Homework IV

STAT 5685/6685 - Fall semester 2023

Due: Friday, November 4, 2023 - 5:00 PM

Please upload all relevant files & solutions into Canvas. Include your answers (minus code) in a single PDF titled `lastname_firstname_HW4.pdf`, and all your code in a single jupyter notebook titled `lastname_firstname_HW4Code.ipynb`. Any requested plots should be sufficiently labeled for full points.

Unless otherwise stated, programming assignments should use built-in functions in Python, Tensorflow, and PyTorch. In general, you may use the `scipy` stack; 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.

Stater code is provided for all homeworks at `https://github.com/KevinMoonLab/STAT-6685`. You are not required to use this code, although it is highly recommended. In some cases you will be asked to describe what certain parts of the code are doing.

Problem 1 - (6685-20pts) - (5685-10pts)

1. (3/4 pts) 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. (3/4 pts) When discussing the vanishing gradient problem, we made use of the fact that $|\sigma'(z)| < 1/4$ for the sigmoid activation function. 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)

3. (**6685**-12pts) Consider the product $|w\sigma'(wa + b)|$ where σ is the sigmoid function. Suppose $|w\sigma'(wa + b)| \geq 1$.

   (a) (2 pts) Argue that this can only ever occur if $|w| \geq 4$.

(b) (7 pts) 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) (3 pts) Show numerically (e.g., you could make a plot) 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.

4. (2 pts) 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 $\mu < 0$?

Problem 2 - (6685 - 20 pts) - (5685 - 25 pts)

1. Briefly compare and contrast the HPO methods of grid search, random search, successive halving, and hyperband. What are the advantages and disadvantages of each of these methods?

2. Briefly describe how Bayesian optimization, genetic algorithms, and particle swarm optimization can be used to do HPO. What are the advantages and disadvantages of each of these methods?

3. Ray Tune is a popular HPO framework. Read through the Ray Tune key concepts webpage located at `https://docs.ray.io/en/latest/tune/key-concepts.html#tune-60-seconds`. Briefly describe the purpose and function of the following objects within the Ray Tune framework:

   (a) Ray Tune Trainable

   (b) Tune Search Space

   (c) Tune Trials

   (d) Tune Search Algorithms

   (e) Tune Schedulers

   (f) Tune ResultGrid

Problem 3 - 10 points (6685 only)

In class, we've discussed the four fundamental equations of backpropagation in a network with fully-connected layers, and they are given by:

$$\delta_j^L = \frac{\partial C}{\partial a_j^L}\frac{\partial a_j^L}{\partial z_j^L} = \frac{\partial C}{\partial a_j^L}\sigma'(z_j^L) \tag{1}$$

$$\delta_j^l = \frac{\partial C}{\partial z_j^l} = \sum_k \frac{\partial C}{\partial z_k^{l+1}}\frac{\partial z_k^{l+1}}{\partial a_j^l}\frac{\partial a_j^l}{\partial z_j^l} = \sum_k \delta_k^{l+1} w_{kj}^{l+1}\sigma'(z_j^l) \tag{2}$$

$$\frac{\partial C}{\partial b_j^l} = \frac{\partial C}{\partial z_j^l}\frac{\partial z_j^l}{\partial b_j^l} = \delta_j^l \tag{3}$$

$$\frac{\partial C}{\partial w_{jk}^l} = \frac{\partial C}{\partial z_j^l}\frac{\partial z_j^l}{\partial w_{jk}^l} = \delta_j^l a_k^{l-1} \tag{4}$$

Suppose we have a network containing a convolutional layer with an activation function:

$$z_{j,k}^1 = b^1 + \sum_l \sum_m w_{l,m}^1 a_{j+l,k+m}^0$$
$$a_{j,k}^1 = \sigma(z_{j,k}^1),$$

a max-pooling layer:

$$a_{j,k}^2 = \max(a_{2j,2k}^1, a_{2j,2k+1}^1, a_{2j+1,2k}^1, a_{2j+1,2k+1}^1),$$

and a fully-connected output layer with an activation function:

$$z_i^3 = b_i^3 + \sum_{j,k} w_{i;j,k}^3 a_{j,k}^2$$
$$a_i^3 = \sigma(z_i^3).$$

How are the equations of backpropagation modified for this neural network? *You need to show your work for all four backprop equations.*

Problem 4 - (6685-30pts) - (5685-45pts)

1. (10/15 pts) Design and train a CNN with at least two convolutional layers, each followed by a max-pooling layer, for the MNIST dataset. Use dropout and L2 regularization on the weights when training the network. Use standard stochastic gradient descent in this part of the problem. Use grid search to tune hyper-parameters. Include your code. Record your final test accuracy and give a description on how you designed the network and briefly on how you made your design choices (e.g., numbers of layers, initialization strategies, parameter tuning, adaptive learning rate or not, momentum or not, etc.). To get full credit, you will need to get an accuracy of at least 98%.

   See `https://github.com/KevinMoonLab/STAT-6685/blob/master/HW-4/CNN.ipynb` for starter code.

2. (10/15 pts) Starting with the network you designed in the previous problem, replace L2 regularization with L1 regularization and tune the regularization parameter as well as the learning rate. Explain what the `get_loss` method is doing in the `RegularizedNet` class. Use two initialization strategies: 1) initialize with the weights obtained using L2 regularization and 2) initialize randomly. Which initialization strategy worked the best? Based on your results, which regularization worked best on this data?

3. (10/15 pts) Start with the best network you have obtained so far. Keeping the same hyperparameters of this network, you will now apply several different optimizers to retrain the network.

   (a) Use the following approaches to optimization: 1) standard SGD; 2) SGD with momentum; 3) AdaGrad; 4) Adam. Tune any of the associated parameters including the global learning rate using the validation accuracy. Report the final parameters selected in each case, and the final test accuracy in each case. Provide two plots with the results from all four approaches: 1) the training cost vs the number of epochs and 2) the validation accuracy vs the number of epochs. Which optimization approach seems to be working the best and why?

   (b) Pick one of the optimization approaches above. Using the same network, apply batch normalization to each of the hidden layers and retune the (global) learning rate using the validation accuracy. Report the new learning rate and the final test accuracy. Does batch normalization seem to help in this case?

Problem 5 - 20 points (5685 only)

AlexNet Download the AlexNet weights as described below and model and download the test images at `https://github.com/KevinMoonLab/STAT-6685/blob/master/HW-4`.
Starter Code is at `https://github.com/KevinMoonLab/STAT-6685/blob/master/HW-4/Alexnet.ipynb`.
In what follows, attach your code for parts 2 and 3.

If you're using Tensorflow, you can find them at [1]. You will need to download both bvlc_alexnet.npy and caffe_classes.py. Then download the AlexNetTensorflow.py script provided on canvas. Ensure that it runs correctly.

If you're using PyTorch, you can find the code at https://pytorch.org/docs/stable/torchvision/models.html.

Note: Detailed sample code is provided in GitHub for each of the tasks specified in the following prompts. What is expected of you is to play with the code and provide detailed commentary of what the code is doing (you may also write your own code that achieves the same tasks as long as you thoroughly respond to the questions below). Since extensive sample code has been provided, it is expected that your final responses will thoroughly explore the concepts covered in the class and also explain how to handle different kinds of real world situations.

1. **Reading out from a layer (5 pts):** Write code that extracts the output of the first convolutional layer for a given test image. In this layer, readout the output of all of the 64 55×55 arrays (i.e. these are the outputs from the different filters or feature maps in this layer). You could do this by plotting all 64 images in a 8×8 grid similar to the plots shown in lecture about Gabor filters. Plot these images for a single test image using matplotlib's imshow or a similar matrix-to-image function and include them in your writeup. What output shapes do you get? *On top of the code provided, try different things such as image transformations, cropping, image blurring, etc. and each time see what the first convolutional layer is doing. Include a few examples of these transformations* ***in the main pdf.***

2. (5 pts) Now you will visualize some filters as we saw in class. In a similar way as in the previous exercise, write code that plots the 64 filters of size 11×11 applied to the first channel. In the starting code you can see how to plot one of the filters, just extend the code to plot a grid that outputs all the 64 filters. Comment on how the 64 images you get in part (1) are related to these 64 filter images.

3. (10 pts) Try feeding in another image of your choice (not one of the test images) and reading out the top 5 probabilities of classification. Does the classification seem correct? Image inputs should be of the form 227×227 pixels and 3 channels (use .png to be safe). You may need to crop or zero pad your image. *We are looking for some creative exploration here. What you should observe is that Alexnet is pretty robust to noise. Record your experiments and observations in detail in your responses that convey a clear understanding of the subject matter.*

Problem 6 - 20 points (6685 only)

Now we will perform transfer learning using torchvision pretrained models. For the following problem you will have to load ResNet18 which has been pretrained on the ImageