sgd_mnist

March 6, 2020

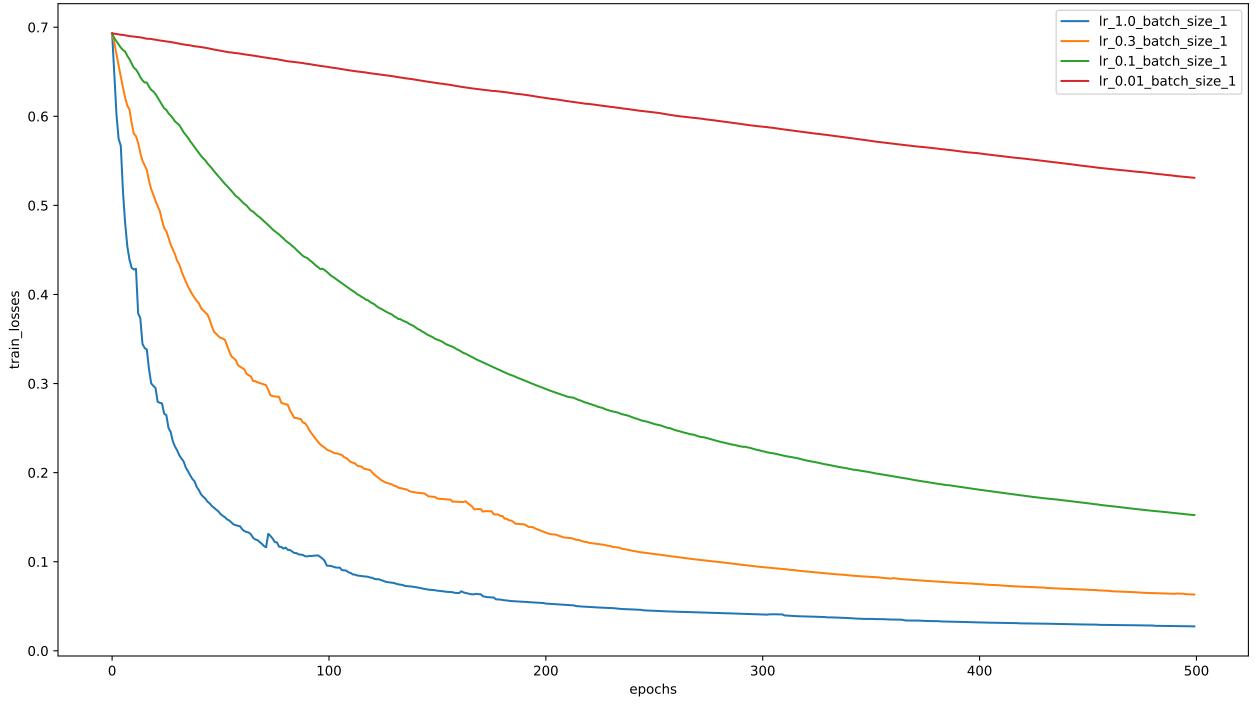
```
[31]: import numpy as np
     import matplotlib as mpl
     import matplotlib.pyplot as plt
     import os
     import argparse
     import pdb
     import time
[32]: def parse_args():
         parser = argparse.ArgumentParser()
         parser.add_argument('--lr', dest='lr', type=float, default=1e-3,__
       →help="learning rate")
         parser.add argument('--batch size', dest='momentum', type=float, default=0.
