a02_5

October 24, 2025

1 5 Exploration (optional)

Report of Jonas Ortner: 2265527 Marmee Pandya: 1963521

[1]:
```
# Safe run block for the PyTorch optimize() call
import math
import numpy as np
import matplotlib.pyplot as plt
import sklearn

import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.utils.data

%load_ext autoreload
%autoreload 2

from a02_helper import *
from a02_functions import optimize

torch.manual_seed(0)
np.random.seed(0)
torch.set_num_threads(1)  # reduce thread-related instability
```

[2]:
```
# Prepare tensors (safe explicit conversions)
Xztorch = torch.from_numpy(Xz.astype(np.float32))
ytorch = torch.from_numpy(y.astype(np.int64))
train = torch.utils.data.TensorDataset(Xztorch, ytorch)
```

[3]:
```
# model (explicit dim in log_softmax)
class LogisticRegression(nn.Module):
    def __init__(self, D, C):
        super(LogisticRegression, self).__init__()
        self.weights = nn.Parameter(torch.randn(D, C) / math.sqrt(D))
        self.register_parameter("W", self.weights)

    def forward(self, x):
```

```
        out = torch.matmul(x, self.weights)
        return F.log_softmax(out, dim=1)

# opt_pytorch that returns Python float from objective()
def opt_pytorch(learning_rate=0.01, batch_size=100, optimizer_name="Adam"):
    model = LogisticRegression(D, 2)
    criterion = torch.nn.NLLLoss(reduction="sum")
    optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate) if␣
 ↪optimizer_name.lower()!="sgd" else torch.optim.SGD(model.parameters(),␣
 ↪lr=learning_rate)

    # Create loader here (will be used by update)
    train_loader = torch.utils.data.DataLoader(train, batch_size=batch_size,␣
 ↪shuffle=True)

    def objective(_):
        model.eval()
        with torch.no_grad():
            outputs = model(Xztorch)
            loss = criterion(outputs, ytorch)
        return float(loss.item())   # <-- IMPORTANT: return Python float

    def update(_1, _2):
        model.train()
        for examples, labels in train_loader:
            outputs = model(examples)
            loss = criterion(outputs, labels)
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()
        W = model.state_dict()["W"]
        w = W[:, 1] - W[:, 0]
        return w

    return (objective, update)
```

