University of Michigan
EECS 442: Computer Vision
Fall 2020. Instructor: Andrew Owens.

Problem Set 4: Backpropagation

Posted: Wednesday, September 30, 2020 **Due:** Wednesday, October 7, 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. Before submitting, please make sure to rename the file to `uniqname_umid.ipynb`.

The starter code can be found at:
`https://colab.research.google.com/drive/1tsSLMo74X38IkjBNgntIR_4NU9uzOxrk?usp=sharing`

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*

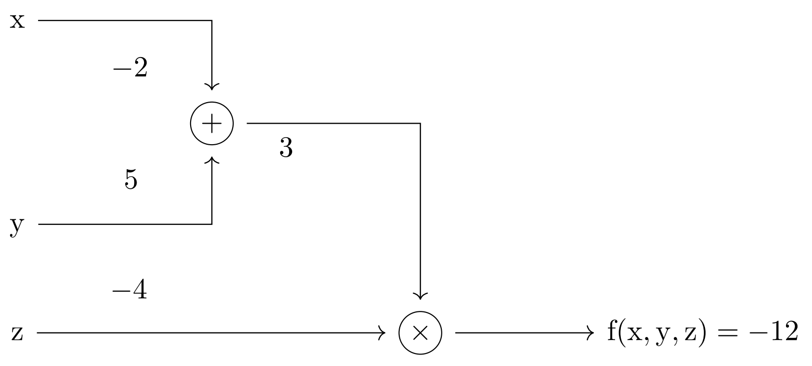
Recall that we can represent a formula as a computation graph, which can make it easier to reason about computing gradients. The following diagram is an example of the equation $f(x, y, z) = (x + y)z$:

$$\begin{tikzcd}
\text{x} \arrow[r, -] & \arrow[d] & & \\
& \oplus \arrow[r, -] & \arrow[dd] & \\
\text{y} \arrow[r, -] & \arrow[u] & & \\
\text{z} \arrow[rr] & & \otimes \arrow[r] & \text{f(x, y, z)}
\end{tikzcd}$$

(a) Given the input $\vec{x} = [x_0, x_1], \vec{w} = [w_0, w_1, w_2]$, draw a computation graph for $f(\vec{x}, \vec{w}) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$.

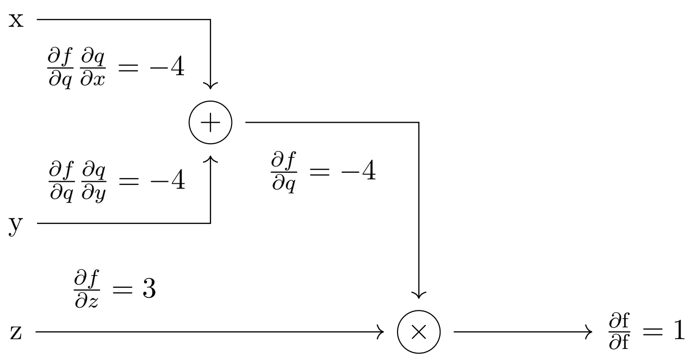
Note: Please use the following operations: $+, \times, -, +1, \exp, \frac{1}{x}$. **(1 point)**

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



Next, given $\vec{w} = [2, 1, -3], \vec{x} = [4, 1]$, 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 $q = x + y$)


x
$\frac{\partial f}{\partial q}\frac{\partial q}{\partial x} = -4$
$+$
$\frac{\partial f}{\partial q}\frac{\partial q}{\partial y} = -4$
$\frac{\partial f}{\partial q} = -4$
y
$\frac{\partial f}{\partial z} = 3$
z
$\times$
$\frac{\partial \mathrm{f}}{\partial \mathrm{f}} = 1$

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 point)**

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 point)**

(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, you'll train a two-layer neural network (a multi-layer perceprton) to recognize objects in tiny images. The model and codebase will be very similar to the linear classifier that you trained in PS3. However, instead of providing you with the formula for the gradients, you will calculate them yourself using backpropagation.

Our network will have two fully connected layers (i.e. linear layers), and a softmax layer to perform classification. As in PS3, we'll train the network to minimize cross-entropy 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, for an image $\mathbf{x}$, we compute the unnormalized class probability (scores), $\mathbf{s} = (s_1, s_2, ..., s_C)$, as follows:

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

where C is the total number of classes and $\mathbf{W_i}, \mathbf{b_i}$ are the parameters of the fully connected layers.

The softmax loss, L_i, for a single image with label y, can be calculated as,

$$L_i(\mathbf{s}, y) = -\log \frac{e^{s_y}}{\sum_{j=1}^{C} e^{s_j}} \tag{3}$$

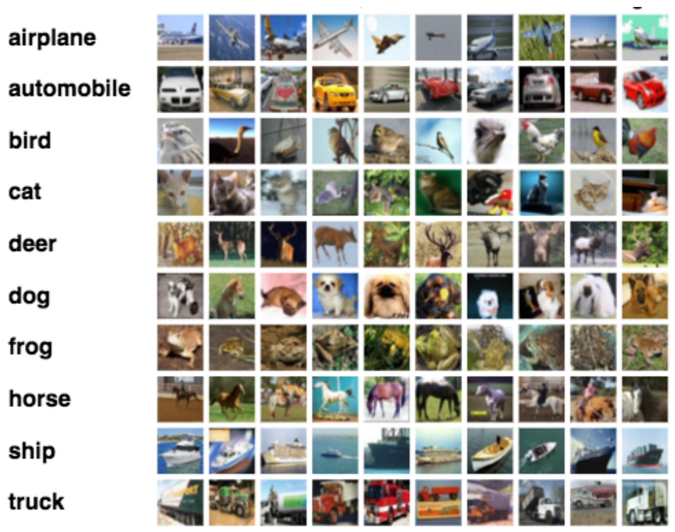


Figure 1: The CIFAR-10 dataset, which we'll be classifying in this problem. [1]

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 and ReLU layers. **(2 points)**

Layer reference:

- Fully connected layer:

$$\mathbf{y} = \mathbf{W}\mathbf{x} + \mathbf{b} \tag{4}$$

- ReLU:

$$y = \begin{cases} x, & x \geq 0 \\ 0, & \text{otherwise} \end{cases} \tag{5}$$

(b) Finish implementing the SoftmaxClassifier class. **(1 point)**

(c) Set the model hyperparameters (learning_rate, lr_decay, batch_size) and run the given code. If your method has been implemented correctly, you should obtain at least 45% accuracy on the test set. **(1 point)**

(d) Improve your optimization gradient descent method by adding *momentum*. Implement SGD_Momentum function and set up the remaining model hyperparameters same as what you used for (c), then run the given code. **(1 point)**

Hint: SGD with Momentum has the following update rule.

$$\boxed{\begin{aligned} &v \leftarrow dW + \beta * v \\ &W \leftarrow W - \text{learning\_rate} * v \end{aligned}}$$

where β is a scalar in range [0,1], dW is the gradient of the network parameter W, v is velocity matrix initialized as all zeros.

(e) Run the code provided and print out test accuracy in colab Notebook. **(0.5 point)**

(f) Plot and compare the training and validation accuracy, using 1) gradient descent alone, and 2) SGD_Momentum. Which model trains more quickly? Is the ultimate validation accuracy different? **(0.5 point)**

Reference

[1]Krizhevsky, A., Hinton, G. (2009). Learning multiple layers of features from tiny images.

Acknowledgement

Part of the homework and the starter code are taken from a previous EECS 442 class taught by David Fouhey, which itself was adapted from 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.