Practicals to Logistic Regression

Aufgabe P 2. *Logistic Regression, due 16 November, in class*

In this practical you will learn how to apply logistic regression to the task of predicting two digits from the MNIST database: http://yann.lecun.com/exdb/mnist/. The database contains 55000 train images containing digits and 10000 test images. The images are of size 28×28 and are already vectorized to a size of 784. We extracted the digits 0 and 1 for you and thus provided the data import.

(a) Implement the logistic regression model in Python with the iterative gradient descent algorithm. As in the previous practical you should compute the gradient as in the slides. So the loss function is as provided in equation 12, and the gradient as in equation 24. *Some notes:* For this exercise, the training set has a shape of $(11623, 785)$. Why 785 instead of 784? We added the bias term here already. The labels, y, are imported as an array of dimension 11623 of zero and ones and not as a one-hot encoding. Example: the third entry of y is one, then digit one is in the image. If you import digits 1 and 7 then you get an array of ones and sevens. During training you minimize the loss function. In every iteration your loss should decrease. You also want to look how many correct predictions you have at every iteration. Reminder: the belonging to class digit zero is when your prediction, $\hat{y}$, is greater or equal to 0.5. When you test your prediction vector (containing zero and ones) with the labels (also zero and ones) you can use the equal function. Example: $prediction = (1, 0, 1, 1)$ and the true labels are $y = (0, 0, 1, 0)$. When you test on equality you get following result: $correct = (0, 1, 1, 0)$. Your accuracy is: $\frac{0+1+1+0}{4} = 0.5$. You compute the accuracy for the training and test. (50 Pts)

(b) Implement the logistic regression model by means of softmax with Tensorflow and you can use the gradient descent optimizer provided by Tensorflow.
Some notes: For this exercise, we input the data with $tf.data$ and use the function $tf.data.Dataset.from\_tensor\_slices((features, labels))$ to input the numpy arrays (which are tensors) of features being the training set with a shape of $(11623, 784)$ and also the labels. Now we created a tensorflow object. We use this object to shuffle the data, limit the epochs (one epoch is one iteration over the whole dataset), and batch the input data. What is the idea of a batch? Instead of feeding the training algorithm with all the input, we partition it to mini-batches of e.g. size 10 and perform the gradient computation only on this subset instead of the whole one. This kind of training based on mini-batches is useful when not all your data fits into memory. So internally, tensorflow will use training sets of shape $(batchsize, 784)$. Since we would like to perform more than one epoch, we use an iterator for the data which needs to be initialized (the stub already took care of it). On the other hand, for testing we use the one-shot iterator that doesn't need initialization. The one-shot iterator will return as provided in the code a test sample of dimension 784. In order to perform matrix

multiplication you need to reshape it to (1×784) a row of features. This you can do with $tf.expand\_dims(img\_input\_t, 0)$. (50 Pts)

Please note that the fine grained points on the implementation tasks are provided when we look into your practical in November.