Special Topics: Deep Learning

Sheet 1 — COMP 499/691

Assignment 1

Due Date: March 8
Submission: Submit a pdf of your written answers and code and output from programming portions. You will also submit a runnable ipynb (single file) including all answers to programming portions of the questions.
Note: The assignments are to be done individually. You may discuss ideas regarding questions with others but should not share answers. List the names of anyone you have extensively discussed a question with at the top of your final submission.

1. (a) **(5 points)** Consider the 1-hidden layer neural network

$$\hat{y} = \boldsymbol{w}_2^T \rho(\boldsymbol{W}_1 x + \mathbf{b}_1) + \mathbf{b}_2$$

where $\boldsymbol{W}$ is 20×10 and $\boldsymbol{w}_2$ is a vector of size 20. The loss function is given

$$l(\hat{y}, y) = \log(\cosh(\hat{y} - y))$$

Consider the cost function $J = \frac{1}{N}\sum_{i=1}^{N} l(\hat{y}^i, y^i)$
Derive an expression for $\frac{\partial J}{\partial \boldsymbol{W}_1}$, $\frac{\partial J}{\partial \boldsymbol{w}_2}$,$\frac{\partial J}{\partial \boldsymbol{b}_1}$, $\frac{\partial J}{\partial \boldsymbol{b}_2}$ and $\frac{\partial J}{\partial x}$.

(b) **(8 points)** Take $\rho(x) = \tanh(x)$. Implement in pytorch this network and loss (using torch tensors and functionalize, but not nn.module). Validate with test cases the gradients computed by pytorch match those of your answer in (a). To do this use a randomly generated x, y with components from uniform or gaussian distribution.

(c) **(8 points)** Use finite differences method to further validate the gradients from (a) and (b). You should implement this yourself without reliance on pre-existing the torch gradchecking

(d) **(5 point)** Train this model on the sklearn Boston house prices dataset. This is accessible via `sklearn.datasets.load_boston`. For this you may use the optimizer and learning rates of your choice and train for 15 epochs. You are free to use the `torch.optim` package for this part. Report the mean squared error on the train and test set after each epoch. You will need to adjust the size of $\boldsymbol{W}_1$ to fit the size of this data.

2. (a) Consider the neural network

$$f(x) = \boldsymbol{W}_F\rho \circ \boldsymbol{W}_L \dots \rho \circ \boldsymbol{W}_i \dots \rho \circ \boldsymbol{W}_2\rho \circ \boldsymbol{W}_1 x$$

where $\boldsymbol{W}_1$ is $K \times D$, $\boldsymbol{W}_i$ is $K \times K$, and $\boldsymbol{W}_F$ is $P \times K$. Note $f : R^D \to R^P$. Take $\rho(x) = \tanh(x)$

(b) **(5 points)** Consider the case $D = 2, K = 30, P = 10$ use torch autograd to write a function which computes the Jacobian, $\frac{\partial f(x)}{\partial x}$, for a given value of x and $W_1, \dots, W_i, \dots, W_F$ where the given matrices are specified by a dictionary of torch tensors. Implement and test this for $L = 3$

(c) **(12 points)** Implement a function using torch tensors and *forward mode automatic differentiation* to compute $\frac{\partial f(x)}{\partial x}$. Validate (with assert statements) for several test cases that your answer matches the function (b) for $L = 3$.

(d) **(4 points)** Benchmark the forward/backward pass of (b) compared to the forward/forward pass of (c) for L=3,5,10. Report speed of these answers on test cases using GPU and CPU.

(e) **(4 points)** Assume matrix multiply operations between sizes $M_1 \times M_2$ and $M_2 \times M_3$ result in $M_1 * M_2 * M_3$ ops. Briefly discuss the theoretical speed comparisons of (b) and (c).

3. For the following functions find by hand the parameters of a neural network that can fit these functions. You should use either a 1 or 2 hidden layer network and may use either sigmoid or ReLU non-linearities. In each case justify your answer and how you arrived at it (without using numerical tools/software packages).

   1. (**3 points**) The XNOR function
   2. (**4 points**)

$$f(x) = \begin{cases} 2x & \text{when } 0 \le x \le 1/2, \\ 2(1-x) & \text{when } 1/2 \le x \le 1, \\ 0 & \text{otherwise,} \end{cases}$$

   3. (**4 points**) $f(x) = \max(|x| - s, 0) * sign(x)$ where s is a constant.

4. (a) (**5 points**) We will study different ways to initialize models. First create a nn.module that provides a variable number of feedforward layers and takes MNIST digits as inputs. The module should allow constructing a network as follows net = `my_model(depth)`. It may be helpful to use the `nn.ModuleList` construction. For the rest of the exercise we will use a width of 50 hidden units. The input layer of your network should take minibatches of data sized $B \times 784$ where B is the batch size and the output should have 10 values. For this network use a tanh non-linearity.

   (b) (**2 points**) Write a function to initialize your model $w \sim \mathcal{U}(-d, d)$ and biases to zero. We will study the 4 cases $d = 0.01, 0.1, 2.0, \sqrt{\frac{6}{n_i+n_o}}$. Note the final value corresponds to xavier initialization with n_i, n_o the number of input and output connections to a unit. You will perform part (c) and (d) for all 4 values of d and depth 8. This function can be a member of the your model class. You may make use of the built in `torch.nn.init` functions to achieve this.

   (c) (**8 points**) Forward and Backward a minibatch of 256 MNIST digits through the network with depth 8. Compute and plot mean post-activation values at each layer. Now compute and visualize the gradient norm at each layer in a similar fashion. Your plots should have layer on the x-axis and gradient norm or mean activation on the y axis. Each of the 2 plots should show lines for each of the 4 initialization settings of d. Note to get the gradient norms at each layer you can use `retain_grad` on the layer outputs in the forward pass to keep the gradient buffer from clearing at each layer on the backward.

   (d) (**8 points**) For each of the initialization settings train the model for 5 epochs on MNIST, using the cross-entropy loss. You may use SGD with learning rate of 0.01 and minibatch sizes of 128. Record the training accuracy and testing accuracy after each epoch and plot them versus epochs. At the end of training repeat (c) for the trained models.

   (e) (**2 points**) Briefly (max 1 paragraph) discuss your findings, how does depth and initialization affect the activations, gradients, and convergence.

5. (a) (**3 points**)Consider the squared loss $\mathcal{L}(X, w, y) = \frac{1}{2}\|Xw - y\|^2$ for data matrix X of size $N \times D$ and parameters w. y is a vector of labels size N. Find the expression for gradient $\nabla_w \mathcal{L}(X, w, y)$ and minimizer of this loss, $\arg\min_w \mathcal{L}(X, w, y)$

   (b) (**3 points**) Take w_0 as the initialization for gradient descent with step size α and show an expression for the first and second iterates w_1 and w_2 only in terms of α, w_0, X, y.

   (c) (**Extra Credit: 4 points**) Generalize this to show an expression for w_k in terms of α, w_0, X, y, k