Practical Work 1: **Linear Models for Classification**

Tudor Berariu
tudor.berariu@gmail.com

October 5, 2016

You are going to solve a handwritten digit classification problem using both a linear and a non-linear classifier. The linear model's parameters will be computed in closed form, and trained using gradient descent.

1 The classification problem

The goal in classification is to build a model that takes an input vector $\boldsymbol{x}$ and assigns it to a class from a discrete finite set of classes $\{\mathcal{C}_1, \ldots, \mathcal{C}_K\}$.

A data set consists of a set of N input vectors $\boldsymbol{x}^{(n)} \in \mathbb{R}^D$ arranged as a matrix $\mathbf{X} \in \mathbb{R}^{N \times D}$, and a set of N corresponding target vectors $\boldsymbol{t}^{(n)} \in \mathbb{R}^K$ arranged as a matrix $\mathbf{T} \in \mathbb{R}^{N \times K}$. Each $\boldsymbol{t}^{(n)}$ is one-hot coded.

A linear classifier computes K output values y_k, and assigns the input vector to the class corresponding to the largest y_k.

$$y_k(\boldsymbol{x}) = \boldsymbol{w}_k{}^{\mathrm{T}}\boldsymbol{x} + w_{k0} = \widetilde{\boldsymbol{w}}_k^{\mathrm{T}}\widetilde{\boldsymbol{x}} \tag{1}$$

In Formula 1 $\widetilde{\boldsymbol{w}}_k \in \mathbb{R}^D$ represents the parameters of the k^{th} output. The same equation can be written in matrix form as in Formula 2, where $\widetilde{\mathbf{W}}$ is a matrix of size $(D+1) \times K$ having $\widetilde{\boldsymbol{w}}_k$ as columns.

$$\boldsymbol{y}(\boldsymbol{x}) = \widetilde{\mathbf{W}}^{\mathrm{T}}\widetilde{\boldsymbol{x}} \tag{2}$$

2 The MNIST data set

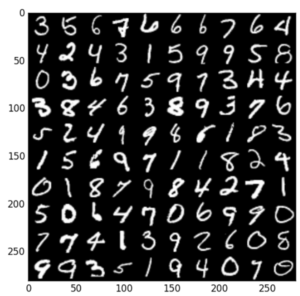
The data set we are going to use today is MNIST which consists of handwritten digits. First, install the MNIST parser using `pip`.

```
$ pip install python-mnist
```

Check that everything works as expected.

```
$ python mnist_loader.py
[[3 5 6 7 6 6 6 7 6 4]
 [4 2 4 3 1 5 9 9 5 8]
 [0 3 6 7 5 9 7 3 4 4]
 [3 8 4 6 3 8 9 3 7 6]
 [5 2 4 9 9 8 6 1 8 3]
 [1 5 6 9 7 1 1 8 2 4]
 [0 1 8 7 9 8 4 2 7 1]
 [5 0 6 4 7 0 6 9 9 0]
 [7 7 4 1 3 9 2 6 0 8]
 [9 9 3 5 1 9 4 0 7 0]]
```

You should also see a matrix of 10×10 digits.



3 Solving a linear classifier in closed form

Consider the sum of squares error function in Formula 3.

$$E\left(\widetilde{\mathbf{W}}\right) = \sum_{n=1}^{N} \frac{1}{2} \left(\widetilde{\mathbf{W}}^{\mathrm{T}} \boldsymbol{x}^{(n)} - \boldsymbol{t}^{(n)}\right)^2 = \frac{1}{2} Tr\left[\left(\widetilde{\mathbf{X}}\widetilde{\mathbf{W}} - \mathbf{T}\right)^{\mathrm{T}} \left(\widetilde{\mathbf{X}}\widetilde{\mathbf{W}} - \mathbf{T}\right)\right] \tag{3}$$

In order to minimize $E\left(\widetilde{\mathbf{W}}\right)$ we compute its gradient and set it to zero. This yields a closed form solution for the optimal parameters (Formula 4).

$$\widetilde{\mathbf{W}} = \left(\widetilde{\mathbf{X}}^{\mathrm{T}} \widetilde{\mathbf{X}}\right)^{-1} \widetilde{\mathbf{X}}^{\mathrm{T}} \mathbf{T} = \widetilde{\mathbf{X}}^{\dagger} \mathbf{T} \tag{4}$$

Use `np.linalg.pinv` to compute the pseudo-inverse of a matrix.

