Stony Brook University
Department of Computer Science
CSE 353: Machine Learning
(Spring 2014)
Instructor: Luis E. Ortiz

Programming Homework 3: Handwritten Digits Recognition using Neural Networks and SVMs

Out: *Thursday, April 24;* **Due**: *Thursday, May 1*

Overview

In this homework you will revisit the problem of learning to recognize optical handwritten digits, which you explored in Homework 1 using nearest neighbors classification, but now using neural networks and support vector machines (SVMs).

NOTE: The implementation of the algorithm to learn SVMs is for extra credit! Also, the Appendix with the derivations of the learning algorithms is only for your convenience, in the case of neural networks, or for your intellectual curiosity, in the case of SVMs.

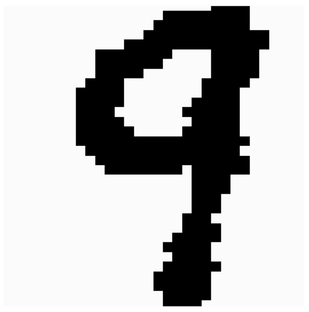
Handwritten Digits Dataset (Revisited)

For your convenience, the following is a copy of the description given in Homework 1. However, to reduce running times, the actual data files are only a subset of those used for Homework 1. Hence, do not use the files from Homework 1. Only use the files provided in this homework.

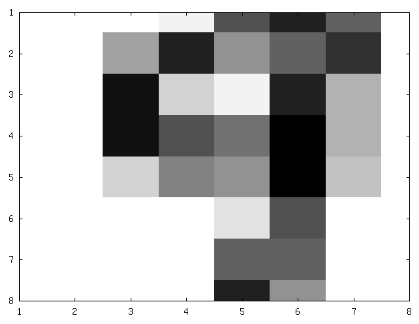
The data set of examples was created by pre-processing 32x32 bitmap images of hanwritten digits (i.e., each image was originally represented as a 32x32 matrix of pixels, each pixel taking value 1 or 0 corresponding to a black or white pixel, respectively). The result of the pre-processing is an 8x8 "grayscale" image where each pixel takes an integer value from 0 to 16. [1]

The following is an example for the digit 9:

Original Image



Transformed Image



For each preprocessed image, each pixel corresponds to an attribute taking one of 16 values. Thus, the data has 64 attributes. Although each attribute is an integer from 0 to 16, treat each attribute as real-valued. There are 10 classes corresponding to each digit.

You are provided five (5) data files.

- `optdigits_train.dat` contains (a permuted version of) the original training data.

[1] Pre-processing is done to "normalize" the data to help correct for small distortions and reduce dimensionality. The resulting images provide a good approximation of the original images for classification purposes. Please visit `http://archive.ics.uci.edu/ml/datasets/Optical+Recognition+of+Handwritten+Digits` for more information about this data set.

- `optdigits_cv.dat` contains the testing data.
- `optdigits_train_trans.dat` contains the transformed image version of the examples in the file `optdigits_train.dat`.
- `optdigits_cv_trans.dat` contains the transformed image version of the test examples.
- `optdigits_trial.dat` contains an example of each digit from the original validation set.
- `optdigits_trial_trans.dat` contains the transformed image version of the examples from `optdigits_trial.dat`.

Each file is composed of one data example per line. Each line of the `..._trans.dat` files contain 65 integers separated by a single empty space. The first 64 integers correspond to the values of each of the attributes (i.e., a number from 0 to 16) and the last integer is the example class label (i.e., the corresponding digit $0, 1, \ldots, 9$). The case is similar for the other files but now each line contains 1025 integers separated by a single space. The first 1024 correspond to the values of a bitmap of the image, where each consecutive sequence of 32 values, starting with the value in the first column, corresponds to a row of black (1) and white (0) pixel values of the image of the handwritten digit. The value of the digit is given as the last element of the 1025 vector/row. A proper reshaping of the vector composed of the first 1024 binary values in the file-row will produce a 32x32 black-and-white image. The training, test and trial data sets have 1934, 946 and 10 examples, respectively.

Neural Networks

First, you are asked to implement FORWARDPROP and BACKPROP.

Model and Implementation Details. All the neural network units are sigmoid. The *error function* used for BACKPROP is the *mean squared error. You must use the batch-learning version of* BACKPROP *described in the* **Neural Networks** *Subsection of the* **Learning Algorithms** *Section, not the online-learning version described in the reading material, Section 18.5 of Russell and Norvig's AI textbook.* See the **Neural Networks** Subsection of the **Learning Algorithms** Section of the homework for details on the implementation.

Neural Network Architectures: You will use two neural network architectures, depending on whether you are using the data in its *transformed* or *original* form:

1. For the *transformed* data, you will use a *perceptron* with 64 input units and 10 output units. For learning, you will run BACKPROP using a learning rate $\lambda = 1$ for a maximum of 1000 rounds.
2. For the *original* data, you will use a neural network with 1 *hidden layer*, 1024 input units, 64 units in the hidden layer and 10 output units. For learning, you will run BACKPROP using a learning rate $\lambda = 21$ for a maximum of 200 rounds.

Weight Initialization and Extra Stopping Conditions: Initialize each neural network weight (including thresholds) with a number independently drawn at random from the uniform distribution over $[-1, 1]$. In both cases, you will stop BACKPROP if, from one round to the next, the *change in the error function* value is $\leq 10^{-6}$ or if the *maximum change in neural network weight* values is $\leq 10^{-4}$.

Evaluation and Report. Perform the following steps for each network architecture described above.

- Record
  1. the training and test error function values and
  2. the training and test misclassification error rate

  for the neural network at each round of BackProp.
- Use your recorded values to plot
  1. the training and test error function values together in one graph and
  2. the training and test misclassification error rate together on a separate graph

  as a function of the number of rounds of BackProp. *Include these two graph plots, along with a brief discussion of the results, in your report.*
- Keep track of the neural network with *lowest training error* during any round of BackProp. For that neural network, report the value of the *error function* on the training and test data, and the train and test *misclassification error rate. Tabulate and include those values in your report.*
- Evaluate each example in the trial data set on each neural network learned and *report the corresponding output classification.*

The ML Neural-Network Black-box

You can use an implementation of neural networks supplied with the programming homework files to help you debug your implementation.

Neural-Network Learning Code. The bash-shell script `learner_nnet_script` is a wrapper for the BackProp algorithm for learning neural networks. This is how it works. Say that given some training data, we want to learn a neural network, which is defined by matrices $\mathbf{W}_{r+1,r}$ corresponding to the weights on the connections between consecutive layers r and $r+1$. If the neural network has $H+2$ layers, where H is the number of hidden layers, $r=0$ corresponds to the input layer and $r=H+2$ corresponds to the output, then for each $r=0,\ldots,H+1$, $\mathbf{W}_{r+1,r}$ would have n_{r+1} rows and n_r columns, corresponding to the number of units in each layer, respectively. In this homework, we want to learn two neural networks, one using the transformed digit data and the other using the untransformed/original digit data.

To learn the first network for the transformed data, first copy the files `optdigits_tra_trans.dat` and `optdigits_cv_trans.dat`, corresponding to the training and test files for the transformed version of the digits data, into the files `D.dat` and `D_cv.dat`, respectively, and place them in the same directory as the neural network code provided. You then run the following bash-shell script as

```
./learn_nnet_script '[64 10]' '1000' '1' '1e-4' '1e-6'
```

which will produce the following files corresponding to a perceptron (i.e., two layer neural network corresponding to the input and output units, without any hidden layers) formed of 64 input units and 10 output units, obtained by running BackProp on the training data file `D.dat` using a *maximum of* 1000 *iterations*, a *learning rate/step size value* of 1, and a value of 10^{-4} as a stopping threshold on the *minimum change in the weight parameter values* between iterations, and a value of 10^{-6} as a stopping threshold on the *minimum change in the training error-function* values between iterations.

`best_nnet_W_1_0.dat`	`best_nnet_err_final.dat`
`last_nnet_W_1_0.dat`	`last_nnet_err_final.dat`
`train_err.dat`	`train_err_rate.dat`
`test_err.dat`	`test_err_rate.dat`

The first two rows above corresponds to the weight files for each consecutive layer and error-related information for the *best* and *last* neural network obtained during BackProp. The weight file `best_nnet_W_1_0.dat` contains the weights parameters between the first and second layer as a table with 10 rows and 65 columns ($64 + 1$ to account for the offset threshold parameter that is input to each unit of each layer). The error information file `best_nnet_err_final.dat` contains a single row of values of the *error function* on the training and test data, followed by the values of the training and test *error rates* of the neural network. The case for the file `last_nnet_W_1_0.dat` and `last_nnet_err_final.dat` is similar. (*You will not be using the files* `last_nnet_W_1_0.dat` *and* `last_nnet_err_final.dat` *in this homework.*) The last two rows of the table of files above corresponds to the training and test error function values and rates that the neural networks found at each iteration of BackProp achieve . Each one of those files is a sequence of values (written as a single column in the file) for each round of BackProp. You can use those values to plot the evolution of the respective error values during BackProp.

Neural-Network Classification Code. Given the weight-parameter files for a neural network obtained using the learning bash-shell script just described, we can use the neural network for classification. To do this, we first copy all the *inputs* of all the examples we want to classify in the file `X.dat`, one example per row, and one column per input feature in the file. Suppose the neural network architecture is a perceptron with 64 inputs units and 10 output units. Then copy the weight parameters into the file `W_1_0.dat`. (For example, if we want to classify using the best neural network found during BackProp in the example above, we would simply copy the content of the file `best_nnet_W_1_0.dat` into the file `W_1_0.dat`.) Then we execute the following bash-shell scripts

```
./classify_nnet_script '[64 10]'
```

which will produce the file `Y.dat` with the classification output that the neural network we are evaluating assigns to each of the examples in the file `X.dat`; the classification of each example in `X.dat` appears as a single column in `Y.dat`, in the same order as given in `X.dat`.

On the Neural-Network Black-box. The actual neural-network algorithms are implemented in a Matlab-like language called Octave. The wrapper assumes that you have installed either Octave or Matlab. [2] You need to edit the line in the script corresponding to the call to Matlab or Octave,

[2] GNU Octave is freely distributed software. See `http://www.gnu.org/software/octave/` for download information.

and replace it with the correct command call and path to the Matlab or Octave program installed in your system. (*The source code was not tested in Matlab; it may need minor modifications to run in that environment.*) Once you have specified a program and edited the script accordingly, the wrapper will call the corresponding program for you.

Support Vector Machines

You are now asked to evaluate the application of SVMs to the *original/untransformed* digit data using support vector machines (SVMs).

NOTE: You do not need to implement an SVM learning algorithm. You only need to use one already provided for you along with the programming homework files. The implementation is that of the SVM learning algorithm described in the **Support Vector Machines** *Subsection of the* **Learning Algorithms** *Section of this homework. You do not need to reimplement the algorithm but can do so if you wish, or you can implement another algorithm, for* **extra credit***!*

Model and Implementation Details. The following are experimental details on the application of the learning algorithm and the different SVM instances considered.

One-against-all Approach to Multiclass Problems: You will use a *one-against-all approach* to cast the L-class classification problem into L individual binary classification problems. For each class k, you will learn an SVM to discriminate between class k and the rest, where class k examples correspond to the positive examples ($y = +1$), while examples of the other classes are the negative examples ($y = -1$). Suppose that the SMV binary classifier learned this way for class k is $H_k(\mathbf{x}) = \text{sign}\left(f_k(\mathbf{x})\right)$, where $f_k(\mathbf{x}) = \sum_{l=1}^{m} \alpha_l(k) y^{(l)} K(\mathbf{x}^{(l)}, \mathbf{x}) + b(k)$. Then, we built an overall multi-class classifier H as

$$H(\mathbf{x}) \equiv H_{k^*}(\mathbf{x}) \text{ where } k^* \equiv \arg \max_{k=1,\ldots,L} f_k(\mathbf{x}) \ .$$

Kernel functions: Recall the following kernel functions.

- polynomial kernel of degree d: $K(\mathbf{x}, \mathbf{z}) = (1 + \mathbf{x} \cdot \mathbf{z})^d$
- Gaussian/RBF kernel with inverse-variance parameter γ: $K(\mathbf{x}, \mathbf{z}) = \exp\left(-\gamma \frac{1}{2} \|\mathbf{x} - \mathbf{z}\|^2\right)$

For this homework, you will considered the following specific instances of SVMs:

- polynomial kernels of three different degrees $d = 1$ (i.e., linear kernel), 4 and 10
- Gaussian/RBF kernel with three different inverse-variance parameter $\gamma = 0.1$, 0.027 and 0.001

Initialization and Stopping Conditions: Initialize the dual variables/example weights as $\boldsymbol{\alpha}^{(1)} = 0$. Run the SVM learning algorithm for a minimum of 15 iteration. Stop after a maximum of 5000 iterations or whenever the relative change in value of the dual objective function g evaluated at the dual variables $\boldsymbol{\alpha}^{(t)}$ and $\boldsymbol{\alpha}^{(t+1)}$ obtained at consecutive iteration t and $t + 1$ of the learning algorithm satisfies

$$\frac{g(\boldsymbol{\alpha}^{(t+1)}) - g(\boldsymbol{\alpha}^{(t)})}{g(\boldsymbol{\alpha}^{(t)})} \leq 10^{-3} \ .$$

Evaluation and Report. Perform the following steps, for each SVM instance listed above, using the *original/untransformed* digit data *only*.

- Report the training and test *misclassification error rate* for each SVM-based multiclass classifier H learned
- Also report the training and test *misclassification error rate* for each binary classifier SVM H_k learned for class k
- Evaluate each example in the trial data set on each SVM learned and *report the corresponding output classification*

The ML SVM Black-box

SVM Learning Code. The bash-shell script `learner_svm_script` is a wrapper for the implementation of a projected-gradient learning algorithm for multiclass SVMs using a one-against-all approach. The code also evaluates the training and test misclassification error rate using the train and test sets given as input. Finally, the script also outputs the classification of a given trial set of examples. *All relevant information is sent to standard output (i.e.,* `stdout`*).*

To use the script, first copy the training, test and trial data sets into the files `D.dat`, `D_cv.dat` and `D_trial.dat`, respectively. Then you may need to modify the file of the corresponding kernel function to set the corresponding parameter. For example, if you want to use a polynomial kernel, then you may need to modify the *polynomial-degree parameter*: in the file `kernel_poly.m`, change the line

```
d = 1;
```

to the appropriate degree. (The line above corresponds to a linear kernel.) So for a quadratic kernel, that line should be

```
d = 2;
```

which changes the degree of the polynomial kernel to 2. Similarly, to use a Gaussian kernel, you must change the following line in file `kernel_gauss.m`,

```
inv_var = 2.7e-2;
```

corresponding to the *inverse-variance parameter* accordingly.

Having copied the respective data file and adapted the appropriate kernel function, executing, for instance, the bash-shell script

```
./learn_svm_script 'poly' '5000' '15' '1e-3' '10'
```

will output all the relevant results of running a projected-gradient learning algorithm, using a polynomial kernel, for a maximum of 5000 iterations, a minimum of 15 iterations, a value of 10^{-3} for the minimum relative improvement in the dual objective function for the SVM, and for a problem with 10 output labels. To use the Gaussian kernel instead of the polynomial kernel in the learning script, replace the first script input, `'poly'`, by the string `'gauss'` in the command line given above.

On the SVM Black-box. Just like for Neural Networks, the actual SVM algorithm is implemented in a Matlab-like language called Octave. The wrapper assumes that you have installed either Octave or Matlab. [3] You need to edit the line in the script corresponding to the call to

[3]GNU Octave is freely distributed software. See `http://www.gnu.org/software/octave/` for download information.

Matlab or Octave, and replace it with the correct command call and path to the Matlab or Octave program installed in your system. (*The source code was not tested in Matlab; it may need minor modifications to run in that environment.*) Once you have specified a program and edited the script accordingly, the wrapper will call the corresponding program for you.

What to Turn In

You need to submit the following.

1. A **written report** (*in PDF*) that includes the information/plots/graphs/images requested above along with a brief discussion of the results. In your discussion, compare and contrast the different classifier models and learning algorithms.

2. All your **code and executable** (as a tared-and-gziped compressed file), with instructions on how to run your program. A platform-independent executable is preferred; otherwise, also provide instructions on how to compile your program. Please use standard tools/compilers/etc. generally available in most popular platforms.

Collaboration Policy: *It is OK to discuss the homework with your peers, but each student must write and turn in his/her own report, code, etc. based on his/her own work.*

Learning Algorithms

Let $D = (\mathbf{x}^{(1)}, y^{(1)}), \ldots, (\mathbf{x}^{(m)}, y^{(m)})$ be a dataset of m-examples.

Neural Networks

For the neural network classifiers considered here, we assume the output is in $y \in \{0,1\}$ for binary classification, and $\mathbf{y} \in \{\mathbf{u} \in \{0,1\}^L | u_k = 1 \text{ and for all } l \neq k, u_l = 0\}$ for L-class classification, where the output vector $\mathbf{y}$ encodes the output class label k by having exactly one 1 in output dimension k (i.e., $y_k = 1$ and $y_l = 0$ for all $l \neq k$).

Let r index the neural network layers, where $r = 0$ is the *input* layer, $r = 1, \ldots, H$ is one of the H *hidden* layers and $r = H+1$ is the *output* layer. Let n_r be the number of units in layer r, so that $n_0 = n$ is the number of features and $n_{H+1} = d$ is the number of outputs.

ForwardProp. Given an input $\mathbf{x} = (x_1, x_2, \ldots, x_n) \in \mathbb{R}^n$ composed of the attribute values for n features, the process to evaluate the output of the d-dimensional output

$$\mathbf{f}(\mathbf{x}; \mathbf{W}) = (f_1(\mathbf{x}; \mathbf{W}), \ldots, f_d(\mathbf{x}; \mathbf{W})))^T$$

of a neural network is as follows. Initialize

$$o_0 \leftarrow -1$$

and for all input units $i = 1, \ldots, n_0$,

$$o_i \leftarrow x_i \ .$$

For all non-input layers $r = 1, \ldots, H+1$, for all units j in layer r, the output of neural network unit j is

$$o_j \leftarrow \text{sigmoid}(z_j) \ .$$

where [4]

$$z_j \leftarrow \sum_{i=0}^{n_{r-1}} o_i w_{ij}$$

and the output of the sigmoid function (which is used as a heuristic approximation to the threshold function $t(x) = \mathbf{1}[x > 0]$) is

$$\text{sigmoid}(z_j) = (1 + \exp(-z_j))^{-1} \ .$$

The output of the neural network $\mathbf{f}(\mathbf{x}; \mathbf{W})$ is given by the output of the output units i (in layer $r = H+1$) for that given input $\mathbf{x}$ and weights $\mathbf{W}$: i.e., $f_i(\mathbf{x}; \mathbf{W}) = o_i$.

