Special Topics: Deep Learning

Sheet 1 — COMP 691 Winter 2023

Assignment 1

Due Date: March 1

Submission Instructions: **Follow the submission instructions carefully or your assignment may not be fully graded.**

1) Submit a SINGLE pdf of your written answers and code and output from programming portions. Make sure to include all 3 in the pdf, do not asssume we will re-run your code, do not submit jpeg files, do not submit multiple pdfs.

2) It is suggested (but not required) you submit a runnable ipynb (single file) including all answers to programming portions of the questions. This will be used if there are doubts but assume this ipynb might not be evaluated thus all your answers should be readable from the single pdf.

3) Do not zip your file, submit a pdf directly to moodle and a ipynb

4) Clearly delineate each question in your answer

Note1: The assignments are to be done individually. You may discuss high level ideas regarding questions with others but should not share answers. List the names of anyone you have discussed a question with at the top of your final submission.

Note2: Do not post questions that implicitly give away answers in MS teams.

Note3: For clarifications or if you suspect there is a mistake/typo contact Instructor

Eugene Belilovsky: eugene.belilovsky@concordia.ca

1. (a) **(5 points)** Take $\rho(x) = \tanh(x)$. Consider the 1-hidden layer neural network

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

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

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

Consider the cost function $J = \frac{1}{N} \sum_{i=1}^{N} l(\hat{y}^i, y^i)$

Derive an expression for $\frac{\partial J}{\partial \boldsymbol{W}_1}$, $\frac{\partial J}{\partial \boldsymbol{w}_2}$,$\frac{\partial J}{\partial \boldsymbol{b}_1}$, $\frac{\partial J}{\partial \boldsymbol{b}_2}$.

(b) **(10 points)**

- 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, generate random $\{x, y\}$ samples with components from either uniform or Gaussian distribution.
- Show the difference between the corresponding matrices (gradient calculated by hand, and gradient from `torch.autograd`) using `torch.linalg.norm`.

(c) **(10 point)** Train this model on the `sklearn` California Housing Prices datasets.

- For this you may use the optimizer and learning rates of your choice and train for 20-50 epochs.
- Take half the data for training and half for testing.
- Create a validation set from the training set and use it to select a good learning rate.
- You might want to use the *convenient* Xavier initialization.
- You are free to use the `torch.optim` package for this part.
- To speed up things, run the training loop by batches (e.g. 4, 8, 32, 64, etc.). PyTorch's `DataLoader` would be a useful tool to easily fetch a predefined set of batches per training iteration.
- 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.

Eugene Belilovsky: eugene.belilovsky@concordia.ca

2. Consider the neural network

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

where $\boldsymbol{W}_1$ is $K \times D$, $\boldsymbol{W}_i$ is $K \times K$ for $i > 1$, and $\boldsymbol{W}_F$ is $P \times K$. Note $f : R^D \to R^P$. Take $\rho(x) = \tanh(x)$. We will examine different ways to compute the Jacobian $\frac{\partial f(x)}{\partial x}$.

(a) **(5 points)** Use torch tensors to write a function which computes the Jacobian, $\frac{\partial f(x)}{\partial x}$, using *backward mode automatic differentiation* for a given value of x and $W_1, \ldots, W_i, \ldots, W_F$ where the given matrices are specified by a dictionary of torch tensors. Implement and test this for $L = 3$. Your function should only make use of basic matrix operations (e.g. torch.matmul(), torch.tanh(), etc). You may not use autograd or autograd.jacobian for your implementation (but you can use them to unit test your answer). Test it for the case of $D = 2, K = 30, P = 10$, your solution does not have to cover all edge cases of K, P,D it is sufficient it works on the ones provided here.

(b) **(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 $L = 3$. Hint: You must calculate the derivatives and the network's output in the same forward pass (unlike *backward differentiation* where you need two loops, one for the forward pass and one for calculating the gradient).

(c) **(4 points)** Benchmark the Jacobian computation of (b) compared to that of (c) for L=3,5,10. Report speed of these answers on test cases using GPU and CPU.

(d) **(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).

Eugene Belilovsky: eugene.belilovsky@concordia.ca

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/software packages).

   1. (**5 points**)
   $$f(x) = \begin{cases} 2x & \text{when } 0 \leq x \leq 1/2, \\ 2(1-x) & \text{when } 1/2 \leq x \leq 1, \\ 0 & \text{otherwise,} \end{cases}$$

   2. (**5 points**) $f(x) = \max(|x| - s, 0) * sign(x)$ where s is a constant greater than 0.

   Hint: Consider a simpler example $f(x) = |x|$, a valid relu neural network that can give this function is $NN(x) = ReLU(w_1 * x + b_1) + ReLU(w_2 * x + b_2)$ where $w_1 = 1, w_2 = -1, b_1 = b_2 = 0$

Eugene Belilovsky: eugene.belilovsky@concordia.ca

4. (a) (**5 points**) We will study different ways to initialize models. First create a `nn.module` with a variable that defines the number of feedforward layers, while taking MNIST digits as inputs. The module should allow constructing a network as follows:

$$\text{net} = \texttt{my\_model(depth)}.$$

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**) Using a cross-entropy loss at the end of the network: forward and backward a minibatch of 256 MNIST digits through the network with depth 8. Compute and visualize the gradient norm at each layer. Specifically this refers to the $\left\|\frac{\partial L}{\partial a}\right\|$, where a are the post-activation outputs. Your plots should have layer on the x-axis and gradient norm on the y-axis. Note that 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. Perform this for each of the 4 initializations to obtain 4 curves. Note: in this question you do not need to train or update the models.

(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.

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

5. (a) (**6 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) (**6 points**) Generalize this to show an expression for w_k in terms of α, w_0, X, y, k

   (d) (Extra Credit: **4 points**) using the above show that gradient descent converges to the minimum in (a) as $k \to \infty$, you can make assumptions on $X^T X$ and α.

Eugene Belilovsky: eugene.belilovsky@concordia.ca