       return parser.parse_args()
[33]: def load_dataset(data_path="./"):
          image_size = 28 # width and length of mnist image
         num_labels = 10 # i.e. 0, 1, 2, 3, ..., 9
         image_pixels = image_size * image_size
         train_data = np.loadtxt(os.path.join(data_path, "mnist_train.csv"),__
       →delimiter=",")
         test_data = np.loadtxt(os.path.join(data_path, "mnist_train.csv"), ___
      →delimiter=",")
         return {"train_data":train_data,
                 "test_data": test_data,
[12]: def process_data(raw_data, labels_req=[0,1]):
         train_data = raw_data["train_data"]
         test_data = raw_data["test_data"]
```

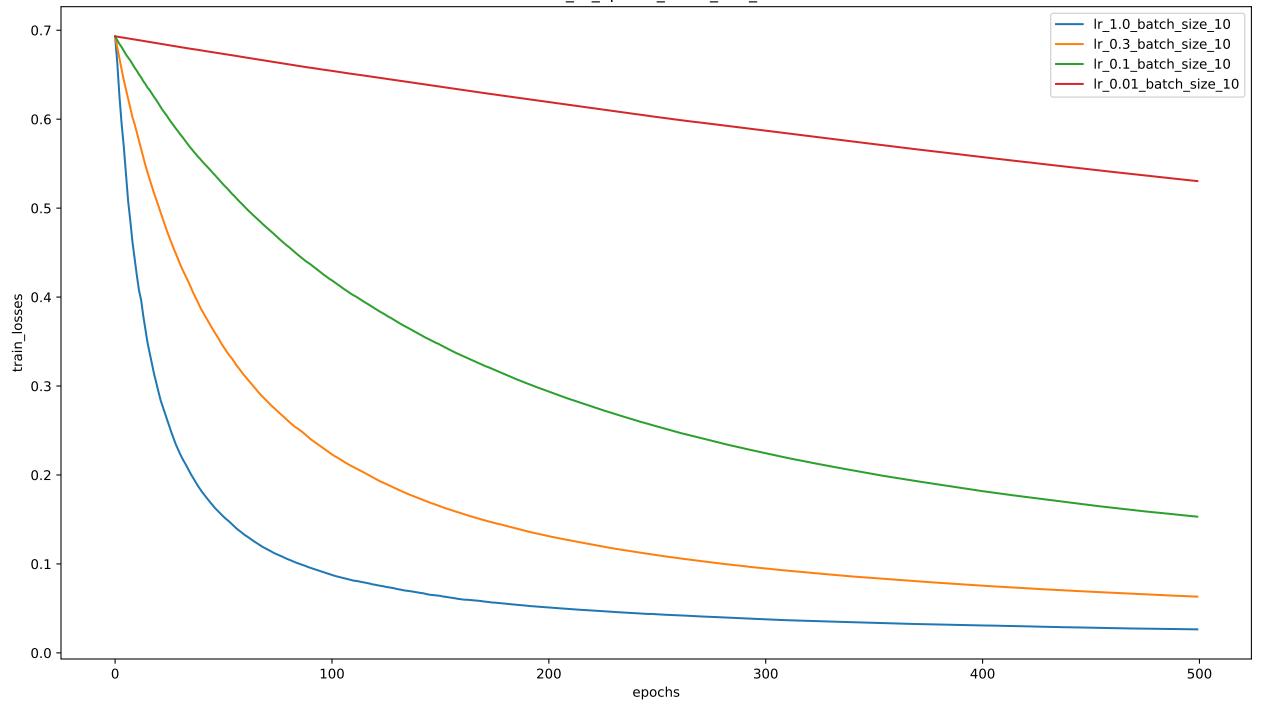
```
# rescale image from 0-255 to 0-1
          fac = 1.0 / 255
          train_imgs = np.asfarray(train_data[:, 1:])
          test_imgs = np.asfarray(test_data[:, 1:])
          train_labels = np.asfarray(train_data[:, :1])
          test_labels = np.asfarray(test_data[:, :1])
          train_imgs = np.divide(train_imgs, np.linalg.norm(train_imgs, axis=1,_
       →keepdims=True))
          test_imgs = np.divide(test_imgs, np.linalg.norm(test_imgs, axis=1,_
       →keepdims=True))
          train_mask = np.isin(train_labels[:,0],labels_req)
          test_mask = np.isin(test_labels[:,0],labels_req)
          dataset = { "X_train": train_imgs[train_mask],
                      "Y_train": train_labels[train_mask] *2.0 - 1.0,
                      "X_test": test_imgs[test_mask],
                      "Y_test": test_labels[test_mask] *2.0 - 1.0,
                  }
          return dataset
[34]: raw_data = load_dataset(data_path="./")
[35]: dataset = process_data(raw_data.copy(),labels_req=[0,1])
