University of Michigan
EECS 504: Foundations of Computer Vision
Winter 2020. Instructor: Andrew Owens.

Problem Set 4: Backpropagation

Posted: Tuesday, February 4, 2020	**Due:** Tuesday, February 11, 2020

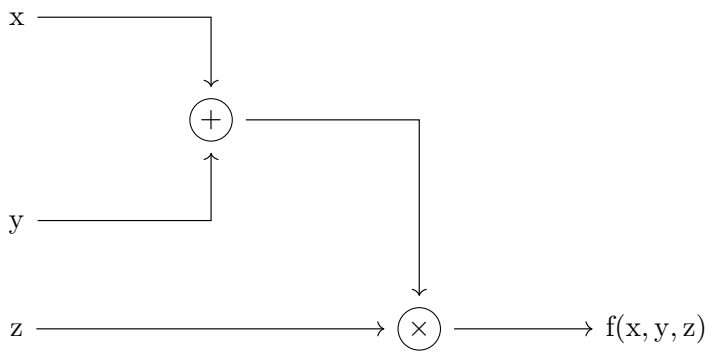
For Problem 4.1, please submit your written solution to Gradescope as a `.pdf` file. For Problem 4.2, please submit your solution to Canvas as a notebook file (`.ipynb`), containing the visualizations that we requested.

The starter code can be found at:
https://drive.google.com/open?id=1m04gqnMQXE6l0n3phzaxnViP-GVw8chA

We recommend editing and running your code in Google Colab, although you are welcome to use your local machine instead.

Problem 4.1 *Understanding backpropagation*

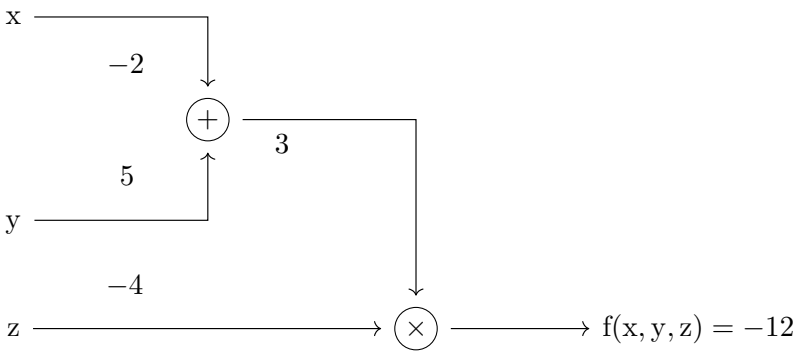
Recall that (as in Lecture 8), we can represent a formula as a computation graph, which can make it easier to reason about computing gradients. The following diagram is an example of the equation $f(x, y, z) = (x + y)z$:



(a) In the same way, given the input $\vec{x} = [x_0, x_1], \vec{w} = [w_0, w_1, w_2]$, draw a computation graph for $f(\vec{x}, \vec{w}) = \frac{1}{1+e^{-(w_0 x_0 + w_1 x_1 + w_2)}}$.

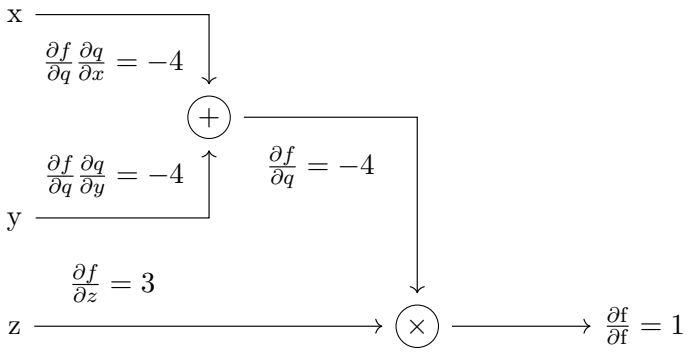
Note: Please use the following operations: $+, \times, -, +1, \exp, \frac{1}{x}$. (1 point)

(b) Given $x = -2, y = 5, z = -4$, we could calculate forward pass on the example diagram given above:



In the same way, given $\vec{w} = [1, 3, -2], \vec{x} = [5, 8]$, calculate forward pass of the equation $f(\vec{x}, \vec{w}) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$.
You can directly write those numbers on the diagram you draw in (a). (1 point)

(c) Recall that we can calculate backward pass by using chain rule, which gives us $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$ as following: (Let $q = x + y$)



In the same way, draw a new diagram and calculate $\frac{\partial f}{\partial \vec{w}}, \frac{\partial f}{\partial \vec{x}}$ on that diagram by using chain rule.

