Exercise II

AMTH/CPSC 663b - Spring semester 2018

Published: Thursday, February 15, 2018
Due: Thursday, March 1, 2018 - 4:00 PM

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. Your homework should be submitted to Canvas before Thursday, March 1, 2017 at 4:00 PM.

Programming assignments should use built-in functions in Python and TensorFlow; 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)

3. Create a network that classifies the MNIST data set using only 2 layers: the input layer (784 neurons) and the output layer (10 neurons). Train the network using stochastic gradient descent. What accuracy do you achieve? You can adapt the code from the Nielson book, but make sure you understand each step to build up the network. Please save your code as `prob1.py`. (Adapted from the Nielsen book, chapter 1)

Problem 2

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 $\sigma'(z_j^L)$ 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 problems 2.1 and 2.2, show that $\delta^l = \Sigma'(z^l)(w^{l+1})^T \dots \Sigma'(z^{L-1})(w^L)^T\Sigma'(z^L)\nabla_a C$.

4. 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.

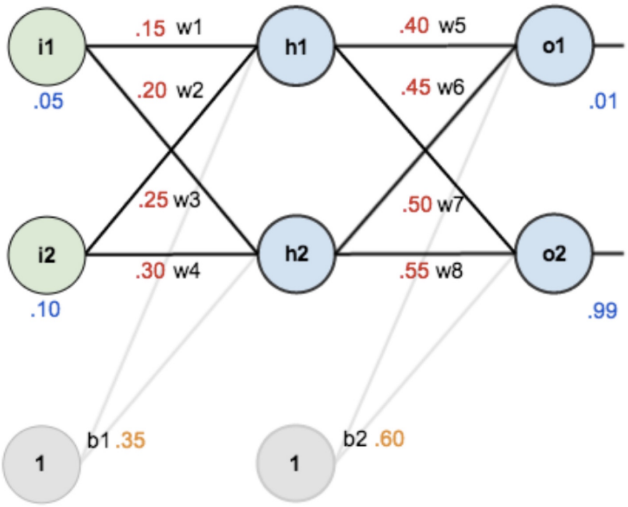

Figure 1: Simple neural network with initial weights and biases.

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. 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. More specifically, given inputs 0.05 and 0.10, the neural network is supposed to output 0.01 and 0.99 after many iterations.

Problem 4

1. Download the python template `prob4_1.py` and read through the code which implements a neural network with TensorFlow based on MNIST data. Implement the TODO part to define the loss and optimizer. Compare the squared loss, cross entropy loss, and softmax with log-likelihood. Plot the

training cost and the test accuracy vs epoch for each loss function (in two separate plots). Which loss function converges fastest?

2. Based on `prob 4.1` add regularization to the previous network. Implement L2 and L1 regularizations separately, and dropout separately. Compare the accuracy and report the final regularization parameters you used (for dropout, report the probability parameter). Are the final results sensitive to each parameter? Please save your code as `prob4_2.py`. You may want to check out the following link for regularization.
https://www.tensorflow.org/versions/r0.12/api_docs/python/contrib.layers/regularizers

Bonus

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

Optional

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: $\delta_j^L = a_j^L - y_j$

References

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