CSE176 Introduction to Machine Learning
Fall semester 2025

Lab: gradient descent and linear models
Miguel Á. Carreira-Perpiñán

The objective of this lab is for you to explore the behavior, in Matlab, of gradient descent (GD) and stochastic gradient descent (SGD) for two linear models (for regression and for classification), and apply them to some datasets. The TA will first demonstrate the results of the algorithms on several datasets, and then you will replicate those results, and further explore the datasets with the algorithms.

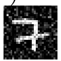
We provide you with the following:

- The scripts `lab07_linregr.m`, `lab07_linregr_MNIST.m`, `lab07_logregr.m`, `lab07_logregr_MNIST.m` set up the problem (toy dataset or MNIST) and plot various figures. The actual algorithms are implemented in the functions below.
- `linfexact.m`, `linfgd.m` and `linfsgd.m` train a linear regressor (from D dimensions to D' dimensions) by solving the normal equations, by gradient descent, or by stochastic gradient descent, respectively.
- `slinfgd.m` and `slinfsgd.m` train a logistic regressor for binary classification (from D dimensions to one output in $[0,1]$) by gradient descent or by stochastic gradient descent, respectively.
- See also the functions `linf.m`, `slinf.m` and `sigmoid.m`.

I Datasets

Firstly, construct your own toy datasets to visualize the result easily and understand the algorithm. Take the input instances $\{\mathbf{x}_n\}_{n=1}^N$ in $\mathbb{R}$ or $\mathbb{R}^2$ and the labels $\{y_n\}_{n=1}^N$ in $\mathbb{R}$ (regression) or $\{0,1\}$ (classification).

Then, try the MNIST dataset of handwritten digits, with instances $\mathbf{x} \in \mathbb{R}^D$ (where $D = 784$). For classification with logistic regression, use the labels $y_n \in \{0,\dots,9\}$ (for binary classification, pick instances from only two digit classes and use labels $y_n \in \{0,1\}$). For linear regression, create a ground-truth mapping $\mathbf{y} = \mathbf{f}(\mathbf{x}) = \mathbf{A}\mathbf{x}+\mathbf{b}$ as follows:

- A random mapping with output dimension D', e.g. take $\mathbf{A}_{D'\times D}$ and $\mathbf{b}_{D'\times 1}$ with elements in $\mathcal{N}(0,1)$.
- A mapping that rotates, scales, shifts and possibly clips the input image $\mathbf{x}$ and adds noise to it, e.g.  → . This makes it easy to visualize the result, since the desired output $\mathbf{f}(\mathbf{x})$ for an image $\mathbf{x}$ should look like $\mathbf{x}$ but transformed accordingly.

II (Stochastic) gradient descent for linear regression

Review Consider first the case of a single output dimension, $y \in \mathbb{R}$. Given a sample $\{(\mathbf{x}_n, y_n)\}_{n=1}^N$ with $\mathbf{x}_n \in \mathbb{R}^D$ and $y_n \in \mathbb{R}$, we solve a linear least-squares regression problem by minimizing:

$$E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^N (y_n - \mathbf{w}^T\mathbf{x}_n)^2 = \sum_{n=1}^N e(\mathbf{w};\mathbf{x}_n,y_n) \qquad \text{with} \qquad e(\mathbf{w};\mathbf{x}_n,y_n) = \frac{1}{2}(y_n - \mathbf{w}^T\mathbf{x}_n)^2$$

where we assume the last element of each $\mathbf{x}_n$ vector is equal to 1, so that w_D is the bias parameter (this simplifies the notation). $E(\mathbf{w})$ is a quadratic objective function over $\mathbf{w}$ with a unique minimizer that can be found in closed form by solving the normal equations (obtained by computing the gradient of E wrt $\mathbf{w}$, setting it to zero and solving for $\mathbf{w}$):

$$\mathbf{X}\mathbf{X}^T\mathbf{w}^* = \mathbf{X}\mathbf{y} \Leftrightarrow \mathbf{w}^* = (\mathbf{X}\mathbf{X}^T)^{-1}\mathbf{X}\mathbf{y}, \text{ where } \mathbf{X}_{D\times N} = (\mathbf{x}_1,\dots,\mathbf{x}_N),\ \mathbf{w}^*_{D\times 1} = (w_1,\dots,w_D)^T,\ \mathbf{y}_{N\times 1} = (y_1,\dots,y_N)^T.$$

This can be done with a routine for solving linear systems, e.g. in Matlab, use `linsolve` or the "\" operator. You can also use `inv` but this is numerically more costly and less stable.