4 Minimizing error using gradient descent

Consider again the sum of squares error function (Formula 5).

$$E\left(\widetilde{\mathbf{W}}\right) = \sum_{n=1}^{N} \frac{1}{2} \left(\widetilde{\mathbf{W}}^{\mathrm{T}} \boldsymbol{x}^{(n)} - \boldsymbol{t}^{(n)}\right)^2 \tag{5}$$

Differentiating $E\left(\widetilde{\mathbf{W}}\right)$ in Formula 5 with respect to $\widetilde{\mathbf{W}}$.

$$\nabla_{\widetilde{\mathbf{W}}} E = \sum_{n=1}^{N} \left(\widetilde{\mathbf{W}}^{\mathrm{T}} \boldsymbol{x}^{(n)} - \boldsymbol{t}^{(n)}\right) \boldsymbol{x}^{(n)\mathrm{T}} = \left(\widetilde{\mathbf{X}}\widetilde{\mathbf{W}} - \mathbf{T}\right)^{\mathrm{T}} \widetilde{\mathbf{X}} \tag{6}$$

Gradient based algorithms are iterative procedures that start with some random parameters $\widetilde{\mathbf{W}}^{(0)}$ and try to search the parameter space by moving in the opposite direction to the gradient of the error.

$$\widetilde{\mathbf{W}}^{(t+1)} = \widetilde{\mathbf{W}}^{(t)} - \eta \nabla_{\widetilde{\mathbf{W}}} E \tag{7}$$

η is a hyper-parameter called *learning rate*.

5 Adding a sigmoid transfer function

Add a logistic transfer function as in Formula 8.

$$\boldsymbol{y}(\boldsymbol{x}) = \sigma\left(\widetilde{\mathbf{W}}^{\mathrm{T}} \widetilde{\boldsymbol{x}}\right) \tag{8}$$

The gradient of the error w.r.t. the parameters takes the form in Formula 9.

$$\nabla_{\widetilde{\mathbf{W}}} E = \left((\mathbf{Y} - \mathbf{T}) \odot \mathbf{Y} \odot (1 - \mathbf{Y})\right)^{\mathrm{T}} \widetilde{\mathbf{X}} \tag{9}$$

6 Tasks

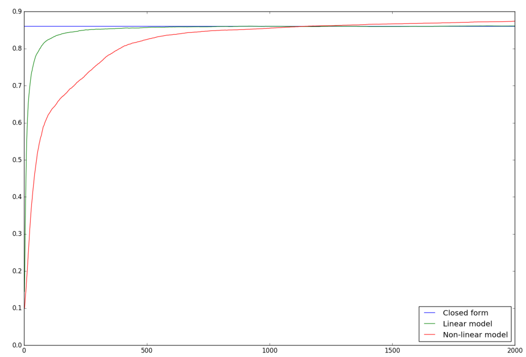
1. See how the data set is loaded and used in `train.py`.
   - You should see three confusion matrices.
2. Compute the closed form parameters of the linear model in `closed_form(self, X, T)` in file `linear_classifier.py`.
3. Compute the output for a given set of inputs. Write your code in `output(self, X)` in file `linear_classifier.py`.
   - Now you should get an accuracy of 86.01% when running `train.py`.

```
$ python train.py
[Closed Form] Accuracy on test set: 0.860100
[[ 944    0    1    2    2    7   14    2    7    1]
 [   0 1107    2    2    3    1    5    1   14    0]
 [  18   54  812   26   15    0   42   22   38    5]
 [   4   17   23  879    5   17   10   21   22   12]
 [   0   22    6    1  881    5   10    2   11   44]
 [  23   18    3   72   24  659   23   14   39   17]
 [  18   10    9    0   22   17  875    0    7    0]
 [   5   40   16    6   26    0    1  884    0   50]
 [  14   46   11   30   27   40   15   12  759   20]
 [  15   11    2   17   80    1    1   77    4  801]]
```

4. Compute a step of parameters update using gradient descent in `update_params(self, X, T, lr)` in `linear_classifier.py`.

- You should now see the evolution of the accuracy on the test set for the linear model.

5. Implement the `output` method in class `SigmoidClassifier`.
6. Implement the `update_params` method in class `SigmoidClassifier`.
   - You should now see a plot similar to this one:



7 Discussion

Provide answers for the following questions.

1. During gradient descent training the accuracy on the test set for the linear model was sometimes a bit higher than the analytic solution. How is this possible?
2. Why do these models not overfit?
3. Why does the nonlinear model outperform the linear one?
4. See what happens when you vary the learning rate.

8 `Numpy` hints

Use:

numpy.linalg.pinv to compute the pseudo-inverse of a function;

numpy.hstack to concatenate arrays horizontally;

numpy.dot to multiply matrices;

numpy.transpose to transpose a matrix;

numpy.exp to compute the exponential element-wise in an array;