# LSDA18 - HW1

Isabela Blucher

May 7, 2018

## 1 VirtualBox & Subsets

### 1.1 Executing the notebook

By keeping the notebook in its original form, we get an error of the type `MemoryError`. The code as is, runs the K-neighbors classifier algorithm in the brute-force way, which means it computes a distance matrix that stores all pairwise distances between training and test set points (space complecity of $O(n^2)$). The VirtualBox that we're using to run the program only has 2GB of memory available, so by trying to compute this enormous distance matrix the computer runs out of available RAM, causing the error to arise.

### 1.2 Subsets for Training

Fitting the model on the first 5000 instances of the training data we get 0.03524 as the classification accuracy score.

A better option to fit the model could be to randomly choose 5000 instances from the training dataset instead of picking the first 5000, as the latter could be biased and thus jeopardize the classification accuracy. By sampling 5000 random rows from the training dataset and fitting the model on those instances, this bias would be eliminated.

Running notebook with the changes mentioned above, the accuracy score changes to 0.4446, which is a significant increase from the original classification accuracy.

## 2 Getting started with TensorFlow: Logistic Regression

### 2.1 TensorBoard

Figures 1 and 2 illustrate the accuracy measure and the cross-entropy for the algorithm implemented in the file `LogisticRegressionTensorBoardAssignment.py`. The orange represents the test set and the blue the training set. We define $\eta$ as the learning rate variable.

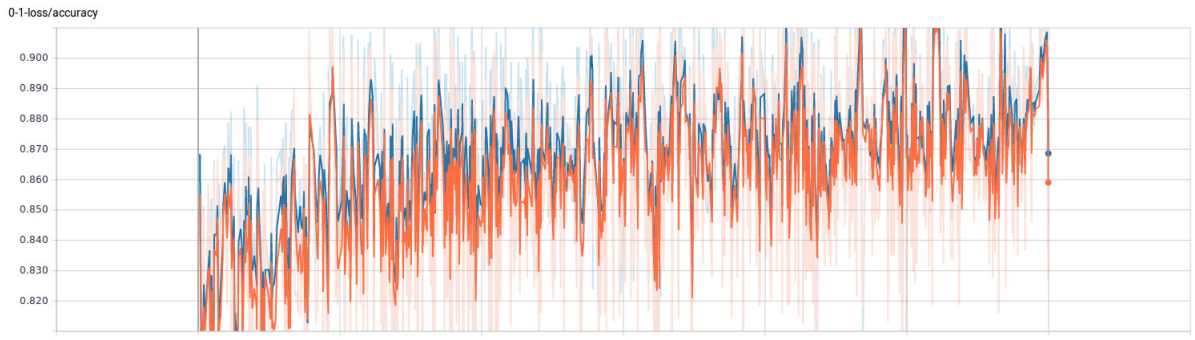


Figure 1: TensorBoard plot of accuracy measure for $\eta = 0.1$

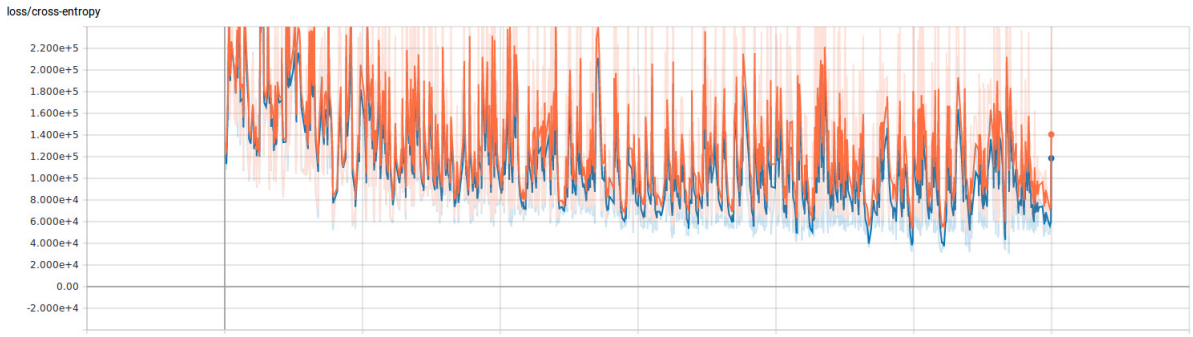


Figure 2: TensorBoard plot of cross-entropy for $\eta = 0.1$

By visual inspection, we can conclude that, as much as the general shape of the accuracy goes up and the cross-entropy goes down, 0.1 is too big a learning rate for the algorithm. The graphs fluctuate too much and the cross-entropy order of magnitude does not get smaller than $10^4$.