The minimizer $\mathbf{w}^*$ of E can also be found iteratively by gradient descent (GD) and stochastic gradient descent (SGD), starting from an initial $\mathbf{w}$. These use the following updates (obtained by computing the gradient $\nabla E(\mathbf{w})$ of E wrt $\mathbf{w}$):

$$\text{GD: } \boldsymbol{\Delta}\mathbf{w} = \eta\sum_{n=1}^N (y_n - \mathbf{w}^T\mathbf{x}_n)\,\mathbf{x}_n \qquad\qquad \text{SGD: } \boldsymbol{\Delta}\mathbf{w} = \eta\sum_{n\in\mathcal{B}} (y_n - \mathbf{w}^T\mathbf{x}_n)\,\mathbf{x}_n$$

where η is the learning rate (step size) and $\mathcal{B}$ is a minibatch (i.e., a subset of $1 < |\mathcal{B}| < N$ points). For GD, at each iteration we set $\mathbf{w} \leftarrow \mathbf{w} + \boldsymbol{\Delta}\mathbf{w}$ with $\boldsymbol{\Delta}\mathbf{w} = -\eta\nabla E(\mathbf{w})$. For SGD, after each minibatch we set $\mathbf{w} \leftarrow \mathbf{w} + \boldsymbol{\Delta}\mathbf{w}$ with $\boldsymbol{\Delta}\mathbf{w} = -\eta\sum_{n\in\mathcal{B}}\nabla e(\mathbf{w};\mathbf{x}_n)$, and repeat over all minibatches to complete one iteration (one epoch), which passes over the whole data once. If the minibatches contain a single data point ($|\mathcal{B}| = 1$), then there are N minibatches and we do pure online learning. If there is a single minibatch containing all points ($|\mathcal{B}| = N$), we do pure batch learning (identical to GD). The fastest training occurs for an intermediate (usually small) minibatch size. It also helps to reorder at random (shuffle) the data points at the beginning of each epoch (so the minibatches vary, and are scanned in a different order), rather than using the same, original order at every epoch (helpful Matlab function: `randperm`).

Consider now the general case of several output dimensions and a sample $\{(\mathbf{x}_n, \mathbf{y}_n)\}_{n=1}^N$ with $\mathbf{x}_n \in \mathbb{R}^D$, $\mathbf{y}_n \in \mathbb{R}^{D'}$. This is equivalent to D' independent single-output regressions. The corresponding equations are as follows, where $\mathbf{X}_{D\times N} = (\mathbf{x}_1, \dots, \mathbf{x}_N)$, $\mathbf{Y}_{D'\times N} = (\mathbf{y}_1, \dots, \mathbf{y}_N)$, $\mathbf{W}_{D'\times D}$ and $\mathbf{w}_d^T$ is the dth row of $\mathbf{W}$:

$$\text{Objective function: } E(\mathbf{W}) = \frac{1}{2}\sum_{n=1}^N \|\mathbf{y}_n - \mathbf{W}\mathbf{x}_n\|^2 = \sum_{n=1}^N e(\mathbf{W}; \mathbf{x}_n, \mathbf{y}_n) \quad \text{with} \quad e(\mathbf{W}; \mathbf{x}_n, \mathbf{y}_n) = \frac{1}{2}\sum_{d=1}^{D'} (y_{nd} - \mathbf{w}_d^T\mathbf{x}_n)^2$$

