Homework 2
600.482/682 Deep Learning
Spring 2023

February 1, 2023

Due Wednesday Feb 15 11:59 pm EST

Instructions. Please submit the following two items to Gradescope (entry code DJGEWX): 1) your report (LaTeX generated PDF) to `homework2-report`, and 2) a zip file containing your Python Jupyter Notebook (.ipynb) and an exported PDF of the notebook (with all cell outputs) to `homework2-notebook`.

1. The goal of this problem is to minimize a function given a certain input using gradient descent by breaking down the overall function into smaller components via a computational graph. The function is defined as:

$$f(x_1, x_2, w_1, w_2) = \frac{1}{1 + e^{-(w_1 x_1 + w_2 x_2)}} + 0.5(w_1^2 + w_2^2)\,.$$

   (a) Please calculate $\frac{\partial f}{\partial w_1}, \frac{\partial f}{\partial w_2}, \frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}$.

   (b) Start with the following initialization: $w_1 = 0.3, w_2 = -0.5, x_1 = 0.2, x_2 = 0.4$, draw the computational graph. Please use backpropagation as we did in class.
   You can draw the graph on paper and insert a photo into your report.
   The goal is for you to practice working with computational graphs. As a consequence, you must include the intermediate values during the forward and backward pass.

2. Doing the computations above is a lot of work, already for such a simple function as in Problem 1. Clearly, it will be desirable to automate the forward and backward propagation process. In contemporary frameworks, such as TensorFlow and PyTorch, backpropagation is handled automatically for all standard operations, which is clearly very convenient. This feature is commonly referred to as "AutoGrad" and becomes possible due to one of the key observations we learned in class: difficult expressions can be broken down into simple subproblems, the analytic gradients of which can be synthesized via chain rule. In this problem, you will implement your own AudoGrad structure.

   (a) Consider the Jupyter Notebook we provide and follow the instructions therein to implement the missing operations and functionality. (To use Jupyter Notebook, we highly recommend uploading the .ipynb file to Google Colaboratory (https://colab.research.google.com/), where you may edit and run your notebook online. If that option is not feasible for you, you can create a local Python 3.6+ environment. The only required external dependencies are `numpy`, `matplotlib`, and `jupyter` which you can install using the package manager of your choosing (e.g. `pip`, `conda`))

   (b) Use the function and output derived in Problem 1 to test whether your AutoGrad structure works as intended.
   Attach a screen shot of your notebook's printed intermediate values for this test case (both during the forward and backward pass) to your report.

3. Your goal in this exercise is to implement a linear classifier using the AutoGrad class you implemented in Problem 2, train it on a 2-dimensional toy dataset, and show the results tested on the **same** dataset. Recall that we formulate a linear classifier as $f(x, W, b) = Wx + b$. To

quantify the model's performance, you will pass its output through a softmax function and then use Cross Entropy loss to measure agreement with the target output.

Write a Python program based on your AutoGrad structure that iteratively computes a small step in the direction of negative gradient. The dataset can be downloaded here (`https://piazza.com/class_profile/get_resource/ld1ozfomn4j2bs/ldlwvh49d035jz`). **Please go through the README.txt carefully before you start.** Initialize your optimization with all elements of W being small random numbers and $b = 0$. As shown in the visualization of the data provided in the link above, you can reasonably expect to see your algorithm converge to a solution that linearly separates the given data samples.

(a) Please plot the following:
   i. Cross Entropy loss with respect to training iterations (loss curve)
   ii. Training accuracy with respect to training iterations (accuracy curve)
   iii. Visualization of the ground truth labels and your model's decision boundaries (decision boundary visualization)

(b) Report your classification accuracy. Briefly discuss your result. Does it match your expectation?

4. In this problem, we would like you to compare the performance of single-layer and multi-layer classifiers, again using your AutoGrad structure.

   (a) Please download the dataset for this question here (`https://piazza.com/class_profile/get_resource/ld1ozfomn4j2bs/ldlwvms0c935xn`). This data is two dimensional. Create another linear model (same as Problem 3) for this dataset and run your classification training.

   Report your classification accuracy. As in Problem 3, please attach plots of the 1) loss curve, 2) accuracy curve and 3) decision boundary visualization. Is this dataset linearly separable?

   (b) Now add one more layer of perceptrons.* Use ReLU as activation functions for the first hidden layer.** Connect outputs with a softmax function. Initialize your optimization procedure with all elements of W being small random numbers and $b = 0$. Use Cross Entropy loss. Train this model on the same two dimensional dataset in (a).

   Report your classification accuracy. Attach plots of the 1) loss curve, 2) accuracy curve and 3) decision boundary visualization. Does it perform better than (a)?

   (c) Briefly discuss your results.

* If the output of a single layer is $f(x, W_1, b_1)$, then adding another layer basically means that the output is $g(f(x, W_1, b_1), W_2, b_2)$: the second layer uses the first layer's output as its input. Therefore, the output dimension of f should match the expected input dimension of g. In our case, the output dimension from the ReLU activation in the first layer should match the input dimension of the second layer.

** The ReLU activation function is a nonlinear function defined as:

$$\text{ReLU}(x) = \max(0, x)$$

If we use ReLU as our activation function for the first layer, instead of $f(x, W_1, b_1) = W_1x+b$, we have

$$f(x, W_1, b_1) = \text{ReLU}(W_1x + b) = \max(0, W_1x + b)$$

The non-linearity of the ReLU function adds non-linearity to the model.