BackProp. Let $\lambda > 0$ be the value of the learning rate/step size and T the maximum number of rounds of backpropagation. At every round $t = 1, \ldots, T$, for all non-input layers $r = 1, \ldots, H+1$, for all units j in layer r, and all units i in layer $r-1$, the neural network weight update is

$$w_{ij}^{(t+1)} \leftarrow w_{ij}^{(t)} - \lambda \frac{1}{m} \sum_{l=1}^{m} o_i^{(l,t)} \delta_j^{(l,t)}$$

[4] Note that for convenience we are denoting the threshold parameter of unit j as $w_{0j} \equiv b_j$.

where for all examples l, $o_i^{(l,t)}$ is the output of unit i when evaluated using forward propagation on example $\mathbf{x}^{(l)}$ as input to the neural network with the weights $\mathbf{W}^{(t)}$ at round t of backpropagation, and

$$\delta_j^{(l,t)} \leftarrow o_j^{(l,t)}(1-o_j^{(l,t)}) \times \begin{cases} \left(o_j^{(l,t)} - y_j^{(l)}\right), & \text{if } j \text{ is an output unit,} \\ \sum_{k=1}^{n_{r+1}} w_{jk}^{(t)} \delta_k^{(l,t)}, & \text{otherwise.} \end{cases}$$

Support Vector Machines

Here is the description of a simple *projected gradient method* to learn the support vector weights $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_m)^T$ with $\alpha_l \geq 0$ for all $l = 1, \ldots, m$. We assume the output of each example $y \in \{-1, +1\}$. Let $\mathbf{K}$ denote the *kernel matrix*: i.e., for all pair of examples l and s, the entry

$$K_{l,s} \equiv K(\mathbf{x}^{(l)}, \mathbf{x}^{(s)}) .$$

Let $\mathbf{Y} \equiv \operatorname{diag}(\mathbf{y})$ denote the $m \times m$ diagonal matrix with entries

$$Y_{l,s} \equiv \begin{cases} y^{(l)}, & \text{if } l = s, \\ 0, & \text{otherwise} \end{cases}$$

(i.e., the output $\mathbf{y}$ is along the diagonal). Let the matrix $\mathbf{Q} \equiv \mathbf{Y}\ \mathbf{K}\ \mathbf{Y}$, having entries

$$Q_{ls} \equiv y^{(l)} K_{l,s} y^{(s)}$$

for all pair of examples indexed by l and s of the dataset D. The SVM dual function can then be expressed as

$$g(\boldsymbol{\alpha}) \equiv \mathbf{1}^T\boldsymbol{\alpha} - \frac{1}{2}\boldsymbol{\alpha}^T\mathbf{Q}\boldsymbol{\alpha} = \sum_{l=1}^{m} \alpha_l - \frac{1}{2}\sum_{l=1}^{m}\sum_{s=1}^{m} y_l \alpha_l y_s \alpha_s K_{ls} .$$

Denote the gradient of the SVM dual function by

$$\boldsymbol{\nabla} g(\boldsymbol{\alpha}) \equiv (\nabla_1 g(\boldsymbol{\alpha}), \ldots, \nabla_m g(\boldsymbol{\alpha}))^T = \mathbf{1} - \mathbf{Q}\boldsymbol{\alpha} ,$$

with

$$\nabla_l g(\boldsymbol{\alpha}) \equiv \frac{\partial g(\boldsymbol{\alpha})}{\partial \alpha_l} = 1 - y^{(l)} \sum_{s=1}^{m} y^{(s)} \alpha_s K(x^{(l)}, x^{(s)}) ,$$

for all examples l.

(Projected) Gradient Ascent for SVMs. Initialize $\alpha_l^{(1)} \leftarrow 0$ for all examples $l = 1, \ldots, m$. Denote the time dependent step size by $\lambda^{(t)} > 0$. At every round $t = 1, \ldots, T$, we perform the following *(projected) gradient update*:

1. *Compute gradient*: $\boldsymbol{\nabla}^{(t)} \leftarrow \mathbf{1} - \mathbf{Q}\boldsymbol{\alpha}^{(t)}$
2. *Compute projected gradient*: $\boldsymbol{\nabla}^{P^{(t)}} \leftarrow \mathbf{Proj}\left(\boldsymbol{\nabla}^{(t)}; \boldsymbol{\alpha}^{(t)}\right)$

3. *Compute step size*: $\lambda^{(t)} \leftarrow \max\left(\min\left(\frac{\left(\boldsymbol{\nabla}^{(t)}\right)^T \boldsymbol{\nabla}^{P(t)}}{\left(\boldsymbol{\nabla}^{P(t)}\right)^T \mathbf{Q}\boldsymbol{\nabla}^{P(t)}}, \lambda_{\text{up}}^{(t)}\right), \lambda_{\text{low}}^{(t)}\right)$ where

$$\lambda_{\text{low}}^{(t)} \equiv \min\left(\min_{l:\nabla_l^{P(t)}>0} \frac{-\alpha_l^{(t)}}{\nabla_l^{P(t)}}, \min_{l:\nabla_l^{P(t)}<0} \frac{C-\alpha_l^{(t)}}{\nabla_l^{P(t)}}\right)$$

and

$$\lambda_{\text{up}}^{(t)} \equiv \max\left(\max_{l:\nabla_l^{P(t)}<0} \frac{-\alpha_l^{(t)}}{\nabla_l^{P(t)}}, \max_{l:\nabla_l^{P(t)}>0} \frac{C-\alpha_l^{(t)}}{\nabla_l^{P(t)}}\right)$$

4. *Compute new example weights*: $\boldsymbol{\alpha}^{(t+1)} \leftarrow \min\left(\max\left(\boldsymbol{\alpha}^{(t)} + \lambda^{(t)}\boldsymbol{\nabla}^{P(t)}, 0\right), C\right)$ where the min and max are performed element-wise

