Error Gradients and Error Backpropagation

Pencil and Paper

1. Given the scalar functions $F(x)$ and $G(x, \omega)$, where $x, \omega \in \mathbb{R}$, consider the composite operation

$$y = F\Big(G(x, \omega)\Big) .$$

If you are given D samples, $x^{(i)}, i = 1, \ldots, D$, define E to be the expected value of y,

$$E = \frac{1}{D}\left[\sum_{i=1}^{D} F\left(G(x^{(i)}, \omega)\right)\right] .$$

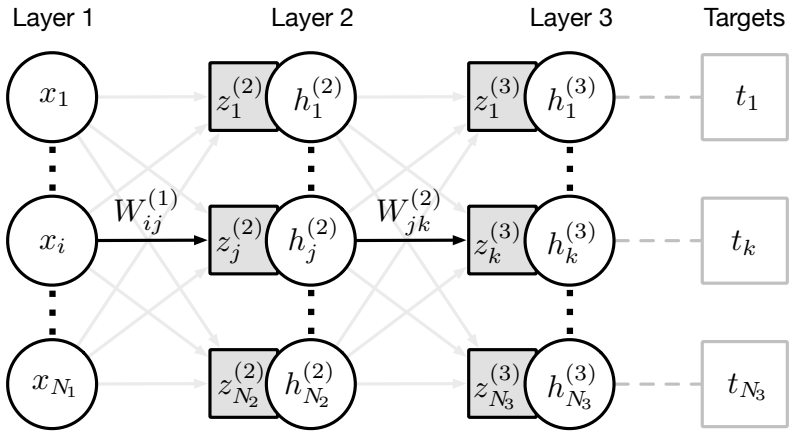
Which of these two is the correct expressions for $\frac{\partial E}{\partial \omega}$,

$$\text{(a)} \quad \frac{1}{D}\sum_{i=1}^{D} F'\left(G(x^{(i)}, \omega)\right)\frac{\partial G(x^{(i)}, \omega)}{\partial \omega} \qquad \text{(b)} \quad \frac{1}{D}F'\left(\sum_{i=1}^{D}\frac{\partial G(x^{(i)}, \omega)}{\partial \omega}\right)$$

where $F'(x) = \frac{\partial F(x)}{\partial x}$? Justify your answer.

Hint: If you find it helpful, you can define $g^{(i)} = G(x^{(i)}, \omega)$, and $f^{(i)} = F(g^{(i)})$.

2. Consider the neural network shown below, with one input layer, one hidden layer, and one output layer (with targets). For each neuron, the input current is shown in the gray box, and the neuron's activity is in the circle. Moreover, $h_j^{(\ell)}$ is the activity of node j in layer ℓ. The input current follows the same labelling format, so that $h_j^{(\ell)} = \sigma(z_j^{(\ell)})$.



In addition, let $\mathbf{z}^{(\ell)} \in \mathbb{R}^{1\times N_\ell}$ and $\mathbf{h}^{(\ell)} \in \mathbb{R}^{1\times N_\ell}$ be row-vectors to store an entire layer of input currents and activations, respectively. Given the connection weights $W^{(\ell)}$, the input current for the next layer is $\mathbf{z}^{(\ell+1)} = \mathbf{h}^{(\ell)}W^{(\ell)}$. Similarly, define $\mathbf{x}$ and $\mathbf{t}$ as row-vectors.

Consider a dataset consisting of input/target pairs $(\mathbf{x}_i, \mathbf{t}_i), i = 1, \ldots, D$. Define X and T as matrices that contain one sample per row,

$$X = \begin{bmatrix} \mathbf{x}_1 \\ \hline \vdots \\ \hline \mathbf{x}_D \end{bmatrix} \qquad T = \begin{bmatrix} \mathbf{t}_1 \\ \hline \vdots \\ \hline \mathbf{t}_D \end{bmatrix}$$

Likewise, define $Z^{(\ell)}$ and $H^{(\ell)}$, such that $Z^{(\ell)}_{d,j}$ and $H^{(\ell)}_{d,j}$ are the input current and activity (respectively) of the jth node in layer ℓ for the dth sample. Notice that the subscripts follow the row-column convention for matrix indexing.

Suppose the loss for a single sample d is $L(\mathbf{h}^{(3)}_d, \mathbf{t}_d)$, and the cost function for the whole dataset is

$$E = \mathbb{E}\Big[L(\mathbf{h}^{(3)}_d, \mathbf{t}_d)\Big]_{d=1,\ldots,D} \ ,$$

where $\mathbb{E}$ denotes the expectation over the dataset.

(a) Given X, the matrix of inputs, write a **single formula** that computes $H^{(2)}$, the activities of the hidden layer **for all the input samples**.

(b) Suppose you have already computed $H^{(2)}$. Write a **single formula** that computes $H^{(3)}$, the activities of the output layer **for all the samples**.

(c) Derive a **single expression** for $\nabla_{Z^{(3)}} E$, the $D \times N_3$ matrix that is the gradient of E with respect to the input currents to the output layer, for all samples.

$$\nabla_{Z^{(3)}} E = \begin{bmatrix} \frac{\partial E}{\partial Z^{(3)}_{1,1}} & \cdots & \frac{\partial E}{\partial Z^{(3)}_{1,N_3}} \\ \vdots & \ddots & \vdots \\ \frac{\partial E}{\partial Z^{(3)}_{D,1}} & \cdots & \frac{\partial E}{\partial Z^{(3)}_{D,N_3}} \end{bmatrix}$$

(d) Assuming you already computed $\nabla_{Z^{(3)}} E$, show that $\nabla_{W^{(2)}} E$, the $N_2 \times N_3$ matrix containing the gradient of E with respect to the connection weights in $W^{(2)}$, can be computed using,

$$\nabla_{W^{(2)}} E = \left(H^{(2)}\right)^{\mathrm{T}} \cdot \nabla_{Z^{(3)}} E \ .$$

Do Some Coding

For these exercises, you will want to get the jupyter notebook `ex03.ipynb` and the file `utils.py`.

3. Review the **Working with Datasets** section of the notebook, where it demonstrates the functionality of the `DiscreteMapping` class. For more information on the class, see its documentation in the `utils.py` module. Notice that the code creates a dataset where the inputs are 2D row vectors, grouped into 2 classes. The targets are one-vectors, where the classes are labelled with 0 and 1. Try plotting the dataset, and listing its inputs and targets.

4. Review the section on the `Operation` class, an abstract base class from which activation functions and loss functions are derived.

   (a) The `Identity` and `Logistic` classes implement activation functions, and their derivatives. You'll remember these from last week's assignment (and the complete code for the `Logistic` and `CrossEntropy` classes will be released after the assignment is due).

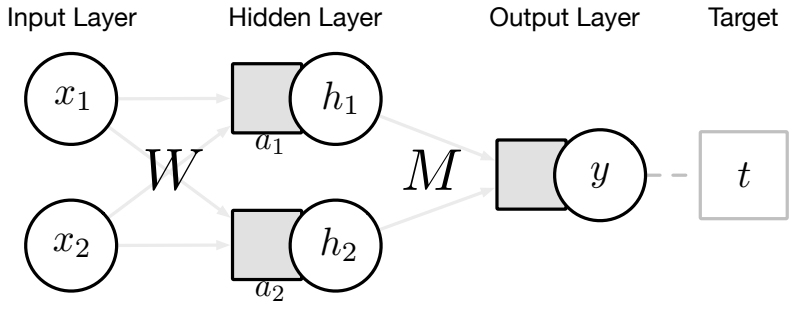
   To use these classes, you first create an instance of the class, and then use that instance as the function. For example, suppose you have input currents stored in `z`. Then,

   ```
   act = Logistic()
   h = act(z)
   dhdz = act.derivative()
   ```

   applies the logistic function to those currents, and then calculates the corresponding derivatives of the function. Try applying the activation functions to the inputs from the dataset.

   (b) The `MSE` and `CrossEntropy` classes implement the expected squared error and expected cross-entropy (respectively), and their derivatives. See their documentation for more information. Create a random y and try computing the expected loss between y and the dataset targets. Again, the full implementation of `CrossEntropy` will be available once the assignment is due.

5. Consider the neural network shown below, with two input nodes, two hidden nodes, and one output node. For each neuron, the input current is shown as the gray box, and the parameter below the gray box shows the bias (note that there is no bias for the output node). Each neuron's activity is denoted by the variable inside the circle. All the hidden nodes and the output node use a logistic activation function (the input layer uses the identity). The connection weights are W and M.



Write a program that, given a small classification dataset, uses gradient descent to learn the connection weights (W and M) and biases (a) that correctly classifies the inputs. Use the classes already defined above to help you keep your code clean and concise.

Implementing Neural Networks

The remaining questions will guide you to understanding the neural-network code-base. The jupyter notebook contains a number of useful classes for implementing neural learning.

6. The `Layer` class is an abstract base class for a number of different derivative classes: `Population`, `Connection`, and `DenseLayer`. See their documentation for more information.

   (a) The `Population` class represents **a collection of neurons**, along with their activation function. The layer can take input currents as input, and outputs neuron activities. Try creating a `Population` object and evaluate it for some input currents.

   (b) The `Connection` class represents **the all-to-all set of connections** between two populations of neurons. Create a `Connection` object, and view its connection-weight matrix and vector of biases. Try feeding the output from your `Population` layer through the `Connection` layer.

   (c) The `DenseLayer` class is a compound layer, including one `Connection` object, followed by one `Population` object. **Edit the `__call__` function so that it computes and returns the operation of the two composed layers.** Then create a `DenseLayer` object, and try using it. Print its connection weights, and activation function.

7. Finally, the `Network` class represents an entire network, consisting of a sequence of layers. See the documentation for more details.

   Try creating a `Network` object and add some layers to it (using `net.add_layer`). Note that it is the responsibility of the user to make sure the sizes of adjacent layers make sense. You can also give the network some input and evaluate its output (eg. `y = net(x)`). Try it, and then view the contents of the layers.

   Create a small network for the dataset in question 3. Feed the dataset inputs through the network. Try evaluating the output of the network using one of the loss functions.