Homework 2

(Sampled Softmax) We talked about softmax classifer in the class. Suppose there are C classes. A softmax classifier takes feature vector $\mathbf{x} \in \mathbb{R}^d$, computes logits

$$z_i = \mathbf{w}_i^\top \mathbf{x} + b_i,$$

then predicts its probability of i-th class as

$$p_i = \frac{e^{z_i}}{\sum_{j=1}^{C} e^{z_j}},$$

where $\mathbf{w}_i \in \mathbb{R}^d, b_i \in \mathbb{R}, i = 1, \ldots C$ are trainable parameters. They are trained by minimizing a cross entropy loss. Specifically, a datum $\mathbf{x}$ with label c incurs training loss

$$\ell = -\log p_c.$$

And we update the trainable parameters via (stochastic) gradient descent.

1. (10') Revisit the class notes and derive the gradients, $\frac{\partial \ell}{\partial \mathbf{w}_i}$, $\frac{\partial \ell}{\partial \mathbf{b}_i}$. Express them as function of p_i.
2. (10') The denominator of p_i requires to compute C terms. That is, $e^{z_j}, j = 1, \ldots, C$. When C is big, this can incur tremendous computational cost.

   Sampled softmax alleviates this by randomly sampling K ($K \ll C$) of these terms to approximate p_i. Specifically, we choose a distribution with probabilty mass function q over the C classes. We draw K class ID's from q. Denote this set of sampled class ID's as $\mathcal{S}$, and assume class i itself is excluded from $\mathcal{S}$. We can then approximate the denominator by

   $$\sum_{j=1}^{C} e^{z_j} \approx e^{z_i} + \frac{1}{K} \sum_{j \in \mathcal{S}} q_j e^{z_j}.$$

   Then p_i is approximated by

   $$\tilde{p}_i = \frac{e^{z_i}}{e^{z_i} + \frac{1}{K} \sum_{j \in \mathcal{S}} q_j e^{z_j}}.$$

   The approximated training loss is therefore

   $$\tilde{\ell} = -\log \tilde{p}_c.$$

   Derive the gradient $\frac{\partial \tilde{\ell}}{\partial \mathbf{w}_i}$, $\frac{\partial \tilde{\ell}}{\partial \mathbf{b}_i}$.

(Linear classifier v.s. MLP) In this exercise, we compare softmax classifier with and without MLP feature extractor. Please attach all code.

We will use MNIST dataset through out. MNIST are 28×28 images of hand written digits (See an illuatration in Figure 1). The training partition has 60,000 images, and test partition has 10,000 images. Use the following code snippet to further split the training paritition into a training set and validation set.

```
import torch
from torchvision import datasets
train_all= datasets.MNIST('../data', train=True, download=True) # 60K images
train_data, val_data = torch.utils.data.random_split(
  train_all, [50000, 10000], torch.Generator().manual_seed(0)) # train: 50K; val: 10K
test_data = datasets.MNIST('../data', train=False) # test: 10K
```

Figure 1: Examples of images in MNIST dataset. Source: wikipedia

1. (30') build a 10-class softmax classifier on the images. Train the classifier via storchastic gradient descent, and report test accuracy.
2. (40') Insert one hidden layer with 1024 hidden units before the softmax classifier. And use ReLU as the activation function at the hidden layer. Train and report test accuracy.
3. (10') Let us count the number of learnable parameters in the above model:
   - input-to-hidden-layer weight matrix: $28^2 \times 1024$
   - input-to-hidden-layer bias: 1024
   - softmax classifer weight matrix: 1024×10
   - softmax classifier bias: 10

   So the total number of learnable parameter is

   $$28^2 \times 1024 + 1024 + 1024 \times 10 + 10 = 814,090.$$

   Now, instead of inserting one hidden layer, we insert $L (L \geq 2)$ hidden layers, each with equal number of hidden units. We keep the total learnable parameters at $814,090$. Derive the number of hidden units per layer. Express it as a function of L.
4. (bonus 50') Train a MLP model with architecture defined in problem 3, where $L = 2, 3, \ldots, 8$. Get test accuracy for each L. Note that in problem 2, we already get the accuracy when $L = 1$. Plot the accuracy against L, where $L = 1, 2, 3, \ldots, 8$.