$$\text{Gradient: } \frac{\partial E}{\partial \mathbf{W}} = -\sum_{n=1}^N (\mathbf{y}_n - \mathbf{W}\mathbf{x}_n)\,\mathbf{x}_n^T \qquad \text{GD: } \Delta\mathbf{W} = \eta\sum_{n=1}^N (\mathbf{y}_n - \mathbf{W}\mathbf{x}_n)\,\mathbf{x}_n^T \qquad \text{SGD: } \Delta\mathbf{W} = \eta\sum_{n\in\mathcal{B}} (\mathbf{y}_n - \mathbf{W}\mathbf{x}_n)\,\mathbf{x}_n^T$$

$$\text{Normal equations: } (\mathbf{X}\mathbf{X}^T)\mathbf{W} = \mathbf{X}\mathbf{Y}^T \Leftrightarrow \mathbf{W} = (\mathbf{X}\mathbf{X}^T)^{-1}\mathbf{X}\mathbf{Y}^T.$$

Exploration: toy problem Run each algorithm (GD or SGD) for, say, 100 iterations $\mathbf{w}^{(0)}, \mathbf{w}^{(1)}, \dots, \mathbf{w}^{(100)}$ from an initial $\mathbf{w} = \mathbf{w}^{(0)}$ (equal to small random numbers, e.g. uniform in $[-0.01, 0.01]$). To visualize the results, we plot the following for each algorithm (GD and SGD):

- The dataset (y_n vs $\mathbf{x}_n$) and the regression line $f(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$.
- The error $E(\mathbf{w})$ over iterations, evaluated on the training set, and also on a validation set.
- A contour plot of the error $E(\mathbf{w})$ and the iterates $\mathbf{w}^0, \mathbf{w}^{(1)}, \dots$

Consider the following questions:

- Does the error $E(\mathbf{w})$ approach the optimal error $E(\mathbf{w}^*)$ (where $\mathbf{w}^*$ is the solution to the normal equations)? How fast does it approach it? Does $E(\mathbf{w})$ decrease monotonically? Or does it oscillate or even diverge?
- Vary the learning rate $\eta > 0$. What happens if it is very small or if it is very big? Try to determine the value of η that gives the fastest convergence for your toy dataset by trial and error. Note: practical values of η are usually (quite) smaller than 1.
- For SGD with fixed η, vary the minibatch size $|\mathcal{B}|$ between 1 and N. How does this affect the speed of convergence?
- For both GD and SGD we keep the learning rate η constant. How does this affect the behavior of SGD as we keep training? What should we do to improve that?
- Repeat all of the above in the following situations:
  - Use a dataset having 10 times as many points.
  - Use a dataset having 3 times larger noise.
  - Use a different initial $\mathbf{w}^{(0)}$.

  Are the best values of η the same as before for GD? How about SGD? What other changes do you observe?

See the end of file lab07_linregr.m for suggestions of things to explore.

Exploration: MNIST See file lab07_linregr_MNIST.m. Select a small enough subset of MNIST as training inputs (using the whole dataset will be slow). Apply the ground-truth linear transformation to the data to generate the output labels $\mathbf{y} = \mathbf{f}(\mathbf{x}) = \mathbf{A}\mathbf{x} + \mathbf{b}$. We can compute the optimal solution exactly by solving the normal equations, and approximately by running GD and SGD. Then, consider the same questions as with the toy dataset. Note that the appropriate values for η, etc. may now be significantly different. To visualize the results, we plot the following for each algorithm (GD and SGD):

- The training and validation error $E(\mathbf{w})$ over iterations, as with the toy dataset.
- For each of the 4 results (true mapping, optimal mapping from the normal equations, and the mappings learned by GD and SGD), we plot:

- The parameters of the linear mapping using `imagesc(A)` and `imagesc(b)`. These are signed values, so we use `colormap(parula(256))`.
- If using as linear mapping the rotation/shift/scale/clip transformation, which produces as output a (possibly smaller) image $\mathbf{y}_n$, we plot the following for a few sample images: the input image $\mathbf{x}_n$, and the outputs $\mathbf{y}_n$ under the 4 mappings.

