IFT3395/6390 Fall 2017
Fondements de l'apprentissage machine

Professor : Aaron Courville
Teaching Assistants : Philippe Lacaille, Tegan Maharaj

Assignment 2 (Theoretical part) and Assignment 3 (Practical part)

Due date : November 30th, 2017

The goal of this assignment homework is to familiarize yourself with gradient backpropagation in neural networks. You will implement and experiment with a neural network - specifically, a Multilayer Perceptron (MLP) - for mutliclass classification. As a starting point, use the section on neural networks in the class materials and suggested readings. To simplify the task, the first part has already some solutions.

Instructions :

- This homework is to be done in teams of 2 or 3. Do not forget to put the names of all the members in the headings of the files and in the comments when you handout the file.
- We ask the format of the submitted homework to be in .pdf and all code should be in python using numpy and matplotlib.
- We strongly recommend your python code to be done in the format of ipython notebook like in the labs. To produce mathematical formulas, you can use LaTeX, LyX, or Word ; If you use ipython notebook you can directly put equations in it.
- When you submit your homework, it will only be accepted via StudiUM and only one file per team. If you have multiple files, an archive (zip, tar..) is accepted.

1 THEORETICAL PART A (20 pts) : derivatives and relations of some basic functions

Given

- logistic sigmoid $\text{sigmoid}(x) = \frac{1}{1+\exp(-x)}$.
- hyperbolic tangent $\tanh(x) = \frac{\exp(x)-\exp(-x)}{\exp(x)+\exp(-x)}$.
- softplus : $\text{softplus}(x) = \ln(1 + \exp(x))$
- $\ll$ sign $\gg$ function sign which returns +1 if its argument is positive, -1 if negative and 0 if null.
- $\mathbf{1}_S(x)$ is the indicator function which returns 1 if $x \in S$ (or x respects condition S) or 0.

— « rectifier » function which keeps only the positive part of its argument : $\text{rect}(x)$ returns x if $x \geq 0$ and returns 0 if $x < 0$. It is also named RELU (rectified linear unit) : $\text{rect}(x) = \text{RELU}(x) = [x]_+ = \max(0, x) = (\mathbf{1}_{\{x>0\}})x$

Answer the following questions :

1. Show that $\text{sigmoid}(x) = \frac{1}{2}\left(\tanh(\frac{1}{2}x) + 1\right)$
2. Show that $\ln \text{sigmoid}(x) = -\text{softplus}(-x)$
3. Show that the sigmoid derivative is : $\text{sigmoid}'(x) = \frac{\text{dsigmoid}}{\text{d}x}(x) = \text{sigmoid}(x)(1-\text{sigmoid}(x))$
4. Show that the tanh derivative is : $\tanh'(x) = 1 - \tanh^2(x)$
5. Write sign using only indicator functions : $\text{sign}(x) = \ldots$
6. Write the derivative of the absolute function $\text{abs}(x) = |x|$. Note : its derivative at 0 is not defined, but your function abs' can return 0 at 0. Note2 : use the function sign : $\text{abs}'(x) = ...$
7. Write the derivative of the function rect. Note : its derivative at 0 is undefined, but your function can return 0 at 0. Note2 : use the indicator function. $\text{rect}'(x) = ...$
8. Let the norm L_2 of a vector be : $\|\mathbf{x}\|_2^2 = \sum_i \mathbf{x}_i^2$. Write the vector of the gradient : $\frac{\partial \|\mathbf{x}\|_2^2}{\partial \mathbf{x}} = \ldots$
9. Let the norm L_1 of a vector be : $\|\mathbf{x}\|_1 = \sum_i |\mathbf{x}_i|$. Write the vector of the gradient : $\frac{\partial \|\mathbf{x}\|_1}{\partial \mathbf{x}} = \ldots$

2 THEORETICAL PART B (40 pts) : Gradient computation for parameters optimization in a neural net for multi-class classification

Let $D_n = \{(\mathbf{x}^{(1)}, y^{(1)}), \ldots, (\mathbf{x}^{(n)}, y^{(n)})\}$ be the dataset with $x^{(i)} \in \mathbb{R}^d$ and $y^{(i)} \in \{1, \ldots, m\}$ indicating the class within m classes. In the following equations, vectors are by default considered to be column vectors.