5. *Re-enforce example weights' constraints*: $\boldsymbol{\alpha}^{(t+1)} \leftarrow \textsc{EnforceConstraints}(\boldsymbol{\alpha}^{(t+1)})$
[*The only reason for this step is to maintain numerical accuracy!*]

Once we are done iterating, given $\boldsymbol{\alpha}^{(T)}$ (i.e., the last values found for the example weights/dual variables) and its respective gradient $\boldsymbol{\nabla}^{(T)}$, then we can *set the offset/threshold value*

$$b^{(T)} \leftarrow \frac{(C\mathbf{1} - \boldsymbol{\alpha}^{(T)})^T \boldsymbol{\nabla}^{(T)}}{(C\mathbf{1} - \boldsymbol{\alpha}^{(T)})^T \mathbf{y}} .$$

The Projection Function: The function $\mathbf{Proj}(\boldsymbol{\nabla}; \boldsymbol{\alpha})$, which outputs the projected version $\boldsymbol{\nabla}^P$ of the gradient $\boldsymbol{\nabla}$ with respect to $\boldsymbol{\alpha}$, is recursively defined as

$$\boldsymbol{\nabla}^P \equiv \mathbf{Proj}(\boldsymbol{\nabla}; \boldsymbol{\alpha}) \equiv \begin{cases} \overline{\boldsymbol{\nabla}}, & \text{if } \overline{\boldsymbol{\nabla}} = \widetilde{\boldsymbol{\nabla}}, \\ \mathbf{Proj}(\widetilde{\boldsymbol{\nabla}}; \boldsymbol{\alpha}), & \text{otherwise,} \end{cases}$$

where [5]

$$\overline{\boldsymbol{\nabla}} \equiv \boldsymbol{\nabla} - \left(\frac{1}{m}\mathbf{y}^T\boldsymbol{\nabla}\right)\mathbf{y}$$

and $\widetilde{\boldsymbol{\nabla}}$ is such that, for all l,

$$\widetilde{\nabla}_l \equiv \begin{cases} 0, & \text{if } (\alpha_l = 0 \text{ and } \overline{\nabla}_l \le 0) \text{ or } (\ \alpha_l = C \text{ and } \overline{\nabla}_l \ge 0), \\ \overline{\nabla}_l, & \text{otherwise.} \end{cases}$$

The $\textsc{EnforceConstraints}$ ***Procedure***: The following pseudocode implements the procedure $\textsc{EnforceConstraints}(\boldsymbol{\alpha})$, which on input $\boldsymbol{\alpha}$ outputs $\boldsymbol{\alpha}^{\text{valid}}$ further enforcing the constraints $\mathbf{y}^T\boldsymbol{\alpha}^{\text{valid}} = 0$ and $0 \le \boldsymbol{\alpha}^{\text{valid}} \le C$, which may have been violated because of numerical errors accumulated during previous operations.

1. Evaluate the possible (signed) offset violation $\beta \equiv \mathbf{y}^T\boldsymbol{\alpha}$

[5]Note that

$$\mathbf{y}^T\overline{\boldsymbol{\nabla}} = \mathbf{y}^T\left(\boldsymbol{\nabla} - \left(\frac{1}{m}\mathbf{y}^T\boldsymbol{\nabla}\right)\mathbf{y}\right) = \mathbf{y}^T\boldsymbol{\nabla} - \left(\frac{1}{m}\mathbf{y}^T\boldsymbol{\nabla}\right)\mathbf{y}^T\mathbf{y} = \mathbf{y}^T\boldsymbol{\nabla} - \left(\frac{1}{m}\mathbf{y}^T\boldsymbol{\nabla}\right)m = 0.$$

2. Let $\boldsymbol{\alpha}^{\text{valid}}$ be such that for all examples l,

$$\alpha_l^{\text{valid}} \leftarrow \alpha_l - y^{(l)}\beta \, \mathbf{1}\left[y^{(l)}\beta < 0\right] \left(\frac{\mathbf{1}\left[y^{(l)} = 1\right]}{m_{\text{sv}}^{+}} + \frac{\mathbf{1}\left[y^{(l)} = -1\right]}{m_{\text{sv}}^{-}} \right)$$

if $\alpha_l > 0$, and $\alpha_l^{\text{valid}} \leftarrow \alpha_l = 0$, otherwise; where

$$m_{\text{sv}}^{+} \equiv \sum_{l=1}^{m} \mathbf{1}\left[\alpha_l > 0, y^{(l)} = 1\right] \text{ and } m_{\text{sv}}^{-} \equiv \sum_{l=1}^{m} \mathbf{1}\left[\alpha_l > 0, y^{(l)} = -1\right]$$

denote the number of positive and negative support vectors with respect to $\boldsymbol{\alpha}$, respectively, and assuming $\frac{0}{0} = 0$

3. If $\alpha_l^{\text{valid}} > C$ for some example l, then rescale $\boldsymbol{\alpha}^{\text{valid}}$ so that $0 \leq \boldsymbol{\alpha}^{\text{valid}} \leq C$ by resetting

$$\alpha_l^{\text{valid}} \leftarrow C \times \frac{\alpha_l^{\text{valid}}}{\max_l \alpha_l^{\text{valid}}} \ .$$

Appendix

Derivation of BackProp

Define the error function as

$$E(\mathbf{W}; D) \equiv \frac{1}{m} \sum_{l=1}^{m} E(\mathbf{W}; (\mathbf{x}^{(l)}, \mathbf{y}^{(l)}))$$

where

$$E(\mathbf{W}; (\mathbf{x}^{(l)}, \mathbf{y}^{(l)})) \equiv \sum_{s=1}^{d} \frac{1}{2} \left(f_s(\mathbf{x}^{(l)}; \mathbf{W}) - y_s^{(l)} \right)^2$$

and $f_s(\mathbf{x}^{(l)}; \mathbf{W})$ is the output of the neural network output unit s when evaluated with input $\mathbf{x}^{(l)}$ and weights $\mathbf{W}$. To simplify notation, let $E \equiv E(\mathbf{W}; D)$ and $E^{(l)} \equiv E(\mathbf{W}; (\mathbf{x}^{(l)}, \mathbf{y}^{(l)}))$ so that we can express $E = \frac{1}{m} \sum_{l=1}^{m} E^{(l)}$. Similarly, let $f_s^{(l)} \equiv f_s(\mathbf{x}^{(l)}; \mathbf{W})$ so that we can express $E^{(l)} = \sum_{s=1}^{d} \frac{1}{2} \left(f_s^{(l)} - y_s^{(l)} \right)^2$ and $E = \frac{1}{m} \sum_{l=1}^{m} \sum_{s=1}^{d} \frac{1}{2} \left(f_s^{(l)} - y_s^{(l)} \right)^2$.

