Machine Learning E16 - Handin 1
Linear classification of handwritten digits

Mark Medum Bundgaard, Morten Jensen, Martin Sand Nielsen
Aarhus University

September 27, 2016

1 Optical Character Recognition

>Skriv lidt om OCR, så nogen rapportnazister et sted i verden bliver lykkelige.<

The algorithms of this exercise is programmed in Python 3.5 and Numpy by expanding the provided examples/template Jupyter Notebook.

1.1 Data sets

Two similar data sets is used for training and testing linear classifiers of handwritten digits. The first is the MNIST database[1] with 60000 examples for training and 10000 for testing. Second we have a smaller test and training set which has been written by students following the course *Machine Learning* at Aarhus University through the last couple of years. We will distinguish them as the MNIST and AU data sets.

Both dataset stores each digits as a 8-bit monochrome 28×28 white on black image. The MNIST digits has been centered and size-normalized, which could result in more consistent results, than with the non-edited AU digits.

The image features to be used for classification is the raw 784 pixel values. With n such sample vectors $\vec{x}_i \in \mathbb{R}^{784}$, we are given the set as a matrix, $\mathbf{X} = [x_1, x_2, \ldots, x_n]^T$. For training and testing a classifier the corresponding labels $y_i \in \{0, 1, \ldots, 9\}$ is given as the column vector $\vec{y} = [y_1, y_2, \ldots, y_n]^T$.

2 Linear classification

The linear classifier can be understood as decision-hyperplane, which can be written as $\vec{w}^T\vec{x} + w_0 = 0$, where an dummy coordinate to the data vector, $x_0 = 1$, allow for having all the classifier parameters in one single vector $\vec{w} = [w_0, w_1, \ldots, w_{784}]^T$. The equality is only true for data points that lie on the decision-surface, where points above or below will result in a negative or positive result, which can be used for classification between two classes, known as the *perceptron*.

[1] http://yann.lecun.com/exdb/mnist/

Of course a 0-9digit OCR chould be able to distinguish all 10 classes of digits. But the simple linear classifier is still implemented for illustrative purposes on the 2's and 7's of the dataset. In a similar way this could been tried to train a one-versus-all classifier, that could tell ie. 2's from all other digits. The Softmax multiclass classifier will be introduced in section 2.4.

2.1 Logistic regression

Since the perceptron-algorithm that can find the parameters of such a linear classifier is at no use when data points are not linear separable, we will try using an approach that can allow for some classification errors, but can be numerically optimized. The logistic sigmoid probability distribution and its derivative

$$\sigma(z) = \frac{1}{1+e^{-z}}, \quad \frac{\partial\sigma(z)}{\partial z} = (1-\sigma(z))\sigma(z)$$

can be used to model the target values with $\sigma(\vec{w}^T\vec{x})$. It can be found that a negative log likelihood (NLL) of the data points (digits and targets) actually come from such a distribution is given by,

$$\mathrm{NLL}(\mathbf{X}, \vec{y}|\vec{w}) = -\sum_{i=1}^{n} y_i \ln\sigma(\vec{w}^T\vec{x}_i) + (1-y_i)\ln\sigma(\vec{w}^T\vec{x}_i),$$

with the gradient $\frac{\partial \mathrm{NLL}}{\partial \vec{w}} = -\mathbf{X}^T(\vec{y} - \sigma(\vec{w}^T\vec{x}))$. This NLL wil be used in a cost function, $E_{in} = \frac{1}{n}\mathrm{NLL}$, and its gradient $\nabla E_{in} = \frac{1}{n}\frac{\partial \mathrm{NLL}}{\partial \vec{w}}$.

2.2 Gradient descent

The E_{in} can be used to compare different learned classifier parameters $\vec{w}$ against each other. Since the cost is convex, the gradient can be used to numerically find the $\vec{w}$ (weights) that minimize cost, by iteratively moving a little downhill in the negative gradient (Gradient descent) direction ,

$$\vec{w}_{\mathrm{i}+1} = \vec{w}_i - \eta \cdot \nabla E_{in}(\vec{w}_i). \tag{1}$$

The *step size* or *learning rate*, η, is a positive scalar hyper parameter that can have a huge impact on training time. Too big step size could result in never converge towards minimum cost, if the algorithm is 'stepping over it'. Too small step size results in lots of iterations and unnecessary computations. It will be computational expensive to compute the cost and gradient over and over for big training sets.

2.3 Stochastic gradient descent

To improve training performance with big data sets, stochastic gradient descent (SGD), takes advantage of the fact that smaller subsets will have approximately similar cost function and gradient. So by taking just a *mini batch* of the permuted full dataset to each iteration in the gradient descent algorithm, it would approximate the same weights. This of course introduces another hyper parameter, the *batch size*, to our training model. When all data points have

been used once, it is permuted again. A small batch size makes computation times way smaller but with increased stochastic descent towards minimum.

We have implemented both gradient descent and SGD for a comparison in section 3.

Theoretical considerations

Since the each image pixels represents a coordinate in the 784-dimensional space, they can be permuted the same way and we would not have lost any of the information our classifiers is going to use. When training a classifier on these permuted sets, the classifier would converge towards the same weights, just permuted as well.

Furthermore we could expect that if the data set is linearly separable, the weights will be similar to those of a successful perceptron algorithm. But since the weights can be scaled with a constant without changing the decision hyperplane, they could converge towards infinity if small numerical uncertainties were to never let the gradient be exactly zero.

Therefore regularization of the weights is explored in section 3.2. An additional term to the cost, that represent a penalty for large weights either as l^1- or l^2-norm, would result in a convex addition to the gradient, which would then still be convex.

2.4 Softmax regression

To expand the use of linear classification to a multiclass case, softmax can be used. The setup for training softmax for K classes is largely similar to the logistic sigmoid. The target values are stored as a 1-of-K vector, which makes the combined target matrix $\mathbb{Y}$ has shape $n \times K$. For each digit softmax computes a K probabilities, one for each possible class. The digit is then classified accordingly to the highest class probability. The softmax,

$$\text{softmax}(\vec{x})_j = \frac{e_j^x}{\sum_{i=1}^K e_i^x}, \text{ for } j = 1, \ldots, K, \tag{2}$$

ensures that the sum of probabilities is always unity. This time the weights are in a matrix $\mathbf{W}$ with size $d + 1 \times K$, to accomodate the K classes and dummy coordinate. When modeling the target vectors with $\text{softmax}(\mathbf{w_j}^T \vec{x})$, we can find a NLL cost function,

$$E_{\text{in}} = -\frac{1}{n} \sum_{\vec{x_i}, \vec{y_i}} \vec{y_j}^T \ln \text{softmax}(\mathbf{W}^T \vec{x_j}), \tag{3}$$

and its gradient,

$$\nabla E_{\text{in}} = -\mathbf{X}^T \left(\mathbf{Y} - \mathbf{softmax}(\mathbf{XW}) \right). \tag{4}$$

These can with just minor adjustments be used in the gradient descent and SGD algorithms used earlier.

3 Results

3.1 Step size

3.2 Regularization

3.3 Training time

>Evaluate min-batch vs batch <

3.4 Classification errors

>Vis resultater (hvor mange fejl osv)< >Vis eksempler på fejlklassificerede billeder<

4 Discussion