[36]: def plot_props(data_arr,prop_names, figname, xlabel, x_data=None):
          fig = plt.figure(figsize=(16,9))
          for i in range(len(data_arr)):
              if(x_data):
                    print(x_data.shape, data_arr.shape)
                  plt.plot(x_data[i], data_arr[i], label=prop_names[i])
              else:
                  plt.plot(data_arr[i], label=prop_names[i])
              plt.ylabel("train_losses")
              plt.xlabel(xlabel)
              plt.legend()
          plt.title(figname)
          plt.savefig("./{}.pdf".format(figname))
            plt.show()
[37]: def get_loss_grad(W, X, y_true, require_grad=True):
          W: weight vector (n,)
```

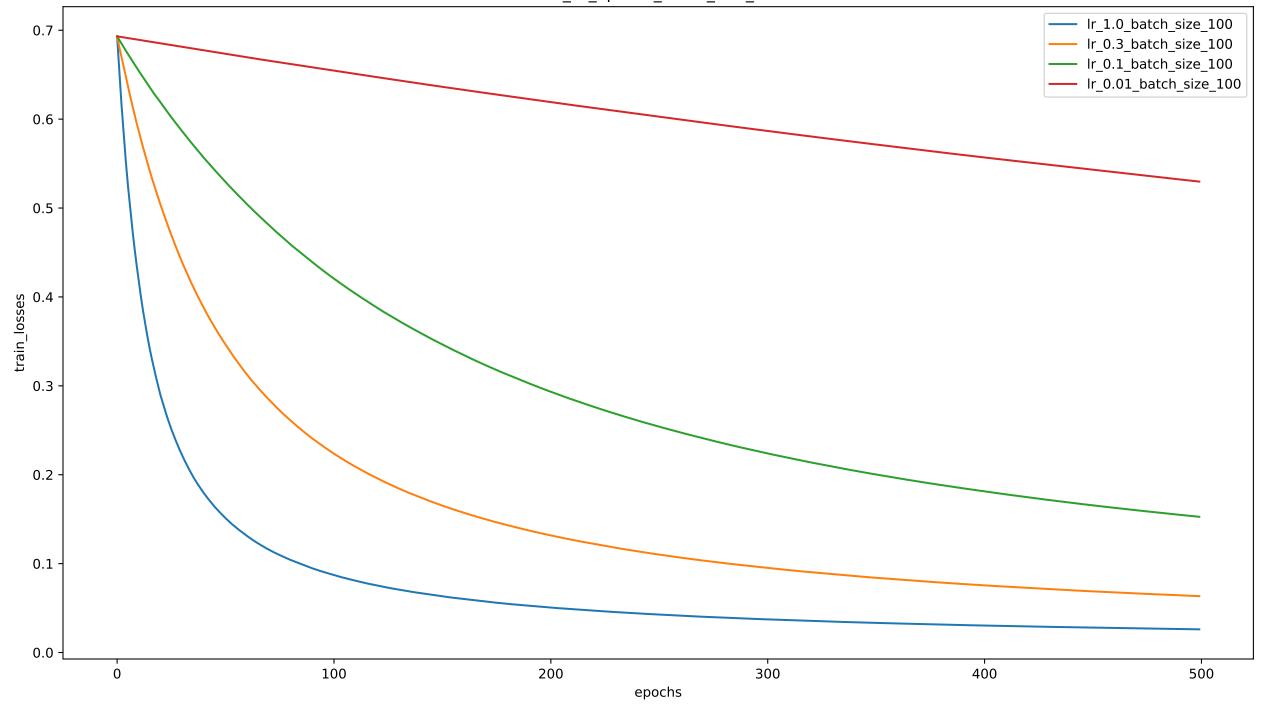
```
X: input batch (batch_size, n)
          111
          dot_prod = np.matmul(X,W)
          expo = np.exp(-np.multiply(y_true, dot_prod))
          loss = np.mean(np.log(1 + expo))
          if require_grad:
              grad = np.divide(expo , (1+expo))
              grad = np.multiply(grad, -1.0*np.multiply(y_true,X))
              grad = np.mean(grad, 0)
              return loss, grad
          return loss
[38]: def test_train_data(W, train_data, train_labels):
          train_loss = get_loss_grad(W, train_data, train_labels, require_grad=False)
          return train_loss
[39]: def main(lr,batch size):
          data_path = "./"
          num_iters = 500
          X_train, Y_train = dataset["X_train"], dataset["Y_train"]
          X_test, Y_test = dataset["X_test"], dataset["Y_test"]
          in_dim = X_train.shape[1]
          W = np.zeros(shape=(in_dim,1))
          train_data_loss_arr = []
          train_time_arr = []
          start time = time.time()
          eval_time = 0.