## 2.2 Command line options

How the `datadir=` command works is it sets the variable `data_dir` as a directory path through the command line. This is done using the `tf.app.flags` module, which is a wrapper for Python's `argparse`. The essence of it is that it handles the parsing of command line arguments when running our program. That explains why we assign a string to the variable on the command line together with the python program.

The code added to set the gradient descent optimizer learning rate through the command line also uses the `tf.app.flags` module, and it sets a float value that is going to be assigned to the variable `learning_rate`. We can also assign a value through the command line using `--learning_rate=`.

```
flags.DEFINE_float('learning_rate' , '0.1', 'learning rate of gradient descent')
```

## 2.3 Visualize compute graph

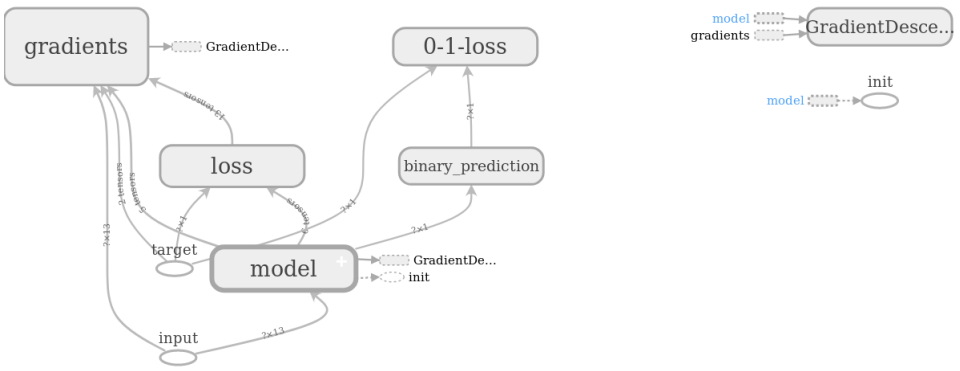


Figure 3: TensorFlow graph plot

## 2.4 Adjust learning rate

By testing the algorithm with different learning rates, we can arrive at conclusions of what is an optimal value, and consequently what values are too big or too small.

A learning rate of $10^{-6}$ generates Figures 4 and 5 as plots for accuracy and loss.

