Introduction

The objective of this session is to implement a multi-layer perceptron with one hidden layer from scratch and test it on MNIST.

1. Activation function

Write the two functions
def sigma(x)
def dsigma(x)
that take as input a float tensor and returns a tensor of same size, obtained by applying component-wise respectively tanh, and the first derivative of tanh.

Hint: The functions should have no python loop, and use in particular torch.tanh, torch.exp, torch.mul, and torch.pow. My versions are 34 and 62 character long.

2. Loss

Write the two functions
def loss(v, t)
def dloss(v, t)
that take as input two float tensors of same dimensions with v the predicted tensor and t the target one, and return respectively ||t − v||^2 , and a tensor of same size equal to the gradient of that quantity as a function of v.

Hint: The functions should have no python loop, and use in particular torch.sum, torch.pow. My versions are 48 and 40 character long.

3. Forward and backward passes

Write a function
def forward _pass(w1, b1, w2, b2, x)

whose arguments correspond to an input vector to the network, and the weight and bias of the two layers, and returns a tuple composed of the corresponding x(0) , s(1) , x(1) , s(2) , and x(2) .

Write a function
def backward _pass(w1, b1, w2, b2, t, x, s1, x1, s2, x2, dl _dw1, dl _db1, dl _dw2, dl _db2)

whose arguments correspond to the target vector, the quantities computed by the forward pass, and the tensors used to store the cumulated sums of the gradient on individual samples, and update the latters according to the formula of the backward pass.

Hint: The functions should have no python loop, and use in particular torch.t, torch.mv, torch.mm, and torch.view, and the functions previously written. The main difficulty is to deal properly with the tensor size and transpose.

4. Training the network

Write the code to train and test a MLP with one hidden layer of 50 units. This network should have an input dimension of 784, which is the dimension of the MNIST training set, and an output dimension of 10, which is the number of classes.

You code should:
1. Load the data using the provided prologue.load _data function, with one-hot label vectors and normalized inputs. Multiply the target label vectors by $\zeta = 0.9$ (so that they are strictly in the value range of tanh).
2. Create the four weight and bias tensors, and fill them with random values sampled according to $N(0, eps)$ with $eps = 1e-6$.
3. Create the four tensors to sum up the gradients on individual samples, with respect to the weights and biases.

4. Perform 1, 000 gradient steps with a step size η equal to 0.1 divided by the number of training samples.
Each of these steps requires to reset to zero the tensors for summing up the gradients, and doing a forward and a backward pass for each training example.
Compute and print the training loss, training error and test error after every step using the class of maximum response as the predicted one.
Hint: My solution is 1987 character long and achieves 3.6% training error and 15.70% test error with 50 hidden units. using the default small sets of prologue.load_data.