Note: You only meed to write the number below the arrow. (1 points)

Note that there can be ambiguity in how we write the computation graph: we can use primitives that have simple local gradients. Noticed that we define operation $\sigma(x)$ in the following way, which is called sigmoid:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{1}$$

(d) (Optional) Show that the derivative of sigmoid is $(1 - \sigma(x))\sigma(x)$. (0 points)

(e) Draw a new computation graph of the equation $f(\vec{x}, \vec{w}) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$ using $\sigma(x)$ as a node of the graph. Calculate forward pass and backward pass, making use of the fact you proved in (d). Comment on whether $\frac{\partial f}{\partial \vec{w}}, \frac{\partial f}{\partial \vec{x}}$ is the as same as (c). (1 point)

Problem 4.2 *Multi-layer perceptron*

In this problem, we will train a two-layer neural network to classify images. Our network will have two layers, and a softmax layer to perform classification. We'll train the network to minimize a cross-entropy loss function (also known as softmax loss). The network uses a ReLU nonlinearity after the first fully connected layer. In other words, the network has the following architecture:
1) input, 2) fully connected layer, 3) ReLU, 4) fully connected layer, 5) softmax.

More concretely, we compute class probabilities $\mathbf{c}$ from an input image $\mathbf{x}$ as:

$$\mathbf{c} = \text{softmax}(\mathbf{W_2}\ \text{relu}(\mathbf{W_1 x} + \mathbf{b_1}) + \mathbf{b_2}), \tag{2}$$

where $\mathbf{W_i}, \mathbf{b_i}$ are the parameters of the fully connected layers.

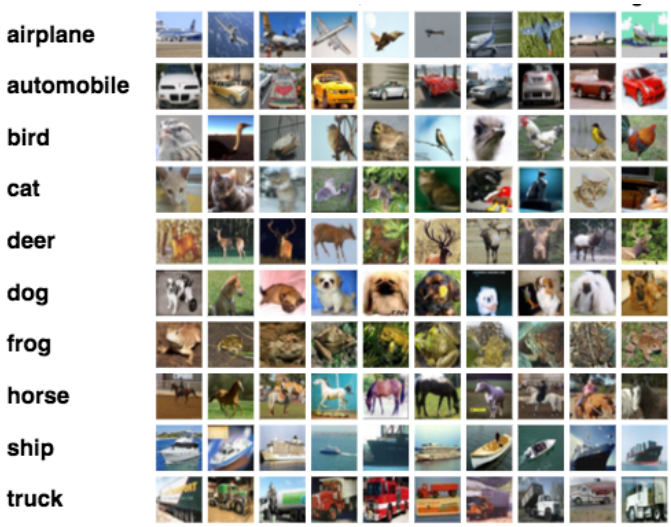


Figure 1: The CIFAR-10 dataset, which we'll be classifying in this problem. [1]

The outputs of the second fully-connected layer are the scores for each class. You should *not* use a deep learning library (e.g. PyTorch) for this problem: instead, you will implement it from scratch.

(a) Implement the fully connected, ReLU, and Softmax layers. (3 points)

Hint:

- Fully connected layer:
$$y = Wx + b \tag{3}$$
- ReLU:
$$y = \begin{cases} x, & x \geq 0 \\ 0, & \text{otherwise} \end{cases} \tag{4}$$
- Softmax:
$$y_i = \frac{e^{x_i}}{\sum_{j=1}^{N} e^{x_j}} \tag{5}$$

Note: When you exponentiate even large–ish numbers in your softmax layer, the result could be quite large and numpy would return `inf`. To avoid these numerical issues, you can first subtract the maximum value of the input to the softmax, exploiting the fact:

$$y_i = \frac{e^{x_i}/e^{\max(x)}}{\sum_{j=1}^{N} e^{x_j}/e^{\max(x)}} = \frac{e^{x_i - \max(x)}}{\sum_{j=1}^{N} e^{x_j - \max(x)}} \tag{6}$$

(b) In this problem, implement the softmax classifier class. (1 point)

(c) In this problem, you need to set up model hyperparameters (hidden_dim, learning_rate, lr_decay, batch_size.) Run the given code and report the accuracy on test set. If your method is implemented correctly, you should obtain at least 45% accuracy on the test set. (1 point)

(d) Plot the both training and validation accuracy across iterations. (0.5 point)

Reference

[1]Learning Multiple Layers of Features from Tiny Images, Alex Krizhevsky, 2009.

Acknowledgement

Part of the homework and the starter code are taken from previous EECS442 by David Fouhey and CS231n at Stanford University by Fei-Fei Li, Justin Johnson and Serena Yeung. Please feel free to similarly re-use our problems while similarly crediting us.