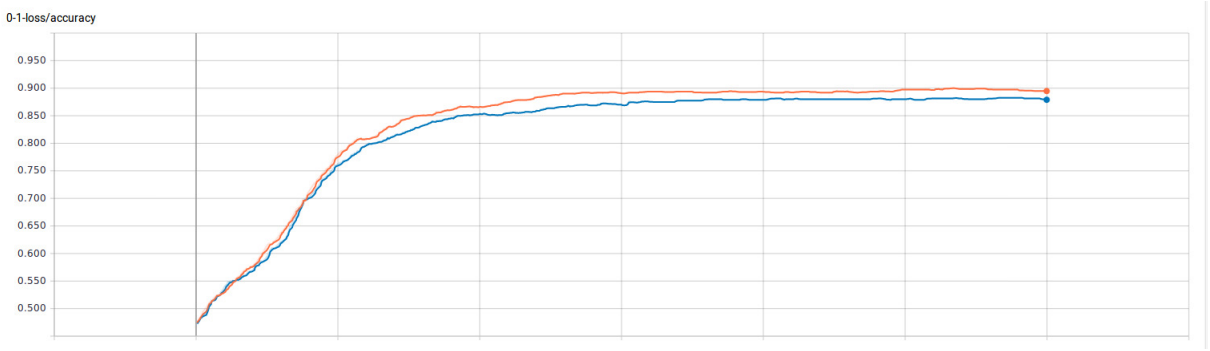


Figure 4: TensorBoard plot of accuracy measure for $\eta = 10^{-6}$

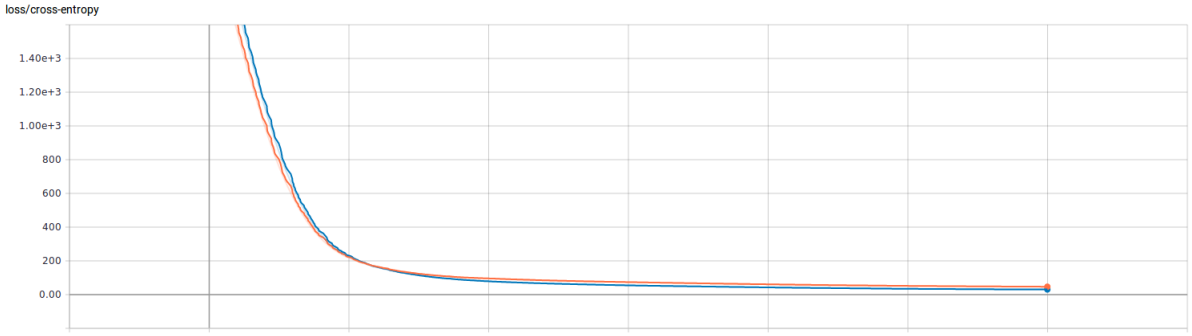


Figure 5: TensorBoard plot of cross-entropy for $\eta = 10^{-6}$

From looking at the plots we can conlude that $10^{-6}$ is a good learning rate value as the accuracy for both training and test sets is around 95% and the cross-entropy becomes smaller by three orders of magnitude and gets very close to zero.

Looking at subsection 2.1 ("TensorBoard"), we have the plots for the learning rate of 0.1 (Figures 1 and 2), which by comparison with Figures 4 and 5 is clearly too large a value. When a gradient descent's learning rate is too big, the algorithm does not converge to the minimum, since we are overshooting the lower point by taking bigger steps. This can be confirmed by the constant fluctuation in values in both the accuracy and loss plots.

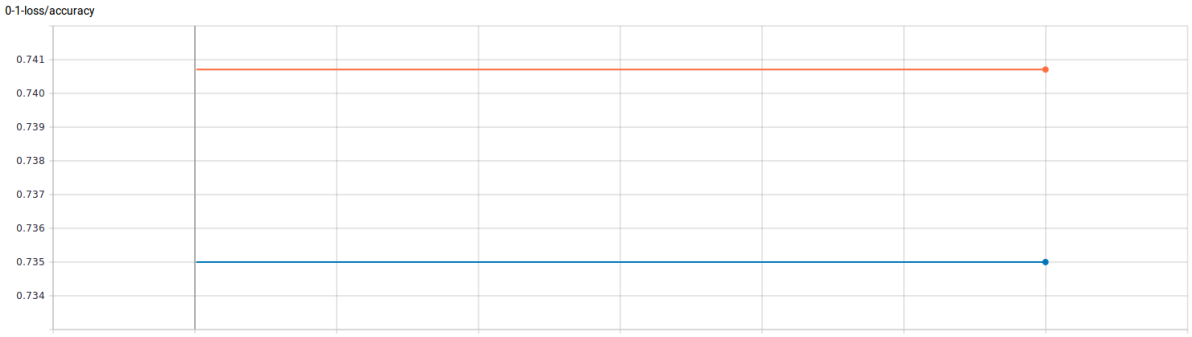


Figure 6: TensorBoard plot of accuracy measure for $\eta = 10^{-9}$

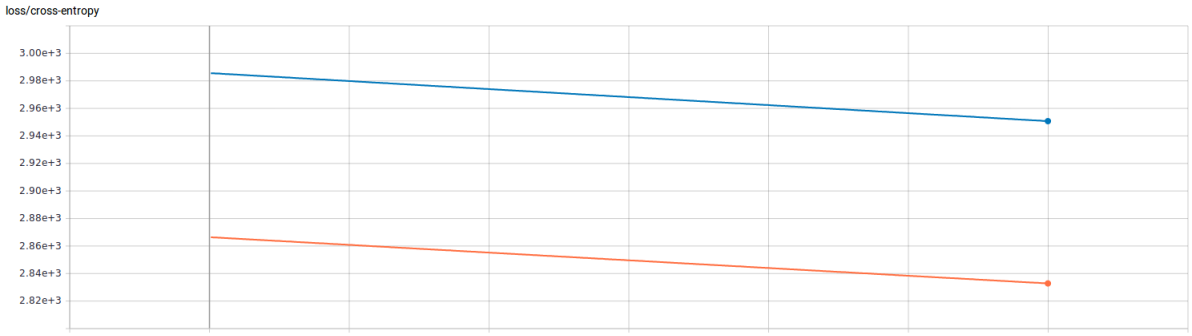


Figure 7: TensorBoard plot of cross-entropy for $\eta = 10^{-9}$

Figures 6 and 7 illustrate the accuracy measure and loss for a learning rate of $10^{-9}$. By inspecting the plots we can conclude that it is a very small value because there are no significant changes to the shapes of the lines. A small gradient descent learning rate is more precise direction-wise, but computing the gradient so frequently makes the descent take a very long time to reach the bottom.

# 3 Large-Scale Nearest Neighbors

## 3.1 Using k-d Trees

Changing the algorithm on the nearest neighbors search to `'kd_tree'`, we guarantee that the space complexity is $O(n)$, where with the brute-force algorithm, computing the distance matrix, the space complexity was $O(n^2)$. This means that the k-d trees algorithm uses a fraction of the memory space that the brute-force approach did, and it explains why we are now able to fit the model.

The accuracy score computed here with the k-d trees algorithm used on the K-neighbors classifier is 0.8966, which is a lot better the the one we got using the brute-force algorithm.

## 3.2 Subset of Features

For this section, our dataset will make use of only the first 5 features of the existing 54. We compute the accuracy score and consider the induced test error as being $error = 1 - accuracy$. The induced test error for the modified dataset is 0.24936.

We also compute the runtimes needed for the testing phase for the full dataset and the modified one. We use Python's `time.clock()`, to calculate elapsed time in seconds for the prediction phase. The full 54-dimensional dataset needs 181.196359 seconds to run, and the modified 5-dimensional dataset needs 1.3962970000000041 seconds. This significant difference in runtime can be explained by the use of the k-d trees algorithm, which is not efficient in high dimensional spaces. Most searches end up being brute searches and most of the points will end up being evaluated, which makes the algorithm no better than an exhaustive search.

## 3.3 Feature Selection

The feature selection scheme was implemented and by computing the induced validation error after each of the five rounds we get the plot in Figure 8.

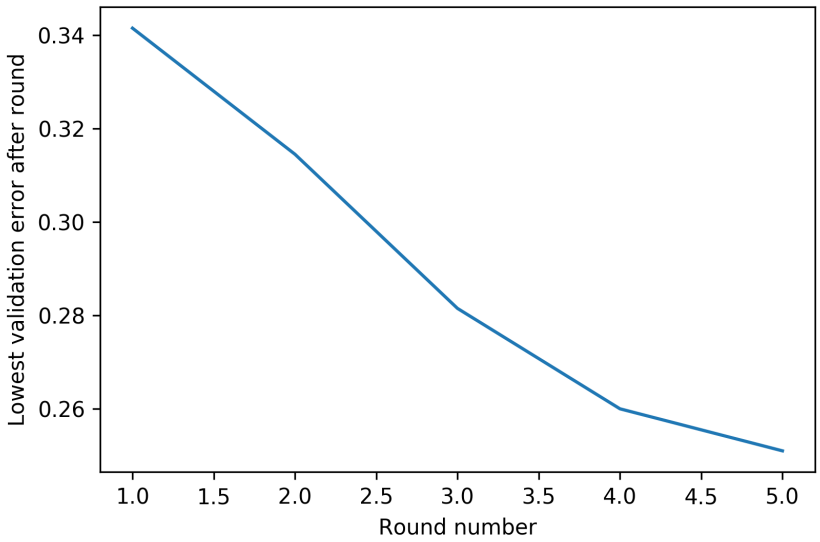


Figure 8: Evolution of lowest validation error after each of the five rounds

As we can see, the five selected features are chosen such that the induced validation error is minimized after every round.

Having selected the best features, which in this case were [0, 5, 9, 12, 23], we retrain our model with modified 5-dimensional training and test sets, containing only the features selected by the algorithm. The induced accuracy on the test set is 0.87712 and the runtime in seconds is 1.6255009999999999. We can see that the runtime achieved before is practically the same as the one achieved now, but the accuracy became better, since we have chosen optimal features to reduce the dimensionality of our datasets.

### 3.4 CIFAR-10

The preprocessing steps done here were very similar to the ones used in previous steps of Exercise 3. The one difference is flattening the images so that instead of getting a matrix of dimensions $32 \times 32 \times 3$, we get a vector with $32 \cdot 32 \cdot 3 = 3072$ elements. Since this is a very high-dimensional dataset, the same forward feature selection was implemented to choose the 10 best features. We use 10000 random instances of the training set and separate 8000 to be our training set for feature selection and the other 2000 are part of the validation set. When doing this the accuracy score obtained is 0.2193, which is a very low score. The algorithm used on the nearest neighbors classifier was the k-d trees, as the brute force would result in the same memory error from Exercise 1.

This could be explained by the fact that utilizing raw pixel values as features in a classification problem usually yields poor results as small changes to image can dramatically influence the vector, and thus not be an accurate representation of what we are trying to identify.