Also, let $o_j^{(l)} \equiv o_j(\mathbf{x}^{(l)}; \mathbf{W})$ be the output of unit j of a neural network with weights $\mathbf{W}$ when evaluated on input $\mathbf{x}^{(l)}$. For the "dummy" threshold-related unit, we have $o_0(\mathbf{x}^{(l)}; \mathbf{W}) = -1$, for all input units j, we have

$$o_j(\mathbf{x}^{(l)}; \mathbf{W}) = x_j^{(l)} ,$$

for all non-input units j in layer $r > 0$, we have

$$o_j(\mathbf{x}^{(l)}; \mathbf{W}) = \text{sigmoid}(z_j(\mathbf{x}^{(l)}; \mathbf{W}))$$

where $z_j(\mathbf{x}^{(l)}; \mathbf{W}) \equiv \sum_{i=0}^{n_{r-1}} o_i(\mathbf{x}^{(l)}; \mathbf{W}) \, w_{ij}$. To reduce notation, let $z_j^{(l)} \equiv z_j(\mathbf{x}^{(l)}; \mathbf{W})$ so that we can simply express $o_j^{(l)} = \text{sigmoid}(z_j^{(l)})$ and $z_j^{(l)} = \sum_{i=0}^{n_{r-1}} o_i^{(l)} w_{ij}$ for all non-input units j (in layer $r > 0$).

We then have

$$\frac{\partial E}{\partial w_{ij}} = \frac{1}{m} \sum_{l=1}^{m} \frac{\partial E^{(l)}}{\partial w_{ij}}$$

and

$$\frac{\partial E^{(l)}}{\partial w_{ij}} = \frac{\partial z_j^{(l)}}{\partial w_{ij}} \frac{\partial E^{(l)}}{\partial z_j^{(l)}} .$$

For all non-input units j, define

$$\delta_j^{(l)} \equiv \frac{\partial E^{(l)}}{\partial z_j^{(l)}} .$$

Because

$$\frac{\partial z_j^{(l)}}{\partial w_{ij}} = \frac{\partial}{\partial w_{ij}} \left[\sum_{v=0}^{n_{r-1}} o_v^{(l)} w_{vj} \right] = o_i^{(l)}$$

we can express

$$\frac{\partial E^{(l)}}{\partial w_{ij}} = o_i^{(l)} \delta_j^{(l)} .$$

Now, for all non-input units j, we have

$$\delta_j^{(l)} = \frac{do_j^{(l)}}{dz_j^{(l)}} \frac{\partial E^{(l)}}{\partial o_j^{(l)}} \ .$$

Using the expression for the derivative of the sigmoid function, we have

$$\frac{do_j^{(l)}}{dz_j^{(l)}} = o_j^{(l)}(1 - o_j^{(l)}) \ .$$

In addition, for all hidden layers $r = 1, \ldots, H$, and all units j in layer r, we have

$$\frac{\partial E^{(l)}}{\partial o_j^{(l)}} = \sum_{k=1}^{n_{r+1}} \frac{\partial z_k^{(l)}}{\partial o_j^{(l)}} \frac{\partial E^{(l)}}{\partial z_k^{(l)}} \ .$$

Because we have

$$\frac{\partial z_k^{(l)}}{\partial o_j^{(l)}} = \frac{\partial}{\partial o_j^{(l)}} \left[\sum_{v=0}^{n_r} o_v^{(l)} w_{vk} \right] = w_{jk}$$

and by the definition of $\delta_k^{(l)}$, we can express

$$\frac{\partial E^{(l)}}{\partial o_j^{(l)}} = \sum_{k=1}^{n_{r+1}} w_{jk} \delta_k^{(l)} \ .$$

On the other hand, for all output units j, we have $f_j^{(l)} = o_j^{(l)}$, so that

$$\frac{\partial E^{(l)}}{\partial o_j^{(l)}} = \frac{\partial}{\partial o_j^{(l)}} \left[\sum_{s=1}^{n} \frac{1}{2} \left(o_s^{(l)} - y_s^{(l)} \right)^2 \right] = \left(o_j^{(l)} - y_j^{(l)} \right) \ .$$

Putting everything together, we obtain the partial derivatives forming the gradient used for the backpropagation update rule: for all non-input units j in layer r and all units i in layer $r - 1$, we have

$$\frac{\partial E}{\partial w_{ij}} = \frac{1}{m} \sum_{l=1}^{m} o_i^{(l)} \delta_j^{(l)} \ .$$

where

$$\delta_j^{(l)} = o_j^{(l)}(1 - o_j^{(l)}) \times \begin{cases} \left(o_j^{(l)} - y_j^{(l)} \right), & \text{if } j \text{ is an output unit,} \\ \sum_{k=1}^{n_{r+1}} w_{jk} \delta_k^{(l)}, & \text{otherwise.} \end{cases}$$

Derivation of Projected Gradient Ascent for SVMs

Recall that the update rule is

$$\alpha_l^{(t+1)} \leftarrow \min\left(\max\left(\alpha_l^{(t)} + \lambda^{(t)} \mathrm{Proj}_l\left(\boldsymbol{\nabla} g(\boldsymbol{\alpha}^{(t)})\right), 0\right), C\right) .$$

