Lab 5: Deriving Gradients

Lab 5: Deriving Gradients 100/100 Points

Due: Tue Apr 18, 2023 8:30am4/18/2023

Offline Score:**100/100** Add Comment

100/100 Points

Details

Overview

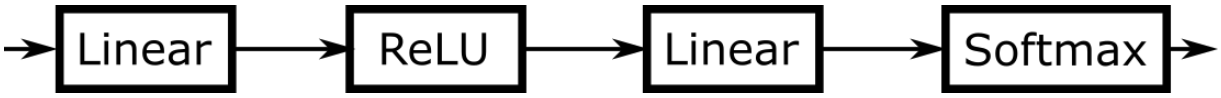
In this week's lab, you will derive the backpropagation equations for your from-scratch library.

Overview of From-Scratch Library

During the next few labs, you will implement all of the core parts of a deep learning algorithm yourself:

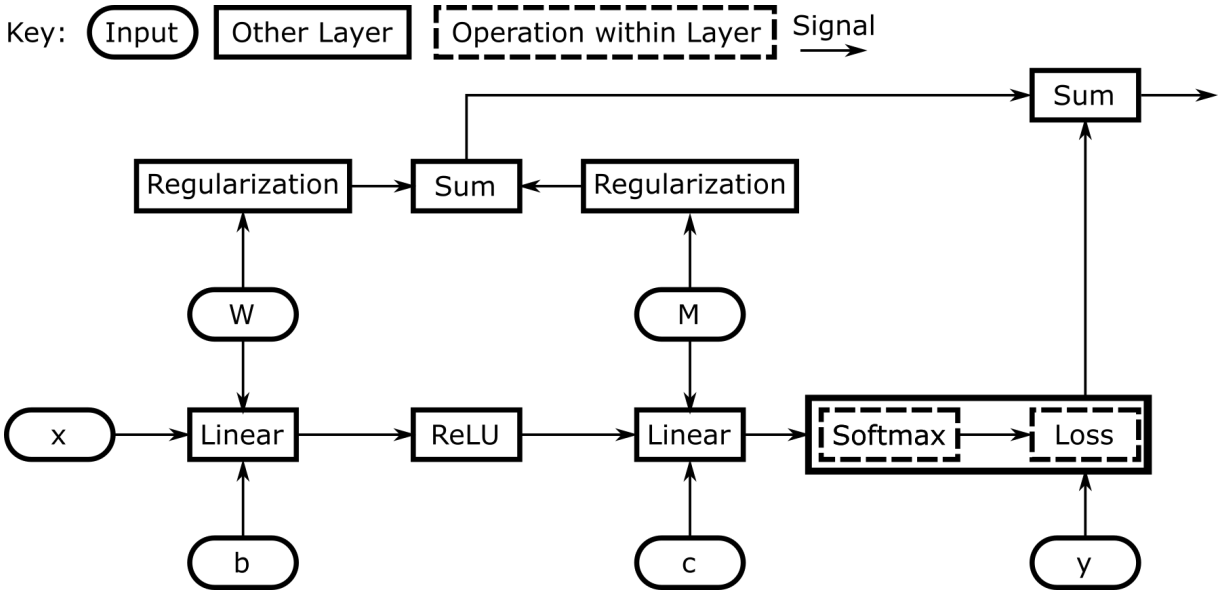
- Forward propagation
- Backward propagation
- Loss and regularization
- Stochastic gradient descent

Your library will support the following network structures:



Simple Network

It will be easy to add layers and even new layer types to your library and your library will support parallel paths in addition to simple sequential networks like the one shown here. In fact, we will use several new layers and parallel paths to implement weight decay:



Full Network

You will implement the following layers:

- Linear

- ReLU
- A single layer implementing both Softmax and Cross-Entropy Loss
- Regularization
- Sum

Your implementation will have some major differences from the PyTorch autodiff functionality:

- We will explicitly model the network as a single graph that can be used for both forward and backward computation. (`torch` models only the backward graph – the forward computations simply occur as they are executed in the code)
- The network will store the nodes in the computation graph in the order they will be executed in a "tape". Using such a "tape" (also known as a Wengert list) makes backpropagation easy: We simply go backward through the list when computing gradients. (`torch` instead uses a full graph structure, queuing backpropagation gradients for computation as soon as all their dependencies are met. This allows for more parallel computation, but also makes the implementation more complex.)
- Your implementation will be written entirely in PyTorch without using the autodiff library. Internally, both PyTorch's forward and backpropagation functionality are written in C++ to reduce overhead.

One of the major differences in the labs ahead is that you will implement the backpropagation code yourself. So your code will not need PyTorch's `requires_grad=True`, `backward()`, etc.

This week

In this week's lab you will develop the backpropagation equations for each layer used by the network, both in their elementwise form and in their vector form.

Details

Writing the backward-propagation equations

Write your derivation report **by hand** on blank, lined, grid, or "digital" paper. In the top-right corner, write your name, the date, the lab assignment number and name, and the page number.

Unlike your forward-propagation examples, focus on only one layer at a time. Assume the inputs and output gradient of this layer are known.

