**–l`Assignment 0 [evolving]**

**Contents:**

* [**Evolving Assignments**](https://docs.google.com/document/d/1RUm7GHJhFJlT9Sd5KPK9iXUEAOGPiEGR9zPswocGIpA/edit#bookmark=id.8cuzoa2su3ik)
* [**Rules for Assignment 0**](https://docs.google.com/document/d/1RUm7GHJhFJlT9Sd5KPK9iXUEAOGPiEGR9zPswocGIpA/edit#bookmark=id.3b4n8euupldq)
* [**Step 1. A simple noisy linear dataset.**](https://docs.google.com/document/d/1RUm7GHJhFJlT9Sd5KPK9iXUEAOGPiEGR9zPswocGIpA/edit#bookmark=id.uc4o1575nxz1) **(Soft due date: Friday)**
* [**Step 2. Evaluating a Prediction of the Linear Function**](https://docs.google.com/document/d/1RUm7GHJhFJlT9Sd5KPK9iXUEAOGPiEGR9zPswocGIpA/edit#bookmark=id.w4rnakvyswz9) **(Soft due date: Friday)**
* [**Step 3. The Loss Surface**](https://docs.google.com/document/d/1RUm7GHJhFJlT9Sd5KPK9iXUEAOGPiEGR9zPswocGIpA/edit#bookmark=id.91v3vsp6xxmr)
* [**Step 4. Solving the Optimization Directly.**](https://docs.google.com/document/d/1RUm7GHJhFJlT9Sd5KPK9iXUEAOGPiEGR9zPswocGIpA/edit#bookmark=id.vcdmv2nsw5m3)
* [**Step 5.** **Computing the Gradient of the Loss Function.**](https://docs.google.com/document/d/1RUm7GHJhFJlT9Sd5KPK9iXUEAOGPiEGR9zPswocGIpA/edit#bookmark=kix.qknvhsf1312l)
* [**parts of the assignment coming up:**](https://docs.google.com/document/d/1RUm7GHJhFJlT9Sd5KPK9iXUEAOGPiEGR9zPswocGIpA/edit#bookmark=id.tkv71pg26ajo)
* [**Submission Instructions**](https://docs.google.com/document/d/1RUm7GHJhFJlT9Sd5KPK9iXUEAOGPiEGR9zPswocGIpA/edit#bookmark=id.kqrpvmnaf9oq)
* [**Resources**](https://docs.google.com/document/d/1RUm7GHJhFJlT9Sd5KPK9iXUEAOGPiEGR9zPswocGIpA/edit#bookmark=id.krjyckxi3vhd)

***Evolving assignments…***

…are one of my responses to online teaching; they will be (or have been) discussed in class.

A key feature is that you are encouraged to ask your questions directly as comments on the assignment itself, and the assignment may be clarified accordingly. Another key feature is that we may share future (draft) parts of the assignment before they are complete (and will indicate them as such) to give you the opportunity to see “where it’s going” and possibly help shape it.

***Rules for Assignment 0.***

* You may discuss this assignment with your classmates but you must write your own code and your own answers to questions.
* You must write your code in python, and, importantly:
  + you can use the matplotlib library
  + ***you can only some functions in the*** numpy library
    - you **may not** use the pre-written functions for regression such as the numpy.polynomial package
  + if you are unsure whether a certain package is allowed, ask here on this document and we will answer. We will then keep help keep track of both your questions and our corresponding answers. Using a library/function that is not allowed can result in significant mark penalties. The intention here is of course not to penalize, it is to make sure you implement the algorithms that we feel are important for you to learn and understand, rather than using a function that has already implemented them. If you’re unsure if a function makes the question “too easy”, just ask us!
* The early partial submission policy will be discussed in class and updated as needed.

**Step 1. A simple noisy linear dataset.**

You will eventually be learning to make predictions based on data. The first step is to understand a simple dataset. We will start with a noisy linear dataset, and to understand it, you will need to be able to generate it. The dataset will consist of a set of N pairs of points: {(x1,y1),(x2,y2), … (xN,yN)} . For now, we will assume that both xi and yi are each scalars, i.e. 1-dimensional points. We will also assume that they have a linear relationship, and that there is measurement noise:

ytrue = wtrue x + btrue + noise .

w and b have the subscript true to distinguish them from your estimates in later steps. Noise is a value added to each measurement, sampled stochastically from a probability distribution as described in step (b) below.

