D200, Problem Set 1: Gradient Descent

Due: 4 February 2025 here in groups of 4.

Stefan Bucher

This problem set will introduce you to the gradient descent algorithm to solve the linear regression problem discussed in the lecture.[1] We will consider a univariate linear regression

$$y = \theta_0 + \theta_1 X + \varepsilon$$

and seek to fit the parameters θ_0 and θ_1.

Problem 1

(1a) Write down the mean-squared error function and its gradient with respect to the parameters.

(1b) Implement the loss function `loss(X,y,theta)` and gradient function `gradient(X,y,theta)` in Python.

Problem 2

(2a) Implement the simple gradient descent algorithm

$$\theta_j \leftarrow \theta_j - \alpha \frac{\partial L}{\partial \theta_j}$$

as a Python function `gradient_descent(X, y, theta_init, alpha, maxsteps, precision)`. The function should take the data X, y, initial parameters θ, step size, maximum number of steps, and a precision tolerance parameter. The function should return the history of the parameters, the cost at each step, and the predictions at each step.

[1] The problem set is adapted from Chi Jin's OxML 2024 Fundamentals class on Optimization.

Then, use this function to print the resulting estimate, and plot the evolution of the parameters and the mean-squared loss over 2000 iterations, with initial parameters $\theta_0 = \theta_1 = 0$ and learning rate $\alpha = 0.01$.

(2b) Compare the resulting estimates with those generated by `scipy.stats`, `statsmodels`, or `sklearn`. Plot the best-fitting line along with a scatterplot of the data.

Problem 3

Visualize the loss surface, using the provided helper functions.

```
from matplotlib import cm
import warnings
warnings.filterwarnings("ignore")


def plotting(history, cost):
    theta1s = np.linspace(slope - 40 , slope + 30, 50)
    theta0s = np.linspace(intercept - 40 , intercept + 30, 50)
    M, B = np.meshgrid(theta1s, theta0s)
    zs = np.array([loss(X, y, theta)
                   for theta in zip(np.ravel(B), np.ravel(M))])
    Z = zs.reshape(M.shape)
    fig = plt.figure(figsize=(10, 10))
    ax = fig.add_subplot(111, projection='3d')
    ax.plot_surface(M, B, Z, cmap=cm.coolwarm, rstride=1, cstride=1, linewidth=0, color='whi
    ax.contour(M, B, Z, 20, color='b', alpha=0.5, offset=0, stride=0)
    ax.view_init(elev=15., azim=73)
    ax.plot([history[-1][1]], [history[-1][0]], [cost[-1]] , markerfacecolor='r', markeredge
    ax.plot([history[0][1]], [history[0][0]], [cost[0]] , markerfacecolor='r', markeredgecol

    ax.plot([t[1] for t in history], [t[0] for t in history], cost , markerfacecolor='r', ma
    ax.plot([t[1] for t in history], [t[0] for t in history], 0 , markerfacecolor='r', marke

    ax.set_xlabel(r'$\theta_1$', fontsize=24)
    ax.set_ylabel(r'$\theta_0$', fontsize=24)
    ax.set_title(f"Iteration: {len(history)}", fontsize=24, fontweight='bold')


def plotting_diverge(history, cost):
    #print("X: ", theta[1]," abs(theta[1]) :", abs(theta[1]))
    theta1s = np.linspace(  -(-0.34 + abs(theta[1])*0.6 + 50) , -0.34 + abs(theta[1])*0.6 +5
```

```
    theta0s = np.linspace(79.18 - abs(theta[1]) -200, 79.18 + abs(theta[1]) +200, 50)
    M, B = np.meshgrid(theta1s, theta0s)
    zs = np.array([loss(X, y, theta)
                   for theta in zip(np.ravel(B), np.ravel(M))])
    Z = zs.reshape(M.shape)
    fig = plt.figure(figsize=(10, 10))
    ax = fig.add_subplot(111, projection='3d')
    ax.plot_surface(M, B, Z, cmap=cm.coolwarm, rstride=1, cstride=1, color='b', alpha=0.8)
    ax.contour(M, B, Z, 20, color='b', alpha=0.5, offset=0, stride=0)
    ax.view_init(elev=15., azim=73)
    ax.plot([history[0][1]], [history[0][0]], [cost[0]], markerfacecolor='r', markeredgecolo
    ax.plot([history[0][1]], [history[0][0]], [cost[0]] , markerfacecolor='r', markeredgecolo

    ax.plot([t[1] for t in history], [t[0] for t in history], cost , markerfacecolor='r', mar
    ax.plot([t[1] for t in history], [t[0] for t in history], 0 , markerfacecolor='r', marker

    ax.set_xlabel(r'$\theta_1$', fontsize=24)
    ax.set_ylabel(r'$\theta_0$', fontsize=24)
    ax.set_title(f"Iteration: {len(history)}", fontsize=24, fontweight='bold')


def gen_gif(method_name,h,c,div=False):
  %matplotlib notebook
  from tqdm import tqdm
  #import os
  # generate gif, run after the specific method run
  theta =  h[-1]
  method_name = method_name

  # use tqdm to create a progress bar
  print("Produce the pngs")
  for i in range(1,len(h)+1):
    history_, cost_ = h[0:i], c[0:i]
    if not div:
      plotting(history_,cost_)
    else:
      plotting_diverge(history_,cost_)
    plt.savefig(f'figures/{method_name}-{i}.png')

  import imageio
  import os
  print("Produce the GIF")
```

```
  with imageio.get_writer(f'figures/{method_name}.gif', mode='I') as writer:
      for filename in tqdm([f'figures/{method_name}-{i}.png' for i in range(1,len(h)+1)], des
          image = imageio.imread(filename)
          writer.append_data(image)
          os.remove(filename)


def plotting_loss_surface():
    theta1s = np.linspace(slope - 40 , slope + 30, 50)
    theta0s = np.linspace(intercept - 40 , intercept + 30, 50)
    M, B = np.meshgrid(theta1s, theta0s)
    zs = np.array([loss(X, y, theta)
                   for theta in zip(np.ravel(B), np.ravel(M))])
    Z = zs.reshape(M.shape)
    fig = plt.figure(figsize=(10, 10))
    ax = fig.add_subplot(111, projection='3d')
    ax.plot_surface(M, B, Z, cmap=cm.coolwarm, rstride=1, cstride=1, linewidth=0, color='b',
    ax.contour(M, B, Z, 20, color='b', alpha=0.5, offset=0, stride=0)
    ax.view_init(elev=15., azim=73)

    ax.set_xlabel(r'$\theta_1$',fontsize=24)
    ax.set_ylabel(r'$\theta_0$',fontsize=24)
```

Use function `plotting(history, costs)` to visualize the loss surface for various values of the hyperparameter `alpha` to get an intuition for how the learning rate affects the performance of the algorithm. Discuss your observations.