# FRTN50 hand in 4

Karl Tengelin

November 2019

## Task 1- Reduced MNIST and Average Confidence

> Design a classifier with a reduced training set consisting of the digits 0-2 and the size reduced with a factor 3, i.e. loadmnist(0:2,reduction=3,set=:train). Use the average confidence formulation in Exercise 4.12 and the following 5:th order polynomial kernel $K(x,y) = \left(x^T y\right)^5$. Solve the problem with a coordinate descent algorithm. What is the best error rate you can achieve?

Here we will use the average confidence classification described in exercise 4.12:

$$\begin{aligned} c_\lambda^A(x) &= \frac{1}{K}\sum_{k=1}^{K}\left(m_\lambda(x) - m_k(x)\right) = \frac{1}{K}\sum_{k=1}^{K}\left(w_\lambda^T x - w_k^T x\right) \\ &= w_\lambda^T x - \frac{1}{K}\sum_{k=1}^{K} w_k^T x \end{aligned} \tag{1}$$

Combined with a hinge loss and a squared 2-norm regularization this gives the following expression:

$$\min_w \underbrace{\sum_{i=1}^{n}\max\left(0,1 - A_i^T X_i^T w\right)}_{f(A^T X^T w)} + \underbrace{\frac{\gamma}{2}\|w\|_2^2}_{g(w)} \tag{2}$$

Which is the primal problem we seek to minimize for our parameters w.

Here

$$X_i = \begin{bmatrix} x_i & & \\ & \ddots & \\ & & x_i \end{bmatrix} \in \mathbb{R}^{pK\times K} \tag{3}$$

and $x_i$ represents the MNIST data we aim to train on/classify, so each $x_i$ is in our case a column vector with length p ($x_i \in \mathbb{R}^p$) containing ones or zeros which

represents pixels in a picture. K signifies number of classes that are examined, so having K = 3 means we have the three classes 0,1 and 2 that we train our SVM for.

$A_i$ is defined as:

$$A_i = e_{y_i} - \frac{1}{K}\mathbf{1} \in \mathbb{R}^K \tag{4}$$

Where $e_i$ is a unit column vector with the i:th element being a one and all other elements zero. So in our case with K = 3 $A_i$ can look the following ways:

$$A_{i0} = \begin{bmatrix} 0.6667 \\ -0.3333 \\ -0.3333 \end{bmatrix}, A_{i1} = \begin{bmatrix} -0.3333 \\ 0.6667 \\ -0.3333 \end{bmatrix}, A_{i2} = \begin{bmatrix} -0.3333 \\ -0.3333 \\ 0.6667 \end{bmatrix} \tag{5}$$

Here $A_{i0}$ represents what A would look like for class 0, $A_{i1}$ for class 1 and $A_{i2}$ for class 2.

We also identify $\mathbb{X}$ and $\mathbb{A}$ as:

$$\mathbb{X} = [X_1, \ldots, X_n] \in \mathbb{R}^{pK \times Kn} \text{and } \mathbb{A} = \text{blkdiag } (A_1, \ldots, A_n) \in \mathbb{R}^{Kn \times n} \tag{6}$$

Where n represents the total training set, i.e the total number of $x_i : s$ we have to train on. So $\mathbb{X}$ contains all training pictures arranged according to 3 and $\mathbb{A}$ contains all variations of 5.

We aim to solve the dual problem which can be stated as:

$$\min_{\mu} f^*(\mu) + g^*(-\mathbb{X}\mathbb{A}\mu) \tag{7}$$

Where the conjugate $g^*$ is defined as $g^*(v) = \frac{1}{2\gamma}\|v\|_2^2$ and we can then rewrite $g^*(-\mathbb{X}\mathbb{A}\mu)$ in 7 as $\frac{1}{2\gamma}\mu^T\mathbb{A}^T\mathbb{X}^T\mathbb{X}\mathbb{A}\mu$, meaning that we can write the dual problem as:

$$\begin{array}{ll} \text{minimize} & \mu^T\mathbf{1} + \frac{1}{2\gamma}\mu^T\mathbb{A}^T\mathbb{X}^T\mathbb{X}\mathbb{A}\mu \\ \text{subject to} & \mu \in [-1,0] \end{array} \tag{8}$$