Implementation notes:

* N will be a user-specified parameter: do not hard-code it into your code.
* The data set should be represented using numpy arrays, not python lists (a bit more info on this is given below).

In this warmup you will generate and visualize random datasets that have this structure. To do this, you will use the numpy.random and matplotlib libraries to:

* (a) write a function   
          chooseParams(..)   
  that chooses values for wtrue and for btrue from a uniform distribution over [-2,2].
* (b) write a function  
          generateNoisyLinearData( n, xLo, xHi, w, b,  
                              noiseType, sigma )   
  that generates a set of n xi points and generates their corresponding label values, yi, given w and b, with noise of type noiseType and possibly with standard deviation sigma. For now, noiseType can be either uniform or gaussian. (If the noise is uniform, then sigma has no impact, and you can assume that it is a uniform distribution over the interval [-1,+1]. If the noise is gaussian, then you can assume zero-mean.) The n xi points should have x values sampled uniformly from the interval [xLo, xHi]. generateNoisyLinearData() should return two arrays, X and Y, containing the generated xi’s (in X) and yi’s (in Y).
* (c) write a function  
          plotLinearData(...)   
  that takes a set of points and plots them in a scatter plot. Plot the points that you generated with the function above. Make sure that the axes are labelled sufficiently that the viewer can see that the points are indeed in the correct ranges.

**Step 2. Evaluating a Prediction of the Linear Function**

The basic idea of supervised learning is:

(1) you are given some data, e.g. x1, x2, x3, …, xN, where each xi is called an *input*, and

(2) you are given corresponding labels (also called outputs), e.g.

output label for x1 is y1,

output label for x2 is y2,

etc,

and

(3) your task is to predict the outputs yi for other inputs xN+1, xN+2,... that you have not seen before.

For now, as described earlier, we will assume that y is a linear function of x, i.e.

y=wx+b,

but with noise added. However, we assume that we do not know what the *true* values of w or b are. We only see the final values of y for the x’s that we were given. If there were no noise, then we could just solve the linear equation. But because noise is added, there will not be an exact solution. So if we estimate a value of w and b, then that might lead to more accurate predictions for some of the y’s, and less accurate predictions for others. Note that in this step, our “estimate” for w and b is just a random guess! Later we will see how to improve on this.

You need to quantify how good a given estimate is for w and b:

* write a function  
    
          loss(w,b,x,ytrue,lossType)  
            # lossType can be an int such that  
            #   lossType == 0 indicates absolute loss, and  
            #   lossType == 1 indicates squared error loss  
            #   (if you’ve done it as a string that’s OK too)  
     
  computing loss for a single data point
  + if lossType is absolute, then the loss is given by the distance from the estimate of y (using w,b,x) to the true value of y
  + if lossType is square, then the loss is given by the square of the distance, i.e. *(ypred - ytrue)2* .
* write function  
          cost(..)  
  which, given estimates for w and b, computes the average loss over all points in a given data set D.
* write a function to plot the cost for a given dataset and a given value of b, as w varies from -2 to 2.
* Try generating various noisy linear datasets. For a given dataset, what determines the value of the lowest cost that you can expect to obtain?

**Step 3. The Loss Surface. (soft due early next week)**

In this step you will plot the surface of the cost as a function of w and b (that is, a function of your *estimates* for w and b, where the true values are unknown and kept fixed).

1. Start by producing a 3D plot of the cost as a function of w and b.

1. Sketch an approximate contour plot based on the above plot (don’t worry about being exact). Check your sketch by producing a contour plot using matplotlib.

1. Based on the above, can you hazard a guess as to the values of w and b that minimize the cost function?

Hint: The mplot3d toolkit (<https://matplotlib.org/stable/tutorials/toolkits/mplot3d.htm>l),np.meshgrid (<https://numpy.org/doc/stable/reference/generated/numpy.meshgrid.html>**)**, and plt.contourf (<https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.contourf.html>) might prove handy.

**Step 4. Solving the Optimization Directly. (will be soft due early next week)**

For linear regression a closed-form expression for the optimal regression parameters is available, and in this step you will attempt to deduce it. This question is asking you to do a mathematical derivation (it is not a programming question).

1. To start with, suppose that that wtrue = 0. That is, assume that y does not actually depend on x. The optimal b is the value that minimizes the cost function, b = argminb Cost(w=0,b). Solve for this optimal b analytically in terms of xi’s and yi’s. As a sanity check, plug in the formula for yi assuming there was no noise (yi=btrue). What does your original expression reduce to?