          loss calc time = 0.
          for i in range(num_iters):
              train data loss = 0.0
              idxs = np.random.choice(X_train.shape[0], batch_size, replace=True)
              X = X_train[idxs]
              y_true = Y_train[idxs]
              loss, grad = get_loss_grad(W, X, y_true)
              train_time_arr.append(time.time() - start_time - loss_calc_time)
              temp = time.time()
              train_data_loss_arr.append(test_train_data(W, X_train, Y_train)*1.0)
              loss_calc_time += time.time() - temp
              W[:,0] -= lr*grad
          return np.array(train_data_loss_arr), np.array(train_time_arr)
```

```
# plot\_props(train\_data\_loss\_arr, "train\_data\_loss\_lr\_{}\_batch\_size\_{}". \\ \hookrightarrow format(lr, batch\_size))
```

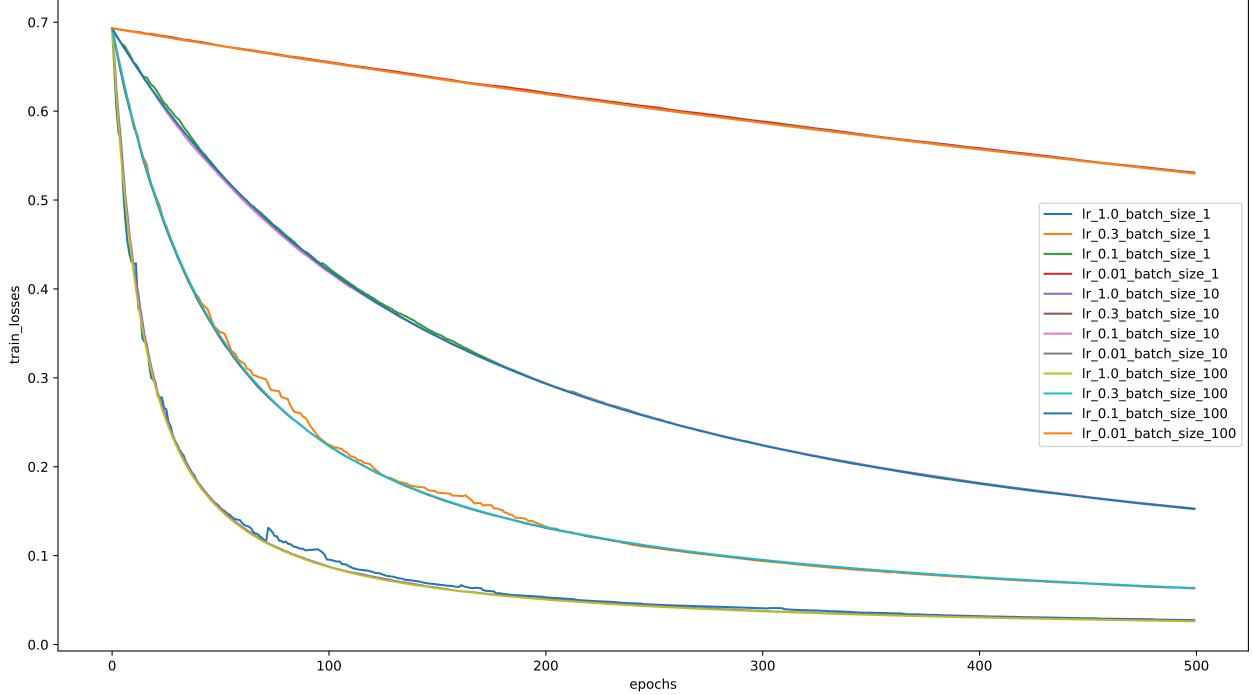
```
[]: lrs = [1.0, 0.3, 0.1, 0.01]
     batch_sizes = [1,10,100]
     \# lrs = [1.0]
     # batch_sizes = [10]
     TRAIN LOSSES = []
     TRAIN_TIMES = []
     PROP_NAMES = []
     for batch_size in batch_sizes:
         print("batch_size", batch_size)
         prop_names = []
         train_losses = []
         train times = []
         for lr in lrs:
             print("learning_rate", lr)
             start_time = time.time()
             train_loss, train_time = main(lr, batch_size)
             train_losses.append(train_loss*1.0)
             train_times.append(train_time*1.0)
               print("Time_taken: {:1f}".format(time.time() - start_time))
             prop_names.append("lr_{} batch_size_{}".format(lr, batch_size))
         plot_props(train_losses, prop_names, "loss_vs_epochs_batch_size_{}".