You should compute the backpropagation equations for each of these layers:

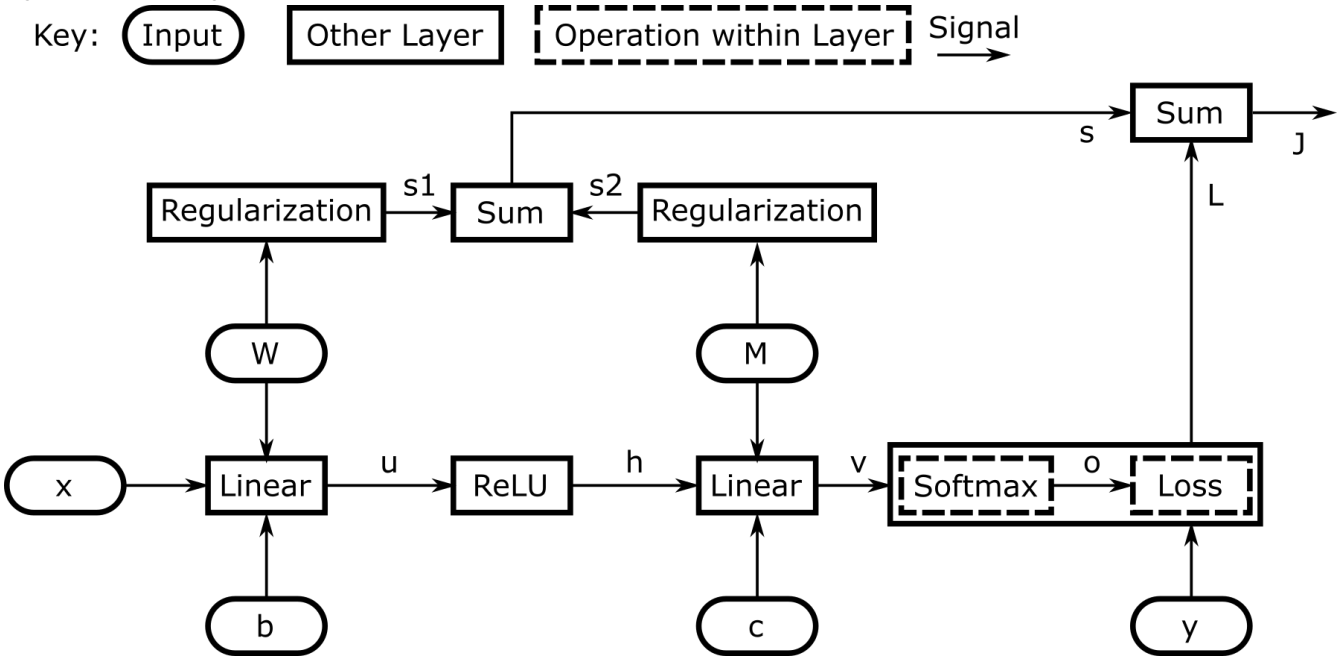
- The Linear layer with a bias
- The ReLU layer
- The Regularization layer uses the *squared* Frobenius norm. (In other words, you don't need to compute a square root when computing this.)
- The L2Loss layer uses the *squared* L2 norm (which is rather like the Frobenius norm, except for vectors, and incorporating the correct output). You may use the same layer for both L2 ad Frobenius if you generalize it.
- If the ReLU layer does not include a regularization constant, a constant multiplier layer
- The Sum layer which accepts two scalar inputs

You do NOT need to compute the backpropagation equations for the combined sofmax-cross-entrop layer: - The Softmax layer performs both the softmax operation as described in **Section 4.1.1.2 (https://www.d2l.ai/chapter_linear-classification/softmax-regression.html#the-softmax)** of our text and the cross-entropy loss described in **Section 4.1.2.1 (https://www.d2l.ai/chapter_linear-classification/softmax-regression.html#log-likelihood)**. You do NOT need to derive the backpropagation equations for this layer.

For this layer, simply use the backpropagation equation $\partial J/\partial \vec{\mathbf{v}} = (\partial J/\partial L)(\vec{\mathbf{o}} - \vec{\mathbf{y}})$ It propagates backwards for both sub-steps in a single step, as described in **Section 4.1.2.2 (https://www.d2l.ai/chapter_linear-classification/softmax-regression.html#softmax-and-cross-entropy-loss)**. Please see the template code for how we deal with this layer having two outputs.

In defining these layers, you don't need to match your equations exactly to what your instructors may have done. You may have different constant multipliers, for example.

Start from your equations for the network, using the variable names defined in the input boxes (rounded rectangles) and on the signal lines in this figure:



See our textbook, **Section 4.7 – Forward Propagation, Backward Propagation, and Computation Graphs (https://www.d2l.ai/chapter_multilayer-perceptrons/backprop.html)** for an example done on a simpler network.

For each layer, select one or two elements from each input and derive $\frac{\partial J}{\partial I_i}$ for that input.

Use the scalar chain rule to incorporate the output gradient that that input affects. Where an input affects multiple outputs, use the scalar multivariate chain rule.

Once you have derived the backpropagation equations for a single element, extrapolate your derivations to all of the elements of the input and write the entire input gradient in element form in terms of the output gradient's elements and the input's (and possibly output's) elements

Finally, write that layer's backpropagation equation in a vector form, using variables to describe the vector, matrix, or tensor inputs or outputs.

Your derivations do not need to include batches. However, your implemention in future weeks will support batches.

It is good to include crossed-out work on your page. Please neatly cross it out by lining it through at most twice or drawing a large X through a portion of the text you don't want to keep.

Final Deliverables

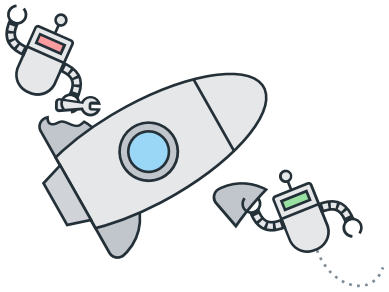
A complete submission consists of:

- Your pages of derivations, including:
  - Forward propagation (resubmit your Lab 3 tests)
  - Backward propagation

Previous Module Previous (https://msoe.instructure.com/courses/13814/modules/items/551278)

Next Module Next (https://msoe.instructure.com/courses/13814/modules/items/551946)

Sorry, Something Broke



Help us improve by telling us what happened

Report Issue