The SVM dual function

$$g(\boldsymbol{\alpha}) \equiv \mathbf{1}^T \boldsymbol{\alpha} - \frac{1}{2}\boldsymbol{\alpha}^T \mathbf{Q} \boldsymbol{\alpha} = \sum_{l=1}^{m} \alpha_l - \frac{1}{2}\sum_{l=1}^{m}\sum_{s=1}^{m} \alpha_l y^{(l)} K(\mathbf{x}^{(l)}, \mathbf{x}^{(s)}) y^{(s)} \alpha_s$$

where the matrix $\mathbf{Q} \equiv \mathrm{diag}(\mathbf{y})\ \mathbf{K}\ \mathrm{diag}(\mathbf{y})$: i.e., the entries

$$Q_{ls} \equiv y^{(l)} K(\mathbf{x}^{(l)}, \mathbf{x}^{(s)}) y^{(s)}$$

for all pair of examples l and s of the dataset D. The gradient of the SVM dual function is

$$\boldsymbol{\nabla} g(\boldsymbol{\alpha}^{(t)}) \equiv \left(\nabla_1 g(\boldsymbol{\alpha}^{(t)}), \ldots, \nabla_m g(\boldsymbol{\alpha}^{(t)})\right)^T ,$$

with

$$\nabla_l g(\boldsymbol{\alpha}) \equiv \frac{\partial g(\boldsymbol{\alpha})}{\partial \alpha_l} = 1 - y^{(l)} \sum_{s=1}^{m} y^{(s)} \alpha_s K(\mathbf{x}^{(l)}, \mathbf{x}^{(s)})$$

for all examples l. The l-component of the gradient of $g(\boldsymbol{\alpha})$ *projected* into the space $\mathbf{y}^T \boldsymbol{\alpha} = \sum_{s=1}^{m} y^{(s)} \alpha_s = 0$ is

$$\nabla_l^P g(\boldsymbol{\alpha}) \equiv \mathrm{Proj}_l\left(\boldsymbol{\nabla} g(\boldsymbol{\alpha}); \boldsymbol{\alpha}\right) \equiv \nabla_l g(\boldsymbol{\alpha}) - y^{(l)} \frac{1}{m} \sum_{s=1}^{m} y^{(s)} \nabla_s g(\boldsymbol{\alpha}) .$$

so that the following *invariant* is maintained by $\boldsymbol{\alpha}^{(t)}$ at all rounds $t = 1, \ldots, T$: $\mathbf{y}^T \boldsymbol{\alpha}^{(t)} = 0$; said differently, the condition

$$\mathbf{y}^T \boldsymbol{\nabla}^P g(\boldsymbol{\alpha}^{(t)}) = \sum_{l=1}^{m} y^{(l)} \mathrm{Proj}_l\left(\boldsymbol{\nabla} g(\boldsymbol{\alpha}^{(t)}); \boldsymbol{\alpha}^{(t)}\right) = 0$$

holds for all rounds $t = 1, \ldots, T$ of (projected) gradient ascent.

Then, the following holds, assuming the running invariant $\mathbf{y}^T \boldsymbol{\alpha}^{(t+1)} = 0$:

$$\mathbf{y}^T \left(\boldsymbol{\alpha}^{(t)} + \lambda \boldsymbol{\nabla}^{P(t)},\right) = \mathbf{y}^T \boldsymbol{\alpha}^{(t)} + \lambda \mathbf{y}^T \boldsymbol{\nabla}^{P(t)} = 0 + \lambda \times 0 = 0 .$$

The optimal step size, which is the result of maximizing $g(\boldsymbol{\alpha} + \lambda \boldsymbol{\nabla}^P)$ with respect to λ, can then be expressed as [6]

$$\lambda \leftarrow \max\left(\min\left(\frac{\boldsymbol{\nabla}^T \boldsymbol{\nabla}^P}{\left(\boldsymbol{\nabla}^P\right)^T \mathbf{Q} \boldsymbol{\nabla}^P}, \lambda_{\mathrm{up}}\right), \lambda_{\mathrm{low}}\right)$$

[6] Note that $\lambda_{\mathrm{low}} \leq 0 \leq \lambda_{\mathrm{up}}$ and $\left(\boldsymbol{\nabla}^P\right)^T \mathbf{Q} \boldsymbol{\nabla}^P \geq 0$, so $\lambda \geq 0$ iff $\boldsymbol{\nabla}^T \boldsymbol{\nabla}^P \geq 0$. One would expect $\boldsymbol{\nabla}^T \boldsymbol{\nabla}^P \geq 0$ because $\boldsymbol{\nabla}^P$ is a (recursive) projection of $\boldsymbol{\nabla}$ into a subspace such that $\mathbf{y}^T \boldsymbol{\nabla}^P = 0$. Hence, we expect $\lambda \geq 0$, in which case the lower bound λ_{low} is vacuous.

where the bounds

$$\lambda_{\text{low}} \equiv \min\left(\min_{l\,:\,\nabla^P_l>0} \frac{-\alpha_l}{\nabla^P_l}, \min_{l\,:\,\nabla^P_l<0} \frac{C-\alpha_l}{\nabla^P_l}\right)$$

and

$$\lambda_{\text{up}} \equiv \max\left(\max_{l\,:\,\nabla^P_l<0} \frac{-\alpha_l}{\nabla^P_l}, \max_{l\,:\,\nabla^P_l>0} \frac{C-\alpha_l}{\nabla^P_l}\right) .$$

The time dependent step size $\lambda^{(t)}$ is the result of the assignment above evaluated in terms of $\boldsymbol{\alpha}^{(t)}$ and $\boldsymbol{\nabla}^{(t)}$.

Finally, the procedure $\boldsymbol{\alpha}^{\text{valid}} \leftarrow \textsc{EnforceConstraints}(\boldsymbol{\alpha})$ enforces the constraints on $\boldsymbol{\alpha}$. That it enforces the box constraints $0 \leq \boldsymbol{\alpha}^{\text{valid}} \leq C$ is easy to see. It enforces the constraint $\mathbf{y}^T\boldsymbol{\alpha}^{\text{valid}} = 0$ because

$$\begin{aligned}
\mathbf{y}^T\boldsymbol{\alpha}^{\text{valid}} =& \sum_{l=1}^{m} y^{(l)}\alpha_l^{\text{valid}} \\
=& \sum_{l\,:\,\alpha_l>0} y^{(l)}\left(\alpha_l - y^{(l)}\beta\,\mathbf{1}\left[y^{(l)}\beta<0\right]\left(\frac{\mathbf{1}\left[y^{(l)}=1\right]}{m^+_{\text{sv}}} + \frac{\mathbf{1}\left[y^{(l)}=-1\right]}{m^-_{\text{sv}}}\right)\right) \\
=& \sum_{l\,:\,\alpha_l>0} y^{(l)}\alpha_l - \sum_{l\,:\,\alpha_l>0} y^{(l)}y^{(l)}\beta\,\mathbf{1}\left[y^{(l)}\beta<0\right]\left(\frac{\mathbf{1}\left[y^{(l)}=1\right]}{m^+_{\text{sv}}} + \frac{\mathbf{1}\left[y^{(l)}=-1\right]}{m^-_{\text{sv}}}\right) \\
=& \sum_{l=1}^{m} y^{(l)}\alpha_l - \beta\sum_{l=1}^{m}\mathbf{1}\left[y^{(l)}\beta<0\right]\left(\frac{\mathbf{1}\left[y^{(l)}=1\right]}{m^+_{\text{sv}}} + \frac{\mathbf{1}\left[y^{(l)}=-1\right]}{m^-_{\text{sv}}}\right) \\
=& \beta - \beta\left(\sum_{l\,:\,\alpha_l>0}\mathbf{1}\left[y^{(l)}\beta<0\right]\frac{\mathbf{1}\left[y^{(l)}=1\right]}{m^+_{\text{sv}}} + \sum_{l\,:\,\alpha_l>0}\mathbf{1}\left[y^{(l)}\beta<0\right]\frac{\mathbf{1}\left[y^{(l)}=-1\right]}{m^-_{\text{sv}}}\right) \\
=& \beta - \beta\left(\frac{1}{m^+_{\text{sv}}}\sum_{l\,:\,\alpha_l>0}\mathbf{1}\left[y^{(l)}\beta<0, y^{(l)}=1\right] + \frac{1}{m^-_{\text{sv}}}\sum_{l\,:\,\alpha_l>0}\mathbf{1}\left[y^{(l)}\beta<0, y^{(l)}=-1\right]\right) \\
=& \beta - \beta\left(\frac{1}{m^+_{\text{sv}}}\sum_{l\,:\,\alpha_l>0}\mathbf{1}\left[\beta<0, y^{(l)}=1\right] + \frac{1}{m^-_{\text{sv}}}\sum_{l\,:\,\alpha_l>0}\mathbf{1}\left[\beta>0, y^{(l)}=-1\right]\right) \\
=& \beta - \beta\left(\mathbf{1}\left[\beta<0\right]\frac{1}{m^+_{\text{sv}}}\sum_{l\,:\,\alpha_l>0}\mathbf{1}\left[y^{(l)}=1\right] + \mathbf{1}\left[\beta>0\right]\frac{1}{m^-_{\text{sv}}}\sum_{l\,:\,\alpha_l>0}\mathbf{1}\left[y^{(l)}=-1\right]\right) \\
=& \beta - \beta\left(\mathbf{1}\left[\beta<0\right] + \mathbf{1}\left[\beta>0\right]\right) \\
=& \beta - \beta(1 - \mathbf{1}\left[\beta=0\right]) = \beta\,\mathbf{1}\left[\beta=0\right] = 0\ .
\end{aligned}$$

The expression of the assignment to the SVM offset/threshold parameter is mostly a consequence of the requirement imposed in the SVM conditions that SVM function $f(\mathbf{x}^{(l)}) = y^{(l)}$ for all support vector examples $\mathbf{x}_l$ (i.e., all l such that $\alpha_l > 0$) (Note that the final SVM binary classification is $\text{sign}(f(\mathbf{x}))$.) More formally, let $\text{SV}(\boldsymbol{\alpha}) \equiv \{l|\alpha_l > 0\}$ be the set of indexes to the support vectors. The first order optimality conditions (i.e., KKT conditions) require that the optimal values of the parameters satisfy the condition: for all $l \in \mathrm{SV}(\boldsymbol{\alpha})$ such that $\alpha_l < C$, we have

$$y^{(l)} \left(\sum_{s=1}^{m} \alpha_s y^{(s)} K_{s,l} + b \right) = 1$$

which, solving for the term involving b, we can express

$$y^{(l)} b = 1 - \sum_{s=1}^{m} \alpha_s Q_{s,l} \; .$$

At this point, in theory, we can just pick one of the support vectors and solve for the value b. To deal with potential numerical problems, we will use an expression that involves all the support vectors. In particular, we have that for all $l \in \mathrm{SV}(\boldsymbol{\alpha})$ we can equivalently express the conditions above as

$$(C - \alpha_l) y^{(l)} b = (C - \alpha_l) \left(1 - \sum_{s=1}^{m} \alpha_s Q_{s,l} \right) \; .$$

and summing at both sides over $l \in \mathrm{SV}(\boldsymbol{\alpha})$, we obtain

$$b \sum_{l \in \mathrm{SV}(\boldsymbol{\alpha})} (C - \alpha_l) y^{(l)} = \sum_{l \in \mathrm{SV}(\boldsymbol{\alpha})} (C - \alpha_l) \left(1 - \sum_{s=1}^{m} \alpha_s Q_{s,l} \right)$$

and finally

$$b = \frac{\sum_{l \in \mathrm{SV}(\boldsymbol{\alpha})} (C - \alpha_l) \left(1 - \sum_{s=1}^{m} \alpha_s Q_{s,l}\right)}{\sum_{l \in \mathrm{SV}(\boldsymbol{\alpha})} (C - \alpha_l) y^{(l)}} \; .$$

Note that in theory, we have that the denominator at the optimal values evaluates to $\sum_{l \in \mathrm{SV}(\boldsymbol{\alpha})} (C - \alpha_l) y^{(l)} = C \sum_{l \in \mathrm{SV}(\boldsymbol{\alpha})} y^{(l)}$ because of another first order optimality condition: $\sum_{l \in \mathrm{SV}(\boldsymbol{\alpha})} \alpha_l y^{(l)} = 0$. So, if we have exactly the same number of positive and negative support vectors, then we have that the denominator is zero. In that case, we can just let

$$b = \frac{1}{|\{s | 0 < \alpha_s < C\}|} \sum_{l \in \{s | 0 < \alpha_s < C\}} y^{(l)} \left(1 - \sum_{s=1}^{m} \alpha_s Q_{s,l} \right) \; .$$