      →format(batch_size), "epochs")
         plot_props(train_losses, prop_names, "loss_vs_time_batch_size_{}".
      →format(batch_size), "train_time", train_times)
         TRAIN_LOSSES.extend(train_losses)
         TRAIN_TIMES.extend(train_times)
         PROP_NAMES.extend(prop_names)
     plot_props(TRAIN_LOSSES, PROP_NAMES, "loss_vs_epochs", "epochs")
     plot_props(TRAIN_LOSSES, PROP_NAMES, "loss_vs_time", "train_time", TRAIN_TIMES)
```

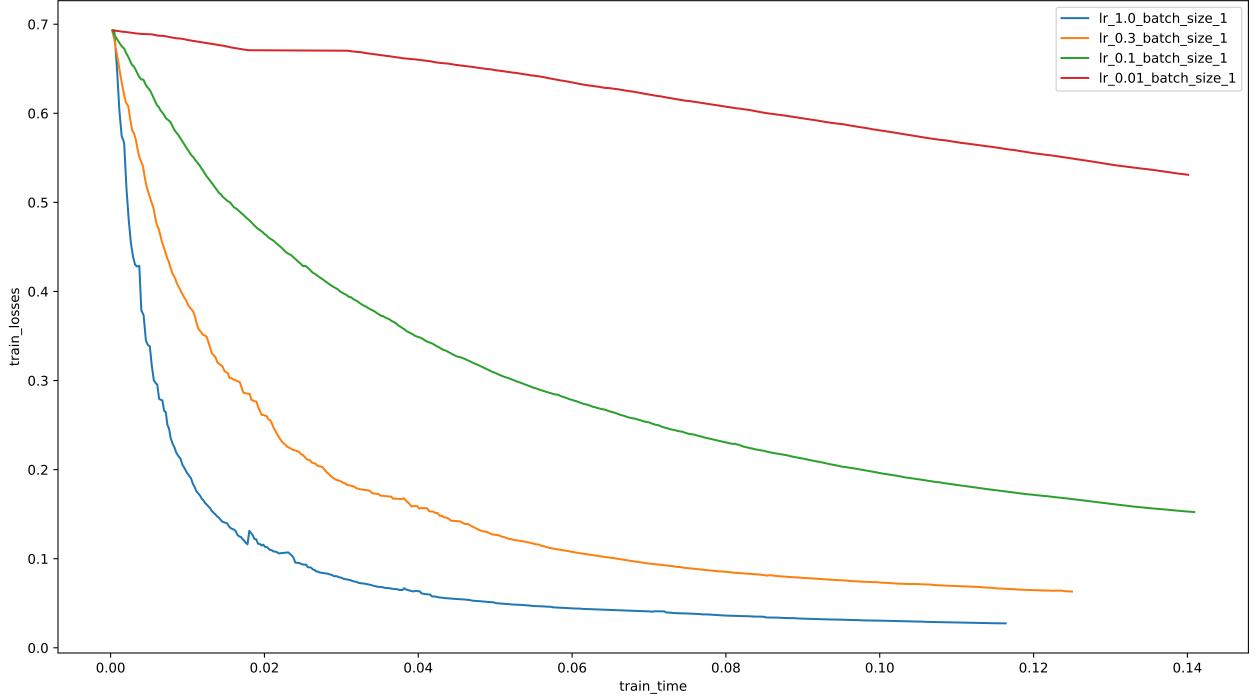












loss_vs_time_batch_size_10