1. Now do the converse: assume that btrue = 0, and solve for w analytically, also in terms of xi’s and yi’s. As a sanity check, plug in the formula for yi assuming there was no noise (yi=wtruexi). What does your original expression reduce to?

Keep in mind that if wtrue = 0, you can take your model to be ypred= b, and similarly, for the case in which btrue= 0, you can take your model to be ypred = w x.

1. Solve for w and b simultaneously, that is, do not assume that either wtrue or btrue are zero and solve for w, b = argminw,b Cost(w,b), in terms of xi’s and yi’s.

**Step 5.** **Computing the Gradient of the Loss Function. (soft due early next week)**

To use gradient descent to solve the linear regression optimization problem, expressions for the partial derivatives of the cost function are required. In this step you will obtain these expressions and write their corresponding python functions.

1. Derive analytical expressions for the partial derivatives of the cost function with respect to both w and b.
2. Write python functions computeDeriv\_dC\_dw(w,b,X,Y), and computeDeriv\_dC\_db(b,w,X,Y) that return the partial derivatives of the cost function with respect to w and b respectively (given inputs X and labels Y).
3. Numerically check the analytical expressions you found for the optimal w and b, wopt and bopt, in the previous step: if they are correct, and you plug in wopt and bopt as your values for w and b, then what should  incomputeDeriv\_dC\_dw(wopt ,bopt,X,Y) and computeDeriv\_dC\_db(bwopt,wopt,X,Y) return? Can you perturb w and b to check that the sign of your numerical derivatives is correct, at least near the optimum?

**Evolving parts of the assignment coming up:**

[draft notes:… add section to require

incorporating vectorization in the code.]

**Step 6.** **Implement Gradient Descent to Solve the Noisy Linear System.   
      (will be soft due Wednesday 19 Jan)**

The goal of your system is to find the model parameters that optimize the loss function.

You already solved this problem analytically, but now you will do this using a method called gradient descent.

The basic idea of gradient descent is that for our current estimate of the model parameters, we look at the loss function, and “move downwards”. That is, we change each parameter just a little bit so that the loss function decreases. We do this by computing the gradient of the loss function at the current parameters, i.e.

                    [ dL/d*w*, dL/d*b* ].

We then “take a small step” in the direction opposite to the gradient. That is, we update each parameter by subtracting a value that is proportional to the partial derivative of the cost with respect to that parameter.

If your parameters at step t are wt and bt, then the update rules will be:

    wt+1= … [fill this in yourself!], and

    bt+1= … [fill this in yourself!].

You have already implemented one of the most important steps for gradient descent: computing the derivatives. All you need to do now is write an   
                  
                updateParams(w,b,X,Y,alpha,epochs)   
  
function that computes the derivatives (by calling your functions from the previous step) and then using this, along with a user-specified learning rate, alpha, to update your estimates of the parameters (in this  case *w* and *b*) according to the update equations you just filled in above. The parameters should get updated using the update rule epochs number of times.

After you write the updateParams function, use it to find the optimal values for w and b. Compare the values you found with those obtained using your analytical expression.

Plot the optimal values for the parameters that you found through gradient descent on the contour plot from step 3. Are they where you expected them to be?

Note that at each iteration step *t*, you will have an updated set of parameters *(wt , bt )* and a corresponding cost, i.e. error Et. Make two plots:

1. a plot of Et as the iterations progress.
2. a plot of the parameters as the training progresses (i.e. in parameter space). Overlay this plot over the contour plot from step 3b).

Try different learning rates. What can you say about the qualitative behavior of gradient descent as the learning rate is changed?

**Step 7**. **Add Regularization**

In this step you will implement regularized linear regression. For our simple dataset and model it is not that useful, but it is something that is used frequently in more complicated models/problems.

Suppose that instead of simply minimizing the squared loss, you also penalized the magnitude of your parameters, *w* and b, that is,

Lreg= L+(½)λ(w2+b2), where λ is a hyperparameter.

What are the new update rules for the parameters? Write a new python function updateParams that updates the parameters with these update rules.

**Step 8. Non-linear data and Non-linear model!**

In this step you will import a non-linear dataset with some noise and perform gradient descent to: 1) try to fit our simple linear regression model to the dataset; 2) fit a more complicated non-linear model. **All the new code you write should be vectorized.** No regularization is needed.