Consider a *Multilayer perceptron* with only one hidden layer (meaning 3 layers total if we count the input and output layers). The hidden layer is has d_h neurons, fully connected to the input and output layers. We shall consider a **rectifier** (Rectified Linear Unit or **RELU**) non-linearity (activation function) for the hidden layer. The output layer has m neurons, equipped with a **softmax** non-linearity. The output of the j^{th} neuron of the output layer gives a score for the class j which is interpreted as the probability of x being of class j.

It is highly recommended that you draw the neural net as it helps understanding all the steps.

1. Let $\mathbf{W}^{(1)}$, a $d_h \times d$ matrix of weights, and $\mathbf{b}^{(1)}$, the bias vector, be the connections between the input layer and the hidden layer. What is the dimension of $\mathbf{b}^{(1)}$? Give the formula of the preactivation vector (before

the non linearity) of the neurons of the hidden layer $\mathbf{h}^a$ given $\mathbf{x}$ as input. Write this first in a matrix form ($h^a = \ldots$), and then detail how to compute one element $\mathbf{h}^a_j = \ldots$. Write the output vector of the hidden layer $\mathbf{h}^s$ with respect to $\mathbf{h}^a$.

2. Let $\mathbf{W}^{(2)}$ a weight matrix and $\mathbf{b}^{(2)}$a bias vector be the connections between the hidden layer and the output layer. What are the dimensions of $\mathbf{W}^{(2)}$ and $\mathbf{b}^{(2)}$? Give the formula of the activation function of the neurons of the output layer $\mathbf{o}^a$ with respect to their input $\mathbf{h}^s$ in a matrix form and then write in a detailed form for $\mathbf{o}^a_k$.

3. The output of the neurons at the output layer is given by :

$$\mathbf{o}^s \quad = \quad \text{softmax}(\mathbf{o}^a)$$

Give the precise equation for $\mathbf{o}^s_k$ using the softmax (formula with the exp). ***Show*** that the $\mathbf{o}^s_k$ are positive and sum to 1. Why is this important ?

4. The neural net computes, for an input vector $\mathbf{x}$, a vector of probability scores $\mathbf{o}^s(\mathbf{x})$. The probability, computed by a neural net, that an observation $\mathbf{x}$ belongs to class y is given by the y^{th} output $\mathbf{o}^s_y(\mathbf{x})$. This suggests a loss function such as :

$$L(\mathbf{x}, y) = \text{-log } \mathbf{o}^s_y(\mathbf{x})$$

Find the equation of L as a function of the vector $\mathbf{o}^a$. It is easily achievable with the correct substitution using the equation of the previous question.

5. Training of the neural net consists in finding the parameters that minimize the empirical risk $\hat{R}$ associated to this loss function. What is $\hat{R}$? What is precisely the set θ of parameters of the network ? How many scalar parameters n_θ are there ? Write down the optimization problem of training the network in order to find the optimal values for these parameters.

6. To find a solution to this optimization problem, we will use the technique called gradient descent. What is the (batch) gradient descent equation for this problem ?

7. We can compute the vector of the gradient of the empirical risk $\hat{R}$ with respect to the parameters set θ this way

$$\begin{pmatrix} \frac{\partial \hat{R}}{\partial \theta_1} \\ \vdots \\ \frac{\partial \hat{R}}{\partial \theta_{n_\theta}} \end{pmatrix} \quad = \quad \frac{1}{n} \sum_{i=1}^{n} \begin{pmatrix} \frac{\partial L(\mathbf{x}_i, y_i)}{\partial \theta_1} \\ \vdots \\ \frac{\partial L(\mathbf{x}_i, y_i)}{\partial \theta_{n_\theta}} \end{pmatrix}$$

