National University of Singapore
School of Computing
CS5242: Neural Networks and Deep Learning
Tutorial 2:Shallow Networks
Pytorch autograd - Linear and Logistic Regression

Pytorch has an autograd feature which is essential for Deep Learning, as it is required for the framework to automatically figure out gradients of loss functions w.r.t all the parameters of any model.

In this tutorial, we will learn the PyTorch Autograd feature and use it to solve linear regression and logistic regression problems.

NOTE: For the whole notebook perform all your operation and instantiate all the variables on a GPU. (Google collab allows you to make environments with a GPU)

Q1. We consider the linear regression case, where we have data `x, y` and we want to find `w, b` so that the data follows `y = x * w + b` approximately.

1. Create a random normal tensor x having dimension `(10, 3)`, where `10` denotes sample size and `3` denotes feature size.
2. Create two randn tensors `w_true, b_true` having size `3 and 1` respectively. Multiply `5` with `w` and `-2` with `b`.
3. Generate tensor `y_true` by using the equation `y_true = x * w_true + b_true`

Note that this value of `w_true` and `b_true` will not be used henceforth, rather it is only being used to generate the data for our regression problem, and to compare at the end our fitted `w, b` with the true values we have here.

Q2. Now define two `randn` tensor `w, b` of the same size as `w_true and b_true`, but with `requires_grad` switched on. Calculate a tensor `y = x * w + b`, check whether this `y` has `requires_grad` switched on.

Q3. Define a loss function loss which takes w and b as arguments and returns loss which is squared error $= \sum(y - y_{true})^2$, where `y = x * w + b`.

Q4. Create a notebook cell with the following tasks

1. Call the loss function and check whether the calculated loss has `requires_grad` switched on
2. Call `.backward()` for the loss. Check gradients of `w` and `b` after this.

3. At this point repeatedly execute this cell, print values of `loss, w, b` and also the gradients of `w` and `b`. Notice the changes in the gradient value of `w, b` and no change in `loss, y, w, b.`
4. Now manually set the gradients of `w` and `b` to zero, and re-execute this whole cell multiple times and check the gradients

Q5. Linear Regression with shallow networks. Tie all the operations together in a different cell with the following steps

1. Re-initialize `w` and `b` with random numbers.
2. Calculate loss involving unknown parameters `w, b`
3. Calculate gradient by calling backward on loss
4. Use gradients to update values of parameters (gradient descent update: Keep learning rate `0.01`)
5. Set gradients to zero
6. Go to step 2 until convergence (value of the loss is less than tolerance, set the tolerance to `1e-5`)
7. Check whether the value of `w` and `b` is close to the true values, i.e. `w_true` and `b_true`

Q6. Now we will change the problem to a Logistic Regression Problem

1. Change `y_true` to be `1` if `y_true > 0` else `0`, make it a float tensor and make sure it is on the device.
2. Define a new loss function to include a sigmoid transformation of prediction `y`, to make it a probability. Change the loss to cross-entropy loss for binary classification with probability y.
3. Repeat the steps of Q5 for this problem and check the final values of w and b, make stopping criteria to be loss value `<= 0.05`. Keep learning rate the same as before.
4. You might notice that you do not recover the `w_true` and `b_true` value.