CS6375 Machine Learning

Assignment 1

Regression using Enhanced Gradient Descent

Author:
Randy Suarez Rodes

Supervisor:
Anurag Nagar Ph.D.

September 15, 2020

Contents

1 Part 1

1.1 Enhancement

We have selected the AMSGrad algorithm as our enhancement for the batch Gradient Descent algorithm. AMSGrad is a variant of the Adam algorithm and as Adam, it stores an exponentially decaying average of past gradients (m_t) and past squared gradients (v_t). Different than Adam, it ensures to use the maximum of past squared gradients v_t. Also for simplicity, the authors of the algorithm removed the debiasing step of Adam. (see towardsdatascience and ruder.io/optimizing-gradient-descent)

The process of updates of AMSGrad is as follows (t represents time stamp of each iteration):

$$m_t = \beta_1 m_{t-1} + (1-\beta_1)\frac{\partial L}{\partial w_t}$$

$$v_t = \beta_2 v_{t-1} + (1-\beta_2)\left(\frac{\partial L}{\partial w_t}\right)^2$$

$$\hat{v}_t = max(\hat{v}_{t-1}, v_t)$$

$$w_{t+1} = w_t - \frac{\alpha}{\sqrt{(\hat{v}_t)} + \epsilon} m_t$$

where m and v are initialized as zero. And L is the squared-loss function:

$$L = \frac{1}{2 \cdot n_{samples}} \sum_{i=1}^{n_{samples}} (y_i - \hat{y}_i)^2$$

where y_i is output and $\hat{y}_i$ is the estimate obtained from the linear model. ϵ is a small floating point value to ensure that we will never divide by zero. β_1 and β_2 are constants with common default values of 0.9 and 0.999.

Up next we present the pseudo-code of our algorithm. We implemented a single batch vectorized version of AMSGrad. The data observations X and outputs Y are omitted for simplicity. Our algorithm requires to include a column of biases of 1 to be inserted as the first column of X:

```
Data: initial weights: $w$, initial learning rate: $\alpha$, number of
      iterations: $it$, tolerance: $tol$
Result: Historical MSE: $hist\_error$, Total iterations: $total\_iter$,
        Updated weights $w$
$\epsilon \leftarrow 10^{-7}$, $\beta_1 \leftarrow 0.9$, $\beta_2 \leftarrow 0.999$
$m \leftarrow 0$ vector, $v \leftarrow 0$ vector, $v_0 \leftarrow 0$ vector
$total\_iter \leftarrow it$, $least\_error \leftarrow MSE$ w.r.t current $w$
add $least\_error$ to the list $hist\_error$
for $i$ = 1 to it do
    // compute Gradient w.r.t. current $w$
        $$\frac{\partial L}{\partial w}$$
        // update $m$ and $v$
        $$m \leftarrow \beta_1 m + (1-\beta_1)\frac{\partial L}{\partial w}$$
        $$v \leftarrow \beta_2 v_0 + (1-\beta_2)\left(\frac{\partial L}{\partial w}\right)^2$$
        // update $v$ as max of previous and current $v$
        (point-wise maximum)
        $$v \leftarrow max(v, v_0)$$
    // update the weights $w$
        $$w \leftarrow w - \frac{\alpha}{\sqrt(\hat{v}) + \epsilon} m$$
    $curr\_error \leftarrow MSE$ w.r.t. updated $w$
    add $curr\_error$ to the list $hist\_error$
    if stopping condition then
        update $total\_iter$
        break for
    end
    if $current\_error < least\_error$ then
        $least\_error \leftarrow current\_error$
    end
end
return $hist\_error$, $total\_iter$, $w$
```

Our *stopping condition* is: $abs(least\_error - curr\_error) < tol$ which guarantees our algorithm will stop if the $current\_error$ does not improve by a value less than the fixed tolerance in relation to the best historical error so far.

1.2 Data Description

We have selected for our assignment the data set Seoul Bike Sharing Demand.

