ECE/CS 559: Neural Networks **Fall 2024**

Homework 5

Due: **Tuesday** October 1, 2024 (by 9pm, on Gradescope)

Note: *You may discuss these problems in groups. However, you must write up your own solutions and mention the names of the people in your group. Also, please do mention any solution manuals, online material, or other sources you used in a major way.*

Submit your code as a **two** `.py` **files**, one for each question, in Gradescope. Make sure that it **runs properly** and generates all the plots that you report on in the main submission.

1. **[Gradient Descent]**

   Let $\mathrm{w} \in \mathbb{R}^2$. Consider the following function:

   $$R(\mathrm{w}) = 13w_1^2 - 10w_1w_2 + 4w_1 + 2w_2^2 - 2w_2 + 1$$

   (a) Find the gradient of R and solve for the optimality condition exactly.

   (b) Implement gradient descent. At each iteration, calculate and save the distance between the iterate and the optimal solution. Run this with $\eta = 0.02, 0.05$, and 0.1 and 500 iterations each. Plot distance vs. iteration in each case. Report it.

   (c) Is even larger η beneficial? Why or why not?

2. **[Backpropagation]**

   Revisit the dataset you generated in HW2 Q2. Imagine that a friend gives you the 1,000 points generated with that neural network, tells you what the architecture looks like (but not the weights), then asks you to train a neural network that imitates theirs.

   (a) You want to use gradient descent, but know that you can't use a step function to do the training. Choose instead to use a sigmoid function φ with parameter $a = 5$. Write down φ and φ' explicitly and implement Python functions for each, that operate on `numpy` arrays.

   (b) Call the first layer weights W and its biases b, the second layer weights U and its bias c, and the output f. Let the hidden layer output be z. Write down the dimensions of all weights, biases, and variables, then write down the forward equations from x to v_z, from v_z to z, from z to v_f, and from v_f to f. Translate these into Python code.

(c) Use the squared loss $\ell(y, f) = (f - y)^2$. Since we are targeting binary outputs and φ limits values to $[0, 1]$, this is not that different from 0-1 loss. Start writing down the backward equations for a single data point, by first writing the expression for $\nabla_f \ell$ followed by the expression for δ_f. Then, write the backward equation from δ_f to δ_z, then from δ_z to δ_x. Use these along with x and z to compute the gradients $\nabla_W \ell$, $\nabla_b \ell$, $\nabla_U \ell$, and $\nabla_c \ell$. (Remember that these gradients should have the same dimensions as the respective weights and biases.) Translate these into Python code.

(d) In Python, initialize all your weights and biases with i.i.d. Gaussian$(\mu = 0, \sigma = 0.1)$ samples. Then, create an outer `epoch` loop and inner `i` loop ranging over the $1,000$ data points. For each data point (x, y), run the forward pass code from (b), then the backward pass code from (c). Then, perform gradient descent with that single point (this is a form of *stochastic gradient descent, SGD*): $W \leftarrow W - \eta \nabla_W \ell$, etc. Use $\eta = 0.01$ and perform 100 epochs. At the end of each epoch, calculate and save the MSE. At the end, plot and report the MSE vs. epoch curve.

(e) In HW2 Q2, you visualized the decision boundary of your friend's neural network. Visualize the decision boundary of yours, by plotting a 3D scatter plot of $(x_1, x_2, f(x_1, x_2))$ using the weights at the end of your training. Report the plot. (You can also visualize this during the training, to see how the boundary evolves.) You may find the following snippet useful.

```
fig = plt.figure(figsize = (7, 10))
ax = plt.axes(projection ="3d")
ax.scatter3D(x1, x2, y_predicted, color = "green")
ax.set_xlabel('x1')
ax.set_ylabel('x2')
ax.set_zlabel('y')
ax.zaxis._axinfo['juggled'] = (2, 2, 1)
plt.show()
```

(f) Try at least three of the following hacks individually or simultaneously. Say what you tried, then report on the MSE vs. epoch curve, the decision boundary, a description of what changed, and your hypothesis as to why.

- Change η.
- Start η high but multiply by 0.9 if the MSE increases (or if it increases for T epochs, where you choose T).
- Choose a different a in φ.
- Change the number of epochs that you perform.
- Do a proper gradient descent (accumulate gradients, update only in the end of an epoch, zero-out gradients, and move to next epoch).
- Use minibatches (accumulate B gradients, update, zero-out, and continue).
- Change the architecture by including more or fewer hidden neurons.
- Change how you initialize the weights and biases, e.g. by setting them to 0 or choosing σ very small or large.