CS 1678/2078 Homework MLP Implementation

Abstract

In this assignment you will be implementing the backpropagation for a multi-layered perception and the rest of the training code necessary to train the network. To submit this assignment, upload a `.pdf` to Gradescope containing your responses to the questions below. You are required to use LaTeX for your write up. Upload a zip of your code to the Code portion of the assignment.

1 Coding up a neural network

For this homework you will be coding up a neural network with multiple hidden layers that use ReLU activation functions. There are two files in the supplied assignment code: `nn.py` and `hw_mlp.py`. The first file contains all the code necessary to create and neural network, loss function, and compute the partial derivatives for the weights. The second file contains all the code you will use to train the neural network on the MNIST data set. You should first complete all the TODOs in the `nn.py` file and then complete the TODOs in the `hw_mlp.py` file. After you should answer the questions below.

This implementation of the neural network breaks a layer into two components, the linear operation

$$z^i = h^{i-1}{W^i}^\top + b^i,$$

where $b^i \in \mathbb{R}^{n_i}$ are the bias terms for each output unit, and the second component is the activation function $h^i = \sigma(z^i)$. For the linear portion, there are two main functions you need to implement, `linear_forward(x,W,b)` and `linear_backward(dz,x,W,b)`. `linear_forward` should return z^i and `linear_backward` should return the partial derivatives for `x,W,b`, i.e., `linear_backward` returns

$$\frac{\partial l(\theta)}{\partial h^{i-1}} = \frac{\partial l(\theta)}{\partial z_i} W^i, \frac{\partial l(\theta)}{\partial W^i} = \frac{\partial l(\theta)}{\partial z^i}^\top h^{i-1}, \frac{\partial l(\theta)}{\partial b^i} = \sum_{j=1}^{m} \frac{\partial l(\theta)}{\partial z^i_{j,\cdot}}.$$

Note that in addition to z^i, `linear_forward` also returns a function to compute the partial derivatives for `x,w,b` given $\frac{\partial l(\theta)}{\partial z^i}$. Similar to `linear_forward`, `relu(x)` returns the output of the ReLU activation function and a function that computes the derivative of the loss function with respect to its input, i.e., `relu(`z^i`)` returns

$$\max(0, z^i), \frac{\partial l(\theta)}{\partial z^i} = \frac{\partial l(\theta)}{\partial h^i} \odot (z^i \geq 0).$$

These functions are all combined into the class `DenseLayer`, which serves as a wrapper for the parameters and choice of activation function (we are only considering ReLU in this homework though). In the DenseLayer class you need to initialize the parameters. b^i should be initialized to zeros, and each $W^i_{j,k}$ should be sampled from a normal distribution with mean 0 and a standard deviation of $\frac{1}{\sqrt{n_{i-1}}}$.

The `NN` class implements a MLP that is made up of `DenseLayers` with the output layer being a linear layer. We make the output a linear layer so that the loss function can compute efficient derivatives.

In this homework, we consider a classification task to classify handwritten digits into classes $0-9$. We will create the probabilities of each class with a softmax function on the outputs of the network. The loss function will be the average log probability of class label for each data point, i.e.,

$$l(\theta) = -\frac{1}{m} \sum_{i=1}^{m} \sum_{j=1}^{10} y_{i,j} \ln p_{i,j},$$

where $y_{i,j} = 1$ if the class label for the i^{th} data point is the j^{th} class and $p_{i,j}$ is the probability of the j^{th} class for the i^{th} data point. For numerical stability reasons most deep learning libraries do not compute $p_{i,j}$ and then do $\ln p_{i,j}$, they just compute $\ln p_{i,j}$ directly, i.e.,

$$\ln p_{i,j} = h^k_{i,j} - \ln \sum_{j'=1}^{n_k} e^{h^k_{i,j'}},$$

where $h^k_{i,j}$ is the j^{th} output of the neural network for the i^{th} data point. This is done so backprop can more efficiently compute the derivatives and have greater numerical stability. However, for this homework you will be computing the partial derivatives of the loss with respect to h^k manually, so you can exploit the efficient form of the derivative.

The partial derivative of the log probability for the softmax function for the network output is

$$\frac{\partial \ln p_{i,j}}{\partial h^k_{i,j'}} = \mathbf{1}_{j=j'} - p_{i,j'},$$

where $\mathbf{1}_A = 1$ if A is true and 0 otherwise. If y represents a one hot encoding of the class labels (as we assumed above), then we can write an efficient version of the derivative of the loss function as

$$\frac{\partial l(\theta)}{\partial h^k} = \frac{p - y}{m}.$$

You will need to implement the loss function and this derivative in your code.

The function `try_nn()` will construct a neural network compute the output, loss function, and partial derivatives of the network's parameters. This function can be useful to check for runtime errors. The function `test_cases()` will check your implementations for correctness. The derivatives will be compared to the derivatives obtained using finite differences. These test cases may not be all encompassing, but should get you most of the way there.

2 Training the NN

In the file `hw_mlp.py` you will need to implement code for the Adam optimizer, mini-batch sampling, and a training loop. The data for MNIST is saved in numpy `.npy` files. Implement these functions, choose your hyperparameters and successfully train the neural network. Then try and find hyperpameters to train the neural network to get as good of test accuracy as you can. Note that 99.79% accuracy is the best achieved so far, see `https://proceedings.mlr.press/v28/wan13.html`. You do not need to hit this number, but you should try and do your best. Keep track of what you are trying and how it impacts training.

The purpose of this activity is to build your intuition about how the hyperparameters impact training and the final test loss. Do not spend more than a few hours doing this. If one configuration of hyperparameters is taking too long to train, you can stop it. In the real-world those with the most compute are usually the ones able to find the best model.

Report below: 1) what you tried, 2) the best hyperparameters you found, and 3) the plot produced by the program.

Note in your submission PDF you do not need to include the text above (or the previous section). To do so delete this line and the ones preceding. Make sure to mark the appropriate pages in the PDF on gradescope.

1. What did you try to get the optimal performance?

2. What were the best hyperparameters you found?

Hyperparameter	Value
Batch Size	?
Hidden Sizes	[?,?]
η	?
β_1	?
β_2	?
ϵ	?
num epochs	?

3. Plot of best parameters.

loss_acc.pdf