involves defining a compound layer combining softmax with cross-entropy loss. Train MLP classifier on the well known MNIST

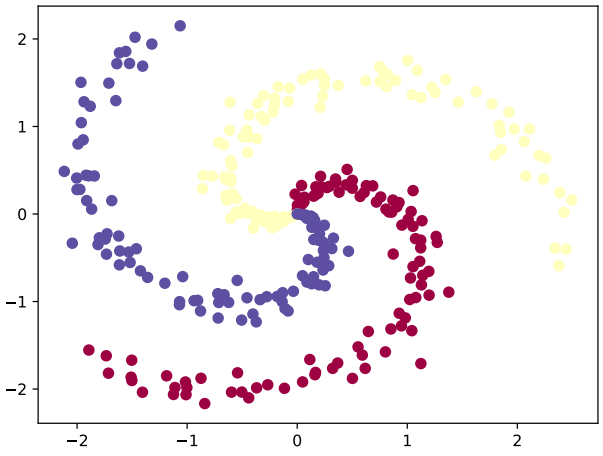

Figure 1: **Spiral dataset.**

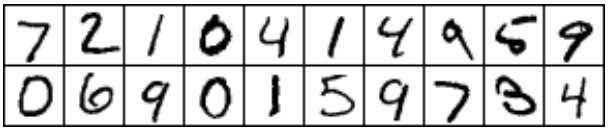

Figure 2: **MNIST digits sample.**

dataset [1]. MNIST is a database of handwritten digits (grayscale images of 28×28 pixels) containing $60,000$ training and $10,000$ test examples. See Figure 1.

2 Download

The sources can be downloaded from this link:

`https://cw.fel.cvut.cz/wiki/_media/courses/be4m33ssu/bp_src.zip`

The zip file contains a single Python3 source file `mlp.py`. Your task is to fill in or modify code at lines indicated by comments.

3 Task assignment

Note that you can be awarded bonus points for computationally efficient implementation. On the other hand, you may lose points for overly badly written code.

Assignment 1 (5 points + 2 bonus)

- *Implement all messages in* `LinearLayer`, `ReLULayer`, `SoftmaxLayer` *and* `LossCrossEntropy` *classes. Note that backward messages are not implemented in separate methods. It is more efficient to compute the* $\boldsymbol{\delta}^l$*'s (*`delta` *methods) directly as the Jacobian matrices representing the backward messages are often sparse. Similarly, there is no method dedicated to*

[1] `http://yann.lecun.com/exdb/mnist/`

compute the parameter message, instead, you will find `grad` *method computing the gradient of loss w.r.t. all layer parameters. Details are given in the corresponding comments.*

- *Implement the* `gradient` *method in* `MLP` *class performing the actual backpropagation.*
- *Train an MLP classifier on the spiral dataset using* `experiment_spiral` *as a base for your experiments. Use the MLP architecture as well as parameters given in the source file. The report will contain figures showing convergence for three settings of the learning rate:* $\alpha \in \{0.2, 1, 5\}$. *Discuss.*

Hint: You might want to use a simpler dataset such as XOR for debugging purposes. XOR contains four samples only while it is not linearly separable, which makes it an ideal candidate for early experiments. Use `experiment_XOR` *to get started.*
Hint: Use matrix operations such as `numpy.dot()` *wherever possible to make computations efficient. Use assertions to check correct tensor sizes and/or numerical problems (*`numpy.isnan()`, `numpy.isinf()` *or* `numpy.seterr()` *might help).*

Assignment 2 (3 points)

- *Give forward and backward messages for a compound layer composed of the softmax layer:*

$$p_k(\boldsymbol{s}) = \frac{e^{s_k}}{\sum_{c=1}^{K} e^{s_c}} \tag{3}$$

and multinominal cross-entropy loss:

$$\ell = -\sum_{k=1}^{K} t_k \log(p_k), \tag{4}$$

where $\boldsymbol{s} = (s_1, \ldots, s_K)$, $\boldsymbol{p} = (p_1, \ldots, p_K)$, $k \in \{1, \ldots, K\}$ *are softmax inputs and outputs,* $\boldsymbol{t} = (t_1, \ldots, t_K)$ *a one-hot encoded vector of targets and* ℓ *the value of loss for a single sample.*

- *Show that softmax is invariant to shift in inputs, i.e.,* $p_k(\boldsymbol{s}') = p_k(\boldsymbol{s})$ *where* $s'_k = s_k + c$ *for* $k \in \{1, \ldots, K\}$ *and* $c \in \mathbb{R}$.

Remark: To prevent overflows and underflows in softmax, $\max(\boldsymbol{s})$ *is often subtracted from all elements of its input* $\boldsymbol{s}$. *Subtracting* $\max(\boldsymbol{s})$ *makes all elements of* $\boldsymbol{s}$ *non-positive which prevents overflow. At least one element becomes zero which makes denominator greater or equal to one.*

Assignment 3 (4 points + 1 bonus)

- *Reimplement softmax according to the previous remark.*
- *Implement the compound layer (*`LossCrossEntropyForSoftmaxLogits` *class) from Assignment 2 and use it instead of separate softmax and loss layers.*
- *Train a deeper architecture as defined in* `experiments_MNIST` *for MNIST dataset.*
- *Give a plot showing the development of mean weight amplitude (mean absolute value) for each linear layer over training epochs. Show values normalized w.r.t. the initial mean amplitude. Discuss.*

Remark: Note that you should use the compound layer for training only. In the network evaluation phase (`MLP.propagate()`*) you should use standard softmax layer to get actual predictions. These two regimes are facilitated by* `output_layers` *parameter in* `MLP` *constructor as well as by the* `MLP.propagate()` *method's parameter of the same name.*
Remark: The size of the network and other parameters in `experiment_MNIST` *are chosen in order to cause numerical instability for the naïve implementation from Assignment 1. You should be able to get reasonable and even higher accuracy using much fewer neurons in the hidden layer still.*