Exercise III

AMTH/CPSC 663b - Spring semester 2018

Published: Thursday, March 8, 2018
Due: Thursday, March 29, 2018 - 4:00 PM

Compress your solutions into a single zip file titled `<lastname and initials>_ assignment3.zip`, e.g. for a student named Tom Marvolo Riddle, `riddletm_assignment3.zip`. Include a single PDF titled `<lastname and initials>_assignment3.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 29, 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 TensorFlow Practice (5 pts)

It is suggested that you go through the TensorFlow Get Started if you haven't done so.
Check out with this link https://www.tensorflow.org/get_started/

- "Getting Started with TensorFlow"
- "Getting Started for ML Beginners"

Check out with this link https://www.tensorflow.org/versions/r1.1/get_started/

- "MNIST for ML Beginners"
- "Deep MNIST for Experts"

Optionally, you may want to read through on the same page
"TensorBoard: Visualizing Learning" and "TensorBoard: Graph Visualization"
Also, you may find it helpful to read through the sample codes.

1. In `tflecture1_2.py`, make sure you understand what `tf.Variable, tf.Session()` do. Explain what `init = tf.global_variables_initializer() and init.run()` do, and how this code differs from `tflecture1_1.py`.

2. Compare `tflecture1_3.py` with `tflecture1_4.py`. What do line 32, line 78-87, line 94-102 in `tflecture1_3.py` do? What does line 57 in `tflecture1_4.py` do? Briefly Explain. Make sure you have installed `sklearn` before testing with the code.
   -Line 32 in `tflecture1_3.py`
   `theta = tf.matmul(tf.matmul(tf.matrix_inverse(tf.matmul(XT, X)),XT),y)`
   -Line 57 in `tflecture1_4.py`.
   `gradients = tf.gradients(mse,[theta])[0]`

3. Based on what `tflecture1_4.py` does, implement the `prob1.py` TO DO part using the TensorFlow optimizer. Is the output different from `tflecture1_4.py`? Save your code as `prob1.py`. Hint: You might only need two lines to set up the optimizer and use the optimizer to minimize the MSE.

Problem 2 (5 pts)

1. It's tempting to use gradient descent to try to learn good values for hyper-parameters such as λ and η. Can you think of an obstacle to using gradient descent to determine λ? Can you think of an obstacle to using gradient descent to determine η?

2. L2 regularization sometimes automatically gives us something similar to the new approach to weight initialization (i.e., we initialize the weights as Gaussian random variables with mean 0 and standard deviation $1/\sqrt{n_{in}}$ where n_{in} is the number inputs to a neuron). Suppose we are using the old approach to weight initialization (i.e., we initialize the weights as Gaussian random variables with mean 0 and standard deviation 1). Sketch a heuristic argument that:

   (a) Supposing λ is not too small, the first epochs of training will be dominated almost entirely by weight decay.

   (b) Provided $\eta\lambda \ll n$ the weights will decay by a factor of $\exp(-\eta\lambda/m)$ per epoch.

   (c) Supposing λ is not too large, the weight decay will tail off when the weights are down to a size around $1/\sqrt{n}$, where n is the total number of weights in the network.

3. Modify your code for problem 1.3 to use an adaptive learning rate using AdagradOptimizer in TensorFlow. Record your output and save your code as `prob2.py`. Comment on how you chose any of the parameters.

Problem 3 (5 pts)

1. When discussing the vanishing gradient problem, we made use of the fact that $|\sigma'(z)| < 1/4$. Suppose we use a different activation function, one whose derivative could be much larger. Would that help us avoid the unstable gradient problem? (Nielsen book, chapter 5)

2. Consider the product $|w\sigma'(wa+b)|$ where σ is the sigmoid function. Suppose $|w\sigma'(wa+b)| \geq 1$.

   (a) Argue that this can only ever occur if $|w| \geq 4$.

   (b) Supposing that $|w| \geq 4$, consider the set of input activations a for which $|w\sigma'(wa+b)| \geq 1$. Show that the set of a satisfying that constraint can range over an interval no greater in width than

   $$\frac{2}{|w|} \ln\left(\frac{|w|(1+\sqrt{1-4/|w|})}{2} - 1\right).$$

   (c) Show numerically that the above expression bounding the width of the range is greatest at $|w| \approx 6.9$, where it takes a value ≈ 0.45. This demonstrates that even if everything lines up just perfectly, we still have a fairly narrow range of input activations which can avoid the vanishing gradient problem.

3. **Identity neuron:** Consider a neuron with a single input, x, a corresponding weight w_1, a bias b, and a weight w_2 on the output. Show that by choosing the weights and bias appropriately, we can ensure $w_2\sigma(w_1 x + b) \approx x$ for $x \in [0,1]$, where σ is the sigmoid function. Such a neuron can thus be used as a kind of identity neuron, that is, a neuron whose output is the same (up to rescaling by a weight factor) as its input.

   Hint: It helps to rewrite $x = 1/2 + \Delta$, to assume w_1 is small, and to use a Taylor series expansion in $w_1\Delta$

4. Recall the momentum-based gradient descent method we discussed in class where the parameter μ controls the amount of friction in the system. What would go wrong if we used $\mu > 1$? What would go wrong if we used $\mu < 0$?

5. Modify your code for problem 2.3 to implement momentum using MomentumOptimizer and save your code as `prob3_a.py`. Then modify the code to implement momentum with an adaptive learning rate by using the AdamOptimizer, and save your code as `prob3_b.py`. Record your outputs for both cases, and compare to your results from problems 1.3 and 2.3. Comment on how you chose any of the tuning parameters.

Problem 4 (5 pts)

1. **AlexNet** Download the AlexNet weights (large file), as well as the associated jpeg images and caffe_classes.py from [2]. Then download the AlexNet.py script provided on canvas. Ensure that it runs correctly. In what follows, attach your code for parts 2 and 3.

2. **Reading out from a layer** Write code that runs a Tensorflow session on one of the sample images to extract the output of the first convolutional layer. What is its output shape? In this layer, readout the output of one of the 96 57×57 arrays. Plot this using matplotlib's imshow or a similar matrix-to-image function for a couple of sample images and include them in your writeup.

3. Write code that extracts the output of the final layer just before the softmax. What is its output shape for a couple of sample images? Include a couple of these images in your writeup.

4. **Backpropagation in a convolutional network:** In class, we've discussed the four fundamental equations of backpropagation in a network with fully-connected layers. Suppose we have a network containing a convolutional layer, a max-pooling layer, and a fully-connected output layer. How are the equations of backpropagation modified?

5. Design and train a CNN for MNIST. Save your code as `prob4.py`. Record your result and give a description on how you design the network and briefly on how you make those choices. (e.g., numbers of layers, initialization strategies, parameter tuning, adaptive learning rate or not, momentum or not, etc.)

   You may want to modify the network based on the following link, if you don't know where to start with. https://www.tensorflow.org/tutorials/layers

References

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

[2] M. Guerzhoy and D. Frossard, "Alexnet implementation + weights in tensorflow." [Online]. Available: http://www.cs.toronto.edu/~guerzhoy/tf_alexnet/