III (Stochastic) gradient descent for logistic regression (= classification)

Review We consider the two-class case, given a sample $\{(\mathbf{x}_n, y_n)\}_{n=1}^N$ with $\mathbf{x}_n \in \mathbb{R}^D$ and $y_n \in \{0, 1\}$. Again we assume the last element of each $\mathbf{x}_n$ vector is equal to 1, so that w_D is the bias parameter in the sigmoid $\sigma(\mathbf{w}^T\mathbf{x}_n) = \frac{1}{1+\exp(-\mathbf{w}^T\mathbf{x}_n)}$. The objective functions $E(\mathbf{w})$ below do not admit a closed-form solution, so we need iterative algorithms. We apply GD and SGD, whose updates are obtained by computing the gradient $\nabla E(\mathbf{w})$ of E wrt $\mathbf{w}$. We can learn $\mathbf{w}$ by maximum likelihood, or by regression.

- By *maximum likelihood*: we minimize the cross-entropy

$$E(\mathbf{w}) = \sum_{n=1}^N e(\mathbf{w}; \mathbf{x}_n, y_n) \qquad \text{where} \qquad \begin{cases} e(\mathbf{w}; \mathbf{x}_n, y_n) = -y_n \log\theta_n - (1-y_n)\log(1-\theta_n) \\ \theta_n = \sigma(\mathbf{w}^T\mathbf{x}_n) = \dfrac{1}{1+\exp(-\mathbf{w}^T\mathbf{x}_n)}. \end{cases}$$

With GD and SGD we have the following updates:

$$\text{GD: } \boldsymbol{\Delta}\mathbf{w} = \eta \sum_{n=1}^N (y_n - \theta_n)\mathbf{x}_n \qquad \text{SGD: } \boldsymbol{\Delta}\mathbf{w} = \eta \sum_{n\in\mathcal{B}} (y_n - \theta_n)\mathbf{x}_n.$$

- By *regression*: we minimize the least-squares error

$$E(\mathbf{w}) = \sum_{n=1}^N e(\mathbf{w}; \mathbf{x}_n, y_n) \qquad \text{where} \qquad \begin{cases} e(\mathbf{w}; \mathbf{x}_n, y_n) = \frac{1}{2}(y_n - \theta_n)^2 \\ \theta_n = \sigma(\mathbf{w}^T\mathbf{x}_n) = \dfrac{1}{1+\exp(-\mathbf{w}^T\mathbf{x}_n)}. \end{cases}$$

With GD and SGD we have the following updates:

$$\text{GD: } \boldsymbol{\Delta}\mathbf{w} = \eta \sum_{n=1}^N (y_n - \theta_n)\theta_n(1-\theta_n)\mathbf{x}_n \qquad \text{SGD: } \boldsymbol{\Delta}\mathbf{w} = \eta \sum_{n\in\mathcal{B}} (y_n - \theta_n)\theta_n(1-\theta_n)\mathbf{x}_n.$$

Exploration: toy problem Proceed as in the linear regression case. The differences now are: 1) the problem is classification rather than regression; 2) the objective function $E(\mathbf{w})$ and hence its gradient and GD/SGD updates are different; and 3) the model is $\sigma(\mathbf{w}^T\mathbf{x})$ rather than $\mathbf{w}^T\mathbf{x}$. Some additional questions to consider:

- How do the contours of the objective function (cross-entropy or least-squares error) look like compared with each other, and compared with the contours of the least-squares error for the linear regression case?
- Try datasets where the two classes are linearly separable, and where they are not linearly separable. Do the contours of the objective function look the same in both cases? Why? How does this affect the optimal parameters? More specifically: in the linearly separable case, what happens to $\|\mathbf{w}\|$ as the number of iterations increases? Will the algorithm ever converge?
- Try using as initial weights $\mathbf{w} \in \mathbb{R}^D$ random numbers uniformly distributed in $[-u, u]$. Try a small value of u (say, 0.01) and a large one (say, 100). What happens, and why? Which initialization is better?

See the end of file `lab07_logregr.m` for suggestions of things to explore.

Exploration: MNIST As in the linear regression case. See file `lab07_logregr_MNIST.m`.