This hints that we only need to know how to compute the gradient of the loss L with an example $(\mathbf{x}, y)$ with respect to the parameters, defined as followed :

$$\frac{\partial L}{\partial \theta} = \begin{pmatrix} \frac{\partial L}{\partial \theta_1} \\ \vdots \\ \frac{\partial L}{\partial \theta_{n_\theta}} \end{pmatrix} = \begin{pmatrix} \frac{\partial L(\mathbf{x}, y)}{\partial \theta_1} \\ \vdots \\ \frac{\partial L(\mathbf{x}, y)}{\partial \theta_{n_\theta}} \end{pmatrix}$$

We shall use **gradient backpropagation**, starting with loss L and going to the output layer $\mathbf{o}$ then down the hidden layer $\mathbf{h}$ then finnaly at the input layer $\mathbf{x}$.

Show that

$$\frac{\partial L}{\partial \mathbf{o}^a} = \mathbf{o}^s - \text{onehot}_m(y)$$

Note : Start from the expression of L as a function of $\mathbf{o}^a$ that you previously found. Start by computing $\frac{\partial L}{\partial \mathbf{o}^a_k}$ for $k \neq y$ (using the start of the expression of the logarithm derivative). Do the same thing for $\frac{\partial L}{\partial \mathbf{o}^a_y}$.

8. What is the numpy equivalent expression (it can fit in 2 operations) ?

   grad_oa = ...

   ...

   IMPORTANT : From now on when we ask to "compute" the gradients or partial derivatives, you only need to write them as function of previously computed derivatives (**do not substitute the whole expressions already computed in the previous questions !**)

9. Compute the gradients with respect to parameters $\mathbf{W}^{(2)}$ and $\mathbf{b}^{(2)}$ of the output layer. Since L depends on $\mathbf{W}^{(2)}_{kj}$ and $\mathbf{b}^{(2)}_k$ only through $\mathbf{o}^a_k$ the result of the chain rule is :

$$\frac{\partial L}{\partial \mathbf{W}^{(2)}_{kj}} = \frac{\partial L}{\partial \mathbf{o}^a_k} \frac{\partial \mathbf{o}^a_k}{\partial \mathbf{W}^{(2)}_{kj}}$$

   and

$$\frac{\partial L}{\partial \mathbf{b}^{(2)}_k} = \frac{\partial L}{\partial \mathbf{o}^a_k} \frac{\partial \mathbf{o}^a_k}{\partial \mathbf{b}^{(2)}_k}$$

10. Write down the gradient of the last question in matrix form and define the dimensions of all matrix or vectors involved.

    Take time to understand why the above equalities are the same as the equations of the last question.

    Give the numpy form :

    grad_b2 = ...

    grad_W2 = ...

11. What is the partial derivative of the loss L with respect to the output of the neurons at the hidden layer ? Since L depends on $\mathbf{h}^s_j$ only through the activations of the output neurons $\mathbf{o}^a$ the chain rule yields :

$$\frac{\partial L}{\partial \mathbf{h}^s_j} = \sum_{k=1}^{m} \frac{\partial L}{\partial \mathbf{o}^a_k} \frac{\partial \mathbf{o}^a_k}{\partial \mathbf{h}^s_j}$$

12. Write down the gradient of the last question in matrix form and define the dimensions of all matrix or vectors involved.
    (What are the dimensions ?)
    Take time to understand why the above equalities are the same as the equations of the last question.
    Give the numpy form :
    grad_hs = ...
13. What is the partial derivative of the loss L with respect to the activation of the neurons at the hidden layer ? Since L depends on the activation $\mathbf{h}_j^a$ only through $\mathbf{h}_j^s$ of this neuron, the chain rule gives :

$$\frac{\partial L}{\partial \mathbf{h}_j^a} = \frac{\partial L}{\partial \mathbf{h}_j^s} \frac{\partial \mathbf{h}_j^s}{\partial \mathbf{h}_j^a}$$