We then utilise the primal dual optimality conditions in order to recover a primal solution:

$$w = \partial g^*(-\mathbb{X}\mathbb{A}\mu) = \frac{-1}{\gamma}\mathbb{X}\mathbb{A}\mu \tag{9}$$

We also don't want to evaluate the big $\mathbb{X}$-martices so we make use of the following kernel:

$$K(x,y) = \left(x^T y\right)^5 \tag{10}$$

This means that we can replace $\frac{1}{\gamma}\mathbb{A}^T\mathbb{X}^T\mathbb{X}\mathbb{A}$ in 8 with a matrix $Q \in \mathbb{R}^{n\times n}$ .
Each element in Q consists of the kernel of two pictures, $x_i$ and $x_j$: $K(x_i, x_j) = \left(x_i^T x_j\right)^5$ arranged in a $K \times K$ matrix, i.e:

$$\begin{bmatrix} K(x_i,x_j) & & \\ & K(x_i,x_j) & \\ & & K(x_i,x_j) \end{bmatrix} \tag{11}$$

left multiplied by $A_i^T$ and right multiplied by $A_j$. As an example we can assume that $x_i$ is of class 0 and $x_j$ is of class 2, this would give the following element in Q:

$$Q(i,j) =$$
$$\frac{1}{\gamma}\begin{bmatrix} 0.6667 & -0.3333 & -0.3333 \end{bmatrix}\begin{bmatrix} K(x_i,x_j) & & \\ & K(x_i,x_j) & \\ & & K(x_i,x_j) \end{bmatrix}\begin{bmatrix} -0.3333 \\ -0.3333 \\ 0.6667 \end{bmatrix} \tag{12}$$

Using this Q instead we can write 8 as:

$$\begin{aligned} &\text{minimize} && \mu^T\mathbf{1} + \tfrac{1}{2}\mu^T Q\mu \\ &\text{subject to} && \mu \in [-1,0] \end{aligned} \tag{13}$$

In the same way we don't want to evaluate big $\mathbb{X}$ when recovering the primal solution, therefore we again make use of the kernel along with the fact that we are working with diagonal matrices, meaning that we can write matrix multiplications simply as a sums (see classifier in *mnist_library.jl*).

## Results

Using a training set consisting of the classes 0-2 with a reduction factor of 3 and doing a proximal gradient coordinate descent on the dual problem (13) over 500 000 iterations we get the following results on the training set and the test set respectively:

| | Error rate | Training iterations | Training time (sec) |
|---|---|---|---|
| Train | 0.00773 | 500 000 | 136.5 |
| Test | 0.0124 | 500 000 | 136.5 |

Table 1: Results for SVM with coordinate gradient descent

We can also examine some of the missclassified pictures from both the train set and the test set to see what they look like and compare them to correctly classified images:

## Missclassified from train set

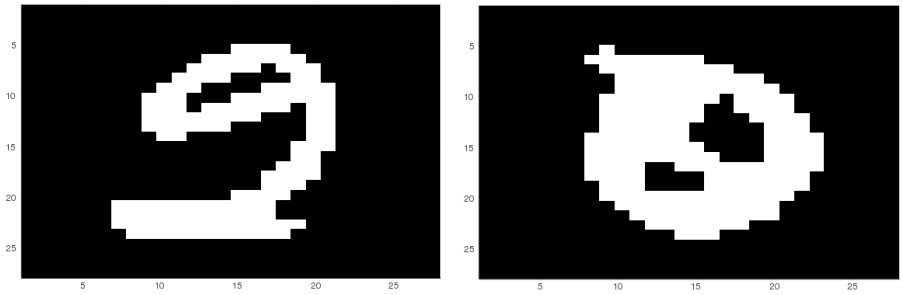

Figure 1: Missclassified pictures from the train set

## Correctly classified from train set

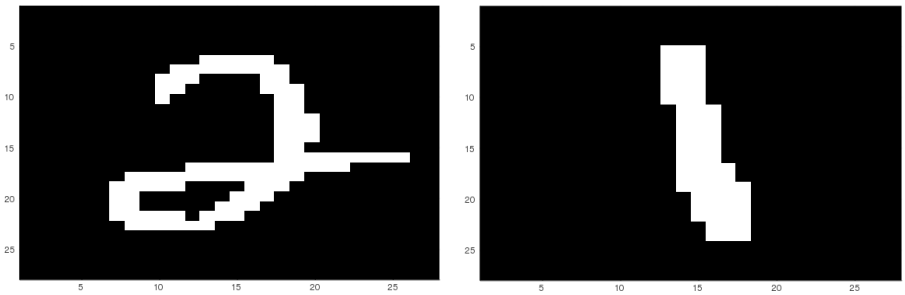

Figure 2: Correctly classified pictures from the train set

## Missclassified from test set

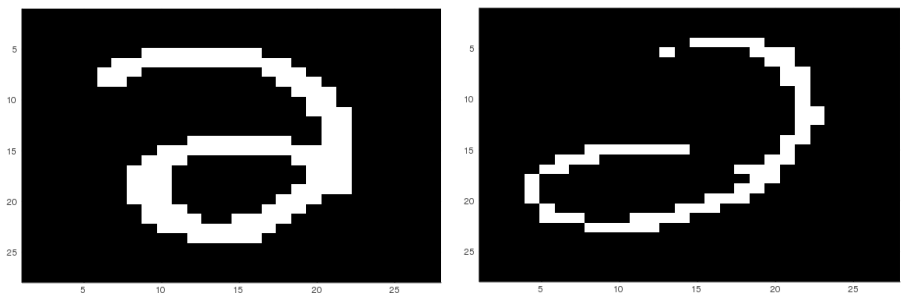

Figure 3: Missclassified pictures from the test set

## Correctly classified from test set

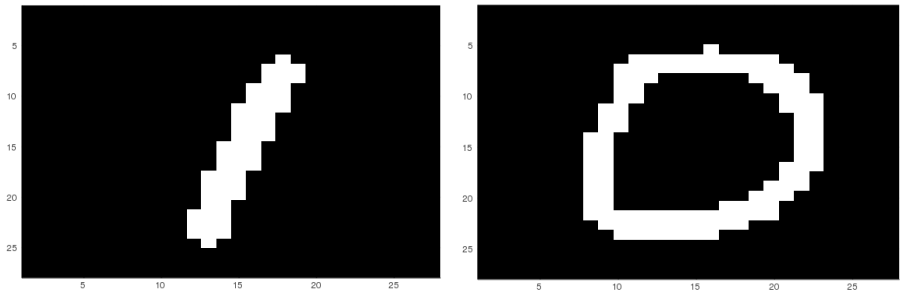


Figure 4: Correctly classified pictures from the test set

Where one can see that the missclassified images appears to be sloppier than the correctly classified ones.

## Task 2 - Algorithm Complexity

Experiment with the data set size, i.e. add/remove more digits and/or smaller/larger reduction factor. Roughly, how large problems can you solve? In theory, how large problems should you be able to solve? Look at how does the following scale with the number of data points and classes/digits.

- The algorithms memory requirement.
- The computational cost for one iteration of your algorithm.
- The computational cost for classification of unseen data.

What are the bottlenecks of the algorithm?

One of the bottlenecks in this algorithm is to create big matrices such as the Q-matrix. The size of the Q-matrix increases quadratically with the number of training points (i.e training pictures) as one can see in how the code is implemented for function *make_Q* in *mnist_library.jl*. Therefore we can say that theoretically when we use a bigger data-set the computational cost for building Q increases by $n^2$.

The following tables illustrates some combinations of number of classes (K) and different reductions for:

- Creating Q
- Doing the coordinate proximal gradient method
- Doing the classification of the unseen data

| K | Reduction | time (sec) | allocations | GiB | gc time | length train |
|---|---|---|---|---|---|---|
| 3 | 4 | 29.83 | 131.04 M | 13.775 | 7.19% | 4655 |
| 3 | 3 | 51.67 | 231.19 M | 24.402 | 10.78% | 6207 |
| 3 | 2 | 119.21 | 520.32 M | 54.919 | 9.94% | 9312 |
| 3 | 1 | 502.92 | 2.08 G | 219.646 | 8.91% | 18623 |
| 2 | 3 | 24.82 | 107.92 M | 9.742 | 7.78% | 4221 |
| 4 | 3 | 92.50 | 408.56 M | 43.120 | 6.36% | 8251 |
| 5 | 3 | 143.93 | 624.04 M | 75.166 | 4.46% | 10198 |

Table 2: Creating Q

| K | Reduction | time (sec) | allocations | GiB | gc time | length train |
|---|---|---|---|---|---|---|
| 3 | 4 | 94.80 | 5.29 M | 17.696 | 6.73% | 4655 |
| 3 | 3 | 127.99 | 3.50 M | 23.391 | 5.94% | 6207 |
| 3 | 2 | 238.51 | 3.50 M | 34.955 | 4.50% | 9312 |
| 3 | 1 | – | – | – | – | – |
| 2 | 3 | 79.74 | 5.29 M | 16.086 | 6.30% | 4221 |
| 4 | 3 | 197.54 | 3.64 M | 31.027 | 4.86% | 8251 |
| 5 | 3 | 224.12 | 3.50 M | 38.263 | 4.22% | 10198 |

Table 3: Algorithm proximal gradient method

| K | Reduction | time (sec) | allocations | GiB | gc time | length train |
|---|---|---|---|---|---|---|
| 3 | 4 | 5.71 | 29.33 M | 3.223 | 11.90% | 4655 |
| 3 | 3 | 10.17 | 52.12 M | 5.727 | 12.18% | 6207 |
| 3 | 2 | 22.60 | 117.30 M | 12.890 | 12.87% | 9312 |
| 3 | 1 | – | – | – | – | – |
| 2 | 3 | 5.02 | 24.53 M | 2.208 | 9.90% | 4221 |
| 4 | 3 | 17.10 | 91.59 M | 10.572 | 9.66% | 8251 |
| 5 | 3 | 28.19 | 139.83 M | 19.270 | 12.72% | 10198 |

Table 4: Classification unseen data

Here the combination $K = 3, \text{reduction} = 1$ has been cancelled due to the long computational time (20% of the algorithm took approximately 45 minutes). We can see that the computational cost increases with the size of the training set (which is to be expected). And we can also see for the algorithm that creates Q that if we increase the data set with a factor 2, see for example K = 3 reduc. = 2 and reduc. = 1, the GiB and number of allocations needed goes up by 4 ($2^2$), which is what we expect from theory.

If we compare table 2 and 3 we can see that the proximal gradient algorithm takes about twice to three times as long as the algorithm for creating Q, however the number of allocations needed varies considerably less for the proximal gradient method.

We do make efforts to minimize allocations needed for the proximal gradient method, in the sense that we are creating three vectors of fixed length and replace the values in these vectors to perform the operations, rather than creating a new vector repeatedly. However similar efforts are made when creating Q (see $mnist_library.jl$) so the fact that the allocations differ that much is a bit of a surprise. One thing that could be improved in the code in hindsight is that the "$A_i$" and "$A_j$" vector that are created in $make\_Q$ could be made more efficient

using the same principle as in the proximal gradient method (i.e by having two vectors which changes value instead of creating new ones) however these are not large vectors ($3 \times 1$) so the effect should not be too significant.

If we examine table 4 we can see that the classification also scales in time with the length of the training set by a factor $n^2$. However compared to the other algorithms the classification takes relatively little time, even though allocations and memory requirement is not that much smaller than the other algorithms. Which means that it is not the classification that is the bottleneck.

In summary we can say that it appears that the bottleneck is the proximal gradient descent method, and that it is probably the proximal operation that takes time since we need to do it multiple times. And apart from the prox operation in the proximal gradient method all the operations are scalar products, which should be easy to compute. Other than that the creation of Q is also quite expensive end especially if we have big training sets due to the the quadratic increase of the complexity.

We should in theory be able to compute bigger problems and if we had enough time it should be possible to train on the whole mnist-set. However we can see that just creating the Q-matrix with the whole data set would take roughly 5000 seconds $\approx$ 1.4 hours and the proximal gradient method takes much longer.