[4]:
```
# Create object and run optimize safely with a numeric eps0
learning_rate = 0.01
batch_size = 100
obj_up = opt_pytorch(learning_rate=learning_rate, batch_size=batch_size,␣
 ↪optimizer_name="Adam")

# run optimization with a numeric eps0
wz_t, vz_t, ez_t = optimize(obj_up, None, nepochs=100, eps0=0.01, verbose=True)
print("Finished optimize; wz_t shape:", np.shape(wz_t))
```

```
Epoch   0: f=  2970.833, eps=0.010000000
Epoch   1: f=   876.598, eps=0.010500000
Epoch   2: f=   777.385, eps=0.011025000
Epoch   3: f=   739.052, eps=0.011576250
Epoch   4: f=   719.818, eps=0.012155063
Epoch   5: f=   707.682, eps=0.012762816
Epoch   6: f=   701.013, eps=0.013400956
Epoch   7: f=   694.605, eps=0.014071004
Epoch   8: f=   689.833, eps=0.014774554
Epoch   9: f=   687.160, eps=0.015513282
Epoch  10: f=   684.529, eps=0.016288946
Epoch  11: f=   682.617, eps=0.017103394
Epoch  12: f=   681.271, eps=0.017958563
Epoch  13: f=   679.351, eps=0.018856491
Epoch  14: f=   677.842, eps=0.019799316
Epoch  15: f=   676.719, eps=0.020789282
Epoch  16: f=   676.463, eps=0.021828746
Epoch  17: f=   675.279, eps=0.022920183
Epoch  18: f=   674.493, eps=0.024066192
Epoch  19: f=   673.347, eps=0.025269502
Epoch  20: f=   672.591, eps=0.026532977
Epoch  21: f=   672.269, eps=0.027859626
Epoch  22: f=   671.803, eps=0.029252607
Epoch  23: f=   671.111, eps=0.030715238
Epoch  24: f=   670.918, eps=0.032250999
Epoch  25: f=   669.548, eps=0.033863549
Epoch  26: f=   669.445, eps=0.035556727
Epoch  27: f=   669.549, eps=0.017778363
Epoch  28: f=   668.135, eps=0.018667282
Epoch  29: f=   667.969, eps=0.019600646
Epoch  30: f=   666.878, eps=0.020580678
Epoch  31: f=   666.727, eps=0.021609712
Epoch  32: f=   666.446, eps=0.022690197
Epoch  33: f=   665.698, eps=0.023824707
Epoch  34: f=   665.710, eps=0.011912354
Epoch  35: f=   665.755, eps=0.005956177
Epoch  36: f=   664.858, eps=0.006253986
Epoch  37: f=   664.673, eps=0.006566685
Epoch  38: f=   664.617, eps=0.006895019
Epoch  39: f=   663.591, eps=0.007239770
Epoch  40: f=   663.064, eps=0.007601759
Epoch  41: f=   662.684, eps=0.007981847
Epoch  42: f=   663.453, eps=0.003990923
Epoch  43: f=   663.013, eps=0.004190469
Epoch  44: f=   662.538, eps=0.004399993
Epoch  45: f=   662.643, eps=0.002199996
Epoch  46: f=   661.504, eps=0.002309996
Epoch  47: f=   661.525, eps=0.001154998
```

```
Epoch  48: f=   661.588, eps=0.000577499
Epoch  49: f=   661.521, eps=0.000606374
Epoch  50: f=   660.461, eps=0.000636693
Epoch  51: f=   660.725, eps=0.000318346
Epoch  52: f=   659.906, eps=0.000334264
Epoch  53: f=   660.336, eps=0.000167132
Epoch  54: f=   659.699, eps=0.000175488
Epoch  55: f=   659.137, eps=0.000184263
Epoch  56: f=   658.919, eps=0.000193476
Epoch  57: f=   659.151, eps=0.000096738
Epoch  58: f=   659.385, eps=0.000048369
Epoch  59: f=   658.367, eps=0.000050787
Epoch  60: f=   658.117, eps=0.000053327
Epoch  61: f=   657.285, eps=0.000055993
Epoch  62: f=   657.501, eps=0.000027997
Epoch  63: f=   657.591, eps=0.000013998
Epoch  64: f=   657.111, eps=0.000014698
Epoch  65: f=   656.612, eps=0.000015433
Epoch  66: f=   656.022, eps=0.000016205
Epoch  67: f=   655.685, eps=0.000017015
Epoch  68: f=   655.468, eps=0.000017866
Epoch  69: f=   655.439, eps=0.000018759
Epoch  70: f=   656.652, eps=0.000009380
Epoch  71: f=   654.678, eps=0.000009849
Epoch  72: f=   654.372, eps=0.000010341
Epoch  73: f=   654.074, eps=0.000010858
Epoch  74: f=   654.038, eps=0.000011401
Epoch  75: f=   654.542, eps=0.000005700
Epoch  76: f=   653.874, eps=0.000005985
Epoch  77: f=   653.736, eps=0.000006285
Epoch  78: f=   652.921, eps=0.000006599
Epoch  79: f=   653.434, eps=0.000003299
Epoch  80: f=   653.294, eps=0.000003464
Epoch  81: f=   653.160, eps=0.000003638
Epoch  82: f=   652.057, eps=0.000003820
Epoch  83: f=   652.190, eps=0.000001910
Epoch  84: f=   651.648, eps=0.000002005
Epoch  85: f=   652.870, eps=0.000001003
Epoch  86: f=   651.803, eps=0.000001053
Epoch  87: f=   652.951, eps=0.000000526
Epoch  88: f=   651.218, eps=0.000000553
Epoch  89: f=   652.019, eps=0.000000276
Epoch  90: f=   651.664, eps=0.000000290
Epoch  91: f=   651.362, eps=0.000000305
Epoch  92: f=   650.431, eps=0.000000320
Epoch  93: f=   649.468, eps=0.000000336
Epoch  94: f=   649.173, eps=0.000000353
Epoch  95: f=   649.094, eps=0.000000370
```

```
Epoch  96: f=   650.383, eps=0.000000185
Epoch  97: f=   648.478, eps=0.000000194
Epoch  98: f=   649.763, eps=0.000000097
Epoch  99: f=   649.024, eps=0.000000102
Result after 100 epochs: f=648.5194702148438
Finished optimize; wz_t shape: torch.Size([57])
```