Data Information (source: UCI Machine Learning Repository website)

The dataset contains weather information (Temperature, Humidity, Windspeed, Visibility, Dewpoint Temperature, Solar radiation, Snowfall and Rainfall) and the number of bikes rented per hour and date information. The target variable is the number of bikes rented.

Attributes information and type of variable:

- Date : year-month-day (date)
- Rented Bike count - Count of bikes rented at each hour (integer)
- Hour - Hour of he day (integer)
- Temperature-Temperature in Celsius (continuous)
- Humidity - % (continuous)
- Windspeed - m/s (continuous)
- Visibility - 10m (continuous)
- Dew point temperature - Celsius (continuous)
- Solar radiation - MJ/m2 (continuous)
- Rainfall - mm (continuous)
- Snowfall - cm (continuous)
- Seasons - Winter, Spring, Summer, Autumn (categorical)
- Holiday - Holiday/No holiday (categorical)
- Functional Day - NoFunc(Non Functional Hours), Fun(Functional hours) (categorical)

1.3 Data Preprocessing

We first proceed to clean our data. No NA values or redundant rows were present in the data.

Second, we have to encode the categorical variables so we are able to find the regression coefficients. Variable Date cannot be simply encode. One possible solution would be assigning a day of the week to each date and then proceed to one hot encode the day of the week variable, but this approach will add more variables to our possible model, therefore, for convenience we will not include this variable in our analysis.

We carried out the process of one hot encoding the variables Seasons, Holiday and Functional day using the pandas function $get\_dummies()$ with the option of $drop\_first = True$, which basically means that one of the categories is dropped and encoded as a non activation of the others (example if we have Spring=0, Summer=0 and Winter=0 this is equivalent to Autumn=1). By using this option we reduce the number of variables added to our data.

For simplicity we rename the columns of our data.

After this transformations we can analyse the correlations between variables. (see logfile page 1).

Notice the attributes Temperature and Dew point temperature are highly correlated. Of these two, Temperature provides more information which respect to the response variable, so we will proceed to drop Dew point temperature. Also several continuous attributes present low correlation with the target variable (Humidity, Wind speed, Visibility, Rainfall, Snowfall, all with a correlation below .2 in absolute value), we will exclude these variables from our analysis. Additionally we will exclude the categorical variable Holiday due to its low correlation (0.07).

By taking a look at a summary of our data we can appreciate how different are the scales of our attributes, so we will proceed to normalize the training set and carry out this same transformation to the testing set.

Finally we need to encode the variable Hour. To capture the cyclical behaviour of this variable we applied a sin-cosine transformation (see this references: kaggle; ianlondon.github.io/blog) We will add two variables named *sin_hour* and *cos_hour* as the result of this transformation and we will drop the variable $Hour$ from our data set.

Now we are in conditions of defining our model:

$$\begin{aligned} \text{Bike}\hat{\text{C}}\text{ount} = w_0 &+ w_1\text{Temperature} \\ &+ w_2\text{Solar\_Rad} \\ &+ w_3\text{Spring} \\ &+ w_4\text{Summer} \\ &+ w_5\text{Winter} \\ &+ w_6\text{Funct\_Day} \\ &+ w_7\text{sin\_hour} \\ &+ w_8\text{cos\_hour} \end{aligned}$$

1.4 Dataset Split

We have split our data into 80/20 proportion of training/testing set.

As we mentioned before we normalized our data due to the presence of different scales of our attributes. The normalization process is only applied to the continuous attributes since the normalization of the categorical variables would totally change the meaning of these variables.

1.5 Tuning Process

For our tuning process, we generated random normal(0,1) initial weights and we ran multiple combinations of learning rates and iterations.

We started by exploring possible values of learning rates between 0.000001 and 1 for a fixed amount of 200 iterations and track the historical MSE over this numbers of iterations. (See logfile pages 2,3,4). We can appreciate that as the learning rate grows AMSGrad starts improving and remains stable even when the learning rate is 1 where the vanilla gradient descent explodes.

The final MSE for each one of the learning rates can be found on page 5. Once again AMSGrad remained stable where vanilla did not.

Therefore we proceed to extend the exploration of possible initial learning rates for AMSGrad to a wider range [1-100]. The graphs of pages 6, 7 and

8 reflect the behaviour of historical MSE over 200 iterations and on page 9 we present the final MSE of this process. We can appreciate the stability of AMSGrad and how it shows improvements on learning rates approximately greater than 40.

After this analysis we ran tuning combinations again, this time on more specific ranges as concluded from our past analysis; [0.05-0.95] for both algorithms and additionally [40-120] for AMSGrad. The results of our tuning process is reflected on the table of page 10. We presented the best learning parameters sorted from the least MSE. We can see that AMSGrad outperforms the Vanilla GD with multiple learning rates and in a lesser amount of iterations.

1.6 Training and Testing set evaluation

With the best parameters of this process we can proceed to stablish comparisons in both training and testing sets.

The training set results are reflected on the table of page 13 along with the results of the algorithm of part 2. We also added the analytical coefficients metrics. Comparing the metrics of our algorithms and the results of the analytical coefficients we can say that our algorithms converged to the best solution. The coefficient of determination value is around 0.48. This value can be improved by building a more robust model (for example including interactions), although this was not the goal of this project and we only focused on the task on hand, implementing an enhancement and prove that works. We are satisfied with this result since we have obtained a great approximation in so few iterations. Also we are satisfied of the results thrown by a broad tuning process.

The predictions metrics results for the testing set are reflected on the table of page 14 also with the results from part 2. We can see that AMSGrad generalized the results better. than Vanilla.

2 Part 2

First of all, we need to select a library from the Scikit Learn linear regression package. On Scikit Learn there is not an exact algorithm to match the AMSGrad algorithm, hence we decided to use SGDRegressor and configure its options as closest as possible to match our algorithms(see comments in the code). We could have used MLPRegressor which has an 'adam' solver, but this algorithm is implemented as a Multi-layer Perceptron regressor optimizer with the minimum number of hidden layers accepted as 1. Therefore MLPRegressor would not end with a linear fit that we can use to compare with our algorithms.

2.1 Tuning Process

We followed a similar work flow as in part 1. Given the same initial random normal(0,1) initial weights, we ran multiple combinations of learning rates and iterations. We first started looking for learning rates in the range of 0.00001 and 1 finding that as learning rates grow, the algorithm gets worse. (See logfile page 11).

With this new knowledge we ran new combinations of learning rates (between 0.0001 and 0.1) and iterations, the results of this tuning process are reflected on the table from page 12.

2.2 Comparisons

With all algorithms best parameters determined, we can stablish comparisons with both the training and testing set.

On the table of page 13, we can appreciate the results of all the algorithms. We can see that both AMSGrad and Vanilla converged to the analytical coefficients and AMSGrad accomplished it with better results and less iterations. The Scikit Learn algorithm SGDRegressor also obtained good results so we can say that this algorithm converged close to the solution and we are satisfied with this result since we did not push a high amount of iterations.

We can see on page 14 the testing set results. We can appreciate that SGDRegressor closely beats our algorithms for this specific testing set.

Also we can see on page 15 how close are our estimated coefficients with analytical the coefficients.

3 Conclusions

To close up our investigation, we present the following conclusions:

- We carried out a comprehensive tuning process that ended with optimal solutions for each algorithm.
- We have demonstrated the effects of improving the vanilla gradient descent by displaying the efficiency of AMSGrad.
- We have check the correctness of our implementations with a close comparison with the analytical coefficients.
- In general we are satisfied with our results on part 1 and part 2.

4 Appendix

See logfile pages 16-19 for graphical representations of the two variables more correlated with the target variable (Temperature and Hour). We can see how the nature of this variables is captured by the predictions of our model.