TDT4171 Artificial Intelligence Methods

Exercise 5

March 22, 2017

- **Delivery deadline: Wednesday April 19th** by 8:00 am.
- Required reading for this assignment: Chapter 18.6 and 18.7 – it is a good idea to have a basic understanding of how the learning algorithms using gradient descent works before beginning this exercise.
- Deliver your solution on *It's Learning*
- Students may work alone.
- Homeworks can be answered in English or in Norwegian.
- This homework counts for 3% of the final grade.
- The homework is graded on a pass/fail basis. A pass grade will only be given when a decent attempt has been made to solve **all** parts of the exercise.
- Cribbing from other students (koking) is not accepted, and if detected will lead to the assignment being failed.

Exercise: Gradient Descent (GD) and Perceptron Algorithm

In this exercise we will explore and implement gradient descent algorithm and perceptron learning. You can implement this in any language you want, but official help will only be available for implementation in Python, Java and C++. For this assignment you should consult Lecture 9 slides and Chapter 18.6 and 18.7 on the book.

I Gradient descent

In this assignment you are suppose to implement gradient descent algorithm. Gradient descent algorithm is central in many learning algorithms, since in many learning paradigms the central "trick" consist in transforming the learning problem into an optimization problem. Generally this might accomplished by defining a loss function that measures how "wrong" is your model, and them optimizing the parameters of your model to decrease the loss. Since in many

situation it is not possible to analytically optimize this loss, gradient descent methods are suitable for this step. In Neural Networks this training technique is very common and appropriate. In this exercise you will start by applying Gradient Descent to a given loss, and later generalize this by writing a training algorithm for the perceptron using gradient descent.

Given a weight vector $w \in \mathbb{R}^d$ and a data point $x \in \mathbb{R}^d$, we define the standard logistic function with these parameters in Eq. 1. This function is useful to map the inner product $w^T x$ to the range [0,1]. Given this mapping we can define a rule for a 2 class classification problem (basically if $\sigma(w, x) > 0.5$, x is assigned to class 1, and if $\sigma(w, x) \leq 0.5$, x is assigned to class 0).

$$\sigma(w, x) = \frac{1}{1 + e^{-w^T x}} \tag{1}$$

Consider the following loss function for $w \in \mathbb{R}^2$:

$$L_{simple}(w) = [\sigma(w, [1, 0]) - 1]^2 + [\sigma(w, [0, 1])]^2 + [\sigma(w, [1, 1]) - 1]^2 \tag{2}$$

I.1 Tasks

1. Plotting: make a grid $\{w = (w_1, w_2), w_1 \in [-6, 6], w_2 \in [-6, 6]\}$ and plot the values of $L_{simple}(w)$ (Matlab, Octave or matplotlib in python should make this easy for you). By inspecting the graph which value of w is minimizing the loss $L_{simple}(w)$? And what is the minimum of $L_{simple}(w)$?
2. Calculate the formula for the derivative $\nabla_w L_{simple}(w) = [\frac{\partial L_{simple}(w)}{w_1}, \frac{\partial L_{simple}(w)}{w_2}]$ (Eq. 2)
3. Apply the Gradient Descent update rule to iteratively minimize $L_{simple}(w)$ and compare your numerical results with the graphical results from item 1.
   - Update rule: $w_i^{\text{new}} \leftarrow w_i^{\text{old}} - \eta \frac{\partial L_{simple}(w)}{w_i}|_{w=w^{\text{old}}}$, with $\eta > 0$
   - Implement a function parametrized by η and number of iterations and return the parameter w that minimizes L_{simple}. For a fixed number of iterations show how different values of η impact the GD algorithm (in particular at least try $\eta = 0.0001, 0.01, 0.1, 1, 10, 100$). You should write at least one paragraph discussing your findings (for example discussing that for some values the algorithm might not converge, or it might be very slow to converge).
   - Plot $L_{simple}(w)$ for w found by Gradient Descent for different values of η.

II Perceptron algorithm with logistic function

In this section you will implement a simple Perceptron for binary classification using logistic function. In lecture 9 slides (specific slide 21) there is a skeleton

algorithm for Gradient Descent with the famous delta rule for the Perceptron, we will repeat here and make adjustments in order to use the logistic function (Eq. 1).

Given the training dataset $D_{\text{train}} = \{(x_n, y_n), n = 1, \cdots, N\}$, datapoints $x_n \in \mathbb{R}^d$ and respective class target $y_n \in \{0, 1\}$, we want to use a Perceptron to predict the labels the testing dataset $D_{\text{test}} = \{(x_n, y_n), n = 1, \cdots, N\}$ (that is not observed at training time). For that matter we will linearly divide the datapoints space in two different regions using a weight vector parametrizing a hyperplane $w^T x + b$ and mapping the result to [0,1] using the logistic function $\sigma(w, x)$. For convenience we will augment the datapoint $x \in \mathbb{R}^d$ with a constant and weight vector $w \in \mathbb{R}^d$ with the bias term, so $x' \in \mathbb{R}^{d+1} = [1, x_1, \cdots, x_d]$ and $w' \in \mathbb{R}^{d+1} = [b, w_1, \cdots, w_d]$, so $w'^T x' = w^T x + b$. From now on when describing the details and the algorithms have in mind that we are working with the augmented data structure (if you don't augment the data structure you will not be able to represent the bias term in the hyperplane equation, so you will be restricted to rotating your hyperplane around the origin). In the output of the Perceptron, given the weights w, the decision rule to classify is:

$$\text{classify}(w, x) = \begin{cases} 0 & \text{if } \sigma(w', x') \leq 0.5 \\ 1 & \text{if } \sigma(w', x') > 0.5 \end{cases} \tag{3}$$

The loss function for each datapoint is the euclidean distance between the predicted value for x_n and the target y_n

$$L_n(w, x_n, y_n) = \frac{1}{2}[\sigma(w, x_n) - y_n]^2 \tag{4}$$

One possible loss function for the complete dataset is the expected value of the loss function for each individual item, or the average loss over the dataset

$$L(w, D) = \frac{1}{N} \sum_{(x_n, y_n) \in D} L_n(w, x_n, y_n) = \frac{1}{N} \sum_{(x_n, y_n) \in D} \frac{1}{2}[\sigma(w, x_n) - y_n]^2 \tag{5}$$

Now we are ready to enunciate two versions of the Perceptron learning algorithm (**Batch Gradient Descent** and **Stochastic Gradient Descent**) :

- **BatchGradientDescent(dataset $D_{\text{train}} = \{(x_n, y_n), x_n \in \mathbb{R}^{d+1}, y_n \in \{0, 1\}\}$, learning rate η, number of iterations T)**
  1. $w \in \mathbb{R}^{d+1} \leftarrow$ random initialization of linear weight units
  2. for all $t \in \{1, \cdots, T\}$
     (a) for all $i \in \{1, \cdots, d+1\}$
        i. $\text{grad}_i \leftarrow 0$
        ii. for all $(x_n, y_n) \in D_{\text{train}}$
           A. $\text{grad}_i \leftarrow \text{grad}_i + \frac{\partial L_n(w, x_n, y_n)}{\partial w_i}$
        iii. $w_i \leftarrow w_i - \frac{1}{N}\eta \cdot \text{grad}_i$

3. return w

- **StochasticGradientDescent(dataset $D_{\text{train}} = \{(x_n, y_n), x_n \in \mathbb{R}^{d+1}, y_n \in \{0, 1\}\}$, learning rate η, number of iterations T)**
  1. $w \in \mathbb{R}^{d+1} \leftarrow$ random initialization of linear weight units
  2. for all $t \in \{1, \cdots, T\}$
     (a) $(x^*, y^*) \leftarrow$ random select point $(x_n, y_n) \in D_{\text{train}}$
     (b) for all $i \in \{1, \cdots, d+1\}$
        i. $\text{grad}_i \leftarrow \frac{\partial L_n(w, x^*, y^*)}{\partial w_i}$
        ii. $w_i \leftarrow w_i - \eta \cdot \text{grad}_i$
  3. return w

