Exercise II

AMTH/CPSC 663b - Spring semester 2021

Published: Friday, February 26, 2021
Due: Friday, March 12, 2021 - 11:59 PM

The current problem set requires a working installation of PyTorch (v1.4), torchvision (v0.5), and matplotlib (v3.1).

Compress your solutions into a single zip file titled `<lastname and initials>_assignment2.zip`, e.g. for a student named Tom Marvolo Riddle, `riddletm_assignment2.zip`. Include a single PDF titled `<lastname and initials>_assignment2.pdf` and any Python scripts specified. Any requested plots should be sufficiently labeled for full points.

Any formatting that allows the TAs to quickly determine which part of the problem the code is used for is fine. This means a Jupyter notebook with headings is allowed, as long as both the *.ipynb file and the PDF from Jupyter are submitted.

Programming assignments should use built-in functions in Python and PyTorch; In general, you may use the `scipy` stack [1]; however, exercises are designed to emphasize the nuances of machine learning and deep learning algorithms - if a function exists that trivially solves an entire problem, please consult with the TA before using it.

Problem 1

1. Provide a geometric interpretation of gradient descent in the one-dimensional case. (Adapted from the Nielsen book, chapter 1)

2. An extreme version of gradient descent is to use a mini-batch size of just 1. This procedure is known as online or incremental learning. In online learning, a neural network learns from just one training input at a time (just as human beings do). Name one advantage and one disadvantage of online learning compared to stochastic gradient descent with a mini-batch size of, say, 20. (Adapted from the Nielsen book, chapter 1)

Problem 2

1. Backpropagation with a single modified neuron (Nielsen book, chapter 2)
   Suppose we modify a single neuron in a feedforward network so that the output from the neuron is given by $f(\sum_j w_j x_j + b)$, where f is some function other than the sigmoid. How should we modify the backpropagation algorithm in this case?

2. Backpropagation with softmax and the log-likelihood cost (Nielsen book, chapter 3)
   To apply the backpropagation algorithm for a network containing sigmoid layers to a network with a softmax layer, we need to figure out an expression for the error $\delta_j^L = \partial C / \partial z_j^L$ in the final layer. Show that a suitable expression is: $\delta_j^L = a_j^L - y_j$

3. Backpropagation with linear neurons (Nielsen book, chapter 2)
   Suppose we replace the usual non-linear σ function (*sigmoid*) with $\sigma(z) = z$ throughout the network. Rewrite the backpropagation algorithm for this case.

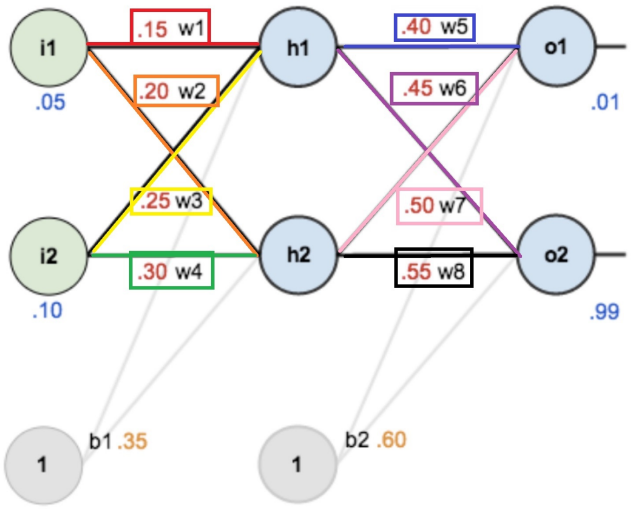

Figure 1: Simple neural network with initial weights and biases.

Problem 3

1. It can be difficult at first to remember the respective roles of the ys and the as for cross-entropy. It's easy to get confused about whether the right form is $-[ylna + (1-y)ln(1-a)]$ or $-[alny + (1-a)ln(1-y)]$. What happens to the second of these expressions when y=0 or 1? Does this problem afflict the first expression? Why or why not? (Nielsen book, chapter 3)

2. Show that the cross-entropy is still minimized when $\sigma(z) = y$ for all training inputs (i.e. even when $y \in (0, 1)$). When this is the case the cross-entropy has the value: $C = -\frac{1}{n}\sum_x [ylny + (1-y)ln(1-y)]$ (Nielsen book, chapter 3)

3. Given the network in Figure 1, calculate the derivatives of the cost with respect to the weights and the biases and the backpropagation error equations (i.e. δ^l for each layer l) for the first iteration using the cross-entropy cost function. Please use sigmoid activation function on h1, h2, o1, and o2. Initial weights are colored in red, initial biases are colored in orange, the training inputs and desired outputs are in blue. This problem aims to optimize the weights and biases through backpropagation to make the network output the desired results.

Problem 4

1. Download the python template `prob4.py` and read through the code which implements a neural network with PyTorch based on MNIST data. Within the provided python file is a basic scaffold of a model training workflow. You may use the existing functions in the script for both `4.1` and `4.2`. Compare the squared loss and cross entropy loss. To do this,

   - Finish the provided script to train the model with each of the above loss functions.
   - Create a plot of the training accuracy vs epoch for each loss function (2 lines, 1 plot)
   - Create a plot of the test accuracy vs epoch for each loss function (2 lines, 1 plot)

   Which loss function converges fastest? Which achieves the highest test accuracy? Provide some rational as to the observed differences.

2. Using the same set-up from `prob 4.1`, let's now add regularization to the previous network. For the following experiments, you may use the best performing loss function from `prob 4.1`. Generalization gap below refers to the (train accuracy - test accuracy).

   - Implement L1 regularization and train the model using $\lambda \in \{0.001, 0.005\}$. Create a plot of the train accuracy, test accuracy, and generalization gap vs epoch for each λ (3 plots, 2 lines each).
   - Implement L2 regularization and train the model using $\lambda \in \{0.001, 0.01, 0.1\}$. Create a plot of the train accuracy, test accuracy, and generalization gap vs epoch for each λ (3 plots, 3 lines each).
   - Apply dropout to both hidden layers and train the model using $p \in \{0.05, 0.1, 0.5\}$. **Hint:** To implement dropout, you can use a special type of PyTorch layer included in torch.nn [2]. Create a plot of the train accuracy, test accuracy, and generalization gap vs epoch for each p value (3 plots, 3 lines each)
   - Using the loss data you've collected so far, create a plot of the test accuracy vs epoch for each of the experiments performed for `prob 4.2`. (8 lines, 1 plot)

   Are the final results sensitive to each parameter? Is there any regularization method which performs best?

Bonus

1. Where does the softmax name come from? (Nielsen book, chapter 3)

Optional

1. Alternate presentation of the equations of backpropagation (Nielsen book, chapter 2)
   Show that $\delta^L = \nabla_a C \odot \sigma'(z^L)$ can be written as $\delta^L = \sum'(z^L)\nabla_a C$, where $\Sigma'(z^L)$ is a square matrix whose diagonal entries are the values $\sigma'(z^L_j)$ and whose off-diagonal entries are zero.
2. Show that $\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l)$ can be rewritten as $\delta^l = \Sigma'(z^l)(w^{l+1})^T \delta^{l+1}$.
3. By combining the results from 1 and 2, show that
   $\delta^l = \Sigma'(z^l)(w^{l+1})^T \dots \Sigma'(z^{L-1})(w^L)^T \Sigma'(z^L)\nabla_a C$.

References

[1] "The scipy stack specification¶." [Online]. Available: https://www.scipy.org/stackspec.html

[2] "Pytorch nn module docs¶." [Online]. Available: https://pytorch.org/docs/stable/nn.html