For the more open-ended questions please provide clear and well-supported answers in order to get full marks. (Note that a very unclear answer may get zero marks even if it uses the “right keywords”.)

1. Go to the Files tab under Assignment 0 in Teams and download ‘dataset\_cos.npy’. Import the dataset using   
   x, y =  np.load("dataset\_cos.npy"). Make a plot of the dataset.   
   You can assume from now on that ytrue= cos(2x+1)+noise.
2. Try to perform linear regression on this dataset using gradient descent. Analytically calculate the cost associated with a model that predicts y = 0 for all x, assuming you have infinitely many sample points (you can disregard noise). In your analytically calculation, you can take x to be uniformly distributed from 0 to 2𝝿. How close is this to the loss you get on gradient descent?
3. Now take your model to predict ypred= cos(w\*x+b). What are the parameter update rules for gradient descent for this model?   
   Suppose you didn’t know that wtrue=2(and btrue=1), based on the dataset provided can you think of a way to estimate wtrue? Describe it clearly.
4. **\*** Perform gradient descent using the model and update rules from step c). Try the following four different initializations:   
       (winitial=0.1,binitial=0.0),   
       (winitial=0.5,binitial=0.0),   
       (winitial=1.5,binitial=0.0),   
         (winitial=1.9,binitial=0.0).   
   You can take the learning rate to be 0.05 in all cases.   
   Run gradient descent for 1000 epochs at most. Show your rgesults.  
   Discuss the performance, e.g. how close did you get to a zero loss for all initializations, and why? Provide some evidence for your answer.
5. Implement gradient descent with momentum. In gradient descent with momentum we use new update rules:  
       mt+1,w = μ mt,w+ dL/dw; wt+1 = wt-⍺ mt+1,w,  
   and similarly for other parameters. μ is the momentum factor (typically taken to be 0.9 or 0.99; take it to be 0.99).  
   Try running gradient descent with momentum and the initializations from question d), using a learning rate of 0.05.   
   Do you converge for more initializations than you did without momentum?    If so, why do you think this is? Does your answer fit with the explanation you gave in question d)?
6. So far we’ve been calculating the overall cost for the whole dataset and updating our parameters based on the derivatives of the cost, but what if instead of updating the parameters based on the full cost we updated the parameters for each sample point and based on the loss associated with that sample point, i.e.,   wt+1 = wt-⍺ dl/dw, where l is the loss for a single data point. Implement this in python and try doing gradient descent using this approach. Does this method improve convergence for more of the initializations (you can take the learning rate to be 0.05)?
7. Implement the method from f) but with momentum (you can take it to be 0.99). What happens if you take the same learning rate as in f)? Take the learning rate to be 0.0005. Do you manage to converge to better minimums than in f), i.e., values for the parameters closer to the true values?
8. **6505 only:** If you found that convergence improved in e), f), and/or g) can you offer an explanation as to why this probably happened in each case?

***Submission Instructions***

Please submit a single ipynb file through Brightspace.

Steps 1 and 2 are due at 11:59PM on the 8th of January, but if you submit them by 8:45AM Friday the 7th, you will get bonus points on your assignment.

Steps 3, 4, and 5 are due at 11:59PM on the 13th of January, but if you submit them by 8:45AM Wednesday the 12th, you will get bonus points on your assignment.

Steps 6 and 7 are due at 11:59PM on the 20th of January, but if you submit them by 8:45AM Wednesday the 19th, you will get bonus points on your assignment.

Step 8 is due at 11:59PM on the 3rd of February, but if you submit it by 8:45AM Wednesday the 2nd, you will get bonus points on your assignment.

***Resources.***

Please add any resources to this list that you have found useful!

* Matplotlib
  + [a quick intro](https://jakevdp.github.io/PythonDataScienceHandbook/04.00-introduction-to-matplotlib.html) to matplotlib: this web page takes ~10min to read and is helpful for making sense of other matplotlib documentation. Once you’ve read it, then at the bottom you can click to continue with more info and examples as needed.
* numpy
  + [a visual intro to numpy](https://jalammar.github.io/visual-numpy/) by Jay Alammar
  + numpy [quickstart](https://numpy.org/doc/stable/user/quickstart.html) on the numpy homepage
  + [Deep Learning by deeplearning.ai | Coursera](https://www.coursera.org/specializations/deep-learning)