    Note $\mathbf{h}_j^s = \text{rect}(\mathbf{h}_j^a)$: the rectifier function is applied elementwise. Start by writing the derivative of the rectifier function $\frac{\partial \text{rect}(z)}{\partial z} = \text{rect}'(z) = \ldots$.
14. Write down the gradient of the last question in matrix form and define the dimensions of all matrix or vectors involved. Give the numpy form.
15. What is the gradient with respect to the parameters $\mathbf{W}^{(1)}$ and $\mathbf{b}^{(1)}$ of the hidden layer ?
    Note : same logic as a previous question
16. Write down the gradient of the last question in matrix form and define the dimensions of all matrix or vectors involved. Give the numpy form.
    Note : same logic as a previous question
17. What are the partial derivatives of the loss L with respect to $\mathbf{x}$?
    Note : same logic as a previous question
18. We will now consider a **regularized** empirical risk : $\tilde{R} = \hat{R} + \mathcal{L}(\theta)$, where θ is the vector of all the parameters in the network and $\mathcal{L}(\theta)$ describes a scalar penalty as a function of the parameters θ. The penalty is given importance according to a prior preferences for the values of θ. The L_2 (quadratic) regularization that penalizes the square norm (norm L_2) of the weights (but not the biases) is more standard, is used in ridge regression and is sometimes called "weight-decay". Here we shall consider a double regularization L_2 and L_1 which is sometimes named "elastic net" and we will use different **hyperparameters** (positive scalars$\lambda_{11}.\lambda_{12}.\lambda_{21}.\lambda_{22}$) to control the effect of the regularization at each layer

$$\begin{aligned}
\mathcal{L}(\theta) &= \mathcal{L}(\mathbf{W}^{(1)}, \mathbf{b}^{(1)}, \mathbf{W}^{(2)}, \mathbf{b}^{(2)}) \\
&= \lambda_{11}\|\mathbf{W}^{(1)}\|_1 + \lambda_{12}\|\mathbf{W}^{(1)}\|_2^2 + \lambda_{21}\|\mathbf{W}^{(2)}\|_1 + \lambda_{22}\|\mathbf{W}^{(2)}\|_2^2 \\
&= \lambda_{11}\left(\sum_{i,j} |\mathbf{W}_{ij}^{(1)}|\right) + \lambda_{12}\left(\sum_{ij} (\mathbf{W}_{ij}^{(1)})^2\right) \\
&\quad + \lambda_{21}\left(\sum_{i,j} |\mathbf{W}_{ij}^{(2)}|\right) + \lambda_{22}\left(\sum_{ij} (\mathbf{W}_{ij}^{(2)})^2\right)
\end{aligned}$$

We will in fact minimize the regularized risk $\tilde{R}$ instead of $\hat{R}$. How does this change the gradient with respect to the different parameters ?

3 PRACTICAL PART (40 pts) : Neural network implementation and experiments

We ask you to implement a neural network where you compute the gradients using the formulas derived in the previous part (including elastic net type regularizations). You must not use an existing neural network library, but you must use the derivation of part 2 (with corresponding variable names, etc). Note that you can reuse the general learning algorithm structure that we used in the demos, as well as the functions used to plot the decision functions.

Useful details on implementation :

- **Numerically stable softmax**. You will need to compute a numerically stable softmax. Refer to posted readings to see how to do this. Start by writing the expression for a single vector, then adapt it for a mini-batch of examples stored in a matrix.
- **Parameter initialization.** As you know, it is necessary to randomly initialize the parameters of your neural network (trying to avoid symmetry and saturating neurons, and ideally so that the pre-activation lies in the bending region of the activation function so that the overall networks acts as a non linear function). We suggest that you sample the weights of a layer from a uniform distribution in $\left[\frac{-1}{\sqrt{n_c}}, \frac{1}{\sqrt{n_c}}\right]$, where n_c is the number of inputs for **this layer** (changing from one layer to the other). *Biases* can be initialized at 0. Justify any other initialization method.
- **fprop** and **bprop**. We suggest you implement methods `fprop` and `bprop`. `fprop` will compute the forward progpagation i.e. step by step computation from the input to the output and the cost, of the activations of each layer. `bprop` will use the computed activations by `fprop` and does the backpropagation of the gradients from the cost to the input following precisely the steps derived in part 2.
- **Finite difference gradient check.** We can estimate the gradient numerically using the finite difference method. You will implement this estimate as a tool to check your gradient computation. To do so, calculate the value of the loss function for the current parameter values (for a single example or a mini batch). Then for each scalar parameter θ_k, change the parameter value by adding a small perturbation ϵ ($10^{-6} < \varepsilon < 10^{-4}$) and calculate the new value of the loss (same example or minibatch), then set the value of the parameter back to its original value. The partial derivative with respect to this parameter is estimated by dividing the change in the loss function by ε. The ratio of your gradient computed by backpropagation and your estimate using finite difference should be between 0.99 and 1.01.
- **Size of the mini batches**. We ask that your computation and gradient

descent is done on minibatches (as opposed to the whole training set) with adjustable size using a hyperparameter K. In the minibatch case, we do not manipulate a single input vector, but rather a batch of input vectors grouped in a matrix (that will give a matrix representation at each layer, and for the input). In the case where the size is one, we obtain an equivalent to the stochastic gradient. Given that numpy is efficient on matrix operations, it is more efficient to perform computations on a whole minibatch. It will greatly impact the execution time.

Experiments : We will use the 2 moons dataset and the task of classifying written digits MNIST (see links on course website).

1. As a beginning, start with an implementation that computes the gradients for **a single** example, and check that the gradient is correct using the finite difference method described above.
2. Display the gradients for both methods (direct computation and finite difference) for a small network (e.g. $d = 2$ and $d_h = 2$) with random weights and for a single example.
3. Add a hyperparameter for the minibatch size K to allow compute the gradients on a minibatch of K examples (in a matrix), by **looping** over the K examples (this is a small addition to your previous code).
4. Display the gradients for both methods (direct computation and finite difference) for a small network (e.g. $d = 2$ and $d_h = 2$) with random weights and for a minibatch with 10 examples (you can use examples from both classes from the dataset 2 moons).
5. Train your neural network using gradient descent on the dataset of the two moons. Plot the decision regions for several different values of the hyperparameters (weight decay, number of hidden units, early stopping) so as to illustrate their effect on the capacity of the model.
6. As a second step, copy your existing implementation to modify it to a new implementation that will use matrix calculus (instead of a loop) on batches of size K to improve efficiency. **Take the matrix expressions in numpy derived in the first part, and adapt them for a minibatch of size K. Show in your report what you have modified (precise the former and new expressions with the shapes of each matrices).**
7. Compare both implementations (with a loop and with matrix calculus) to check that they both give the same values for the gradients on the parameters, first for $K = 1$, then for $K = 10$. Display the gradients for both methods.
8. Time how long takes an epoch on MNIST (1 epoch = 1 full traversal through the whole training set) for $K = 100$ for both versions (loop over a minibatch and matrix calculus).
9. Adapt your code to compute the error (proportion of misclassified examples) on the training set as well as the total loss on the training set during each epoch of the training procedure, and at the end of each epoch, it computes

the error and average loss on the validation set and the test set. Display the 6 corresponding figures (error and average loss on train/valid/test), and write them in a log file.

10. Train your network on the MNIST dataset. Plot the training/valid/test curves (error and loss as a function of the epoch number, corresponding to what you wrote in a file in the last question). Include in your report the curves obtained using your best hyperparameters, i.e. for which you obtained your best error on the validation set. We suggest 2 plots : the first one will plot the error rate (train/valid/test with different colors, precise which color in a legend) and the other one for the averaged loss (on train/valid/test). You should be able to get less than 5% test error. Indicate the values of your best hyperparameters corresponding to the curves. Bonus points are given for a test error of less that 2%.