Notice that the difference between the batch version and the stochastic version is that in the first one each gradient update step for the weight is taking into account the loss over the entire dataset $L(w, D)$ (Eq. 5), while the second one each time the algorithm sees a datapoint it calculates the loss for this single point and update the weight vector $L_n(w, x_n, y_n)$ (Eq. 4). SGD is a more scalable version of the gradient algorithm, but it may be less stable than the batch version.

II.1 Tasks

1. Calculate the final formula for the derivative $\frac{\partial L_n(w, x_n, y_n)}{\partial w_i} = [y_n - \sigma(w, x_n)] \cdot \frac{\partial \sigma(w, x_n)}{\partial w_i}$

   (you will need to calculate this derivative: $\frac{\partial e^{-w^T x}}{\partial w_i} = \frac{\partial e^{-\sum_i w_i x_i}}{\partial w_i} = -x_i \cdot e^{-\sum_i w_i x_i}$)

2. Knowing the formula for $\frac{\partial L_n(w, x_n, y_n)}{\partial w_i}$, fill the gaps in the batch and stochastic algorithm, implement and test them. Now you should be able to calculate grad_i and train your Perceptron with logistic function.[1]

3. Experiments:
   - I will provide you a python file with a skeleton of the methods, it is not a running implementation, but rather an implementation with the basic structure that you can build upon.
   - Datasets: `data.zip` list of files:

     ```
     data_big_nonsep_test.csv  data_big_nonsep_train.csv
     data_big_separable_test.csv  data_big_separable_train.csv
     data_small_nonsep_test.csv  data_small_nonsep_train.csv
     ```

[1] if you didn't succeed in showing the math behind this derivative, you may consult slide 22 from lecture 9 slides`https://ntnu.itslearning.com/ContentArea/ContentArea.aspx?LocationType=1&LocationID=64752&ElementID=3258593&ElementType=64`, it is not the exact same formula we are looking for – there is one summation extra in the slides, because it is the formula valid only for batch GD– but it help you get the right result

`data_small_separable_test.csv` `data_small_separable_train.csv`

- Datasets: You have 4 datasets, each split in training and testing files. Each file has a number of lines with 3 numbers, each representing (x_n, y_n) datapoints, x_n is 2-dimensional and y_n is 0 or 1. The amount of training examples for each class is the same (no problem of class imbalance), and the test set size is 10% the size of the training set. There are two datasets with 11000 points and two with 2200 points (suffix `big` and `small`). The separable (suffix `separable`) datasets should be easier to train, resulting for the same number of iterations, in less errors than in the non-separable datasets (suffix `nonsep`) .
- For each dataset train a Perceptron using both batch and stochastic gradient descent. Keep track of the following: training time, average error (number of errors over total number of testing samples). What is the general trend? Which dataset is more complex to train (takes more time and have more errors)? Discuss differences in performance between batch and stochastic gradient descent using the results on each dataset as support.
- Choose one dataset and make a scatter plot the training points, coloring each class with a different color. Now run your trained Perceptron over the testing dataset and plot in the same graph the testing point coloring each class your diferent colors (use different color for training and testing). This way you should visualize for good your model
- Choose one dataset and one training algorithm (batch or stochastic) and run multiple times training and testing varying the number of iterations (you should at least try to following number of iterations $T = 10, 20, 50, 100, 200, 500, 1000, 2000$), keeping track of the running time and average error for each. Plot the results and discuss your findings (what is the general trend and your explanation for it).

III Deliverables

- Report (preferably PDF) with all the results, derivations, graphs and discussions from previous section;
- Code necessary to run your experiments.
- Zip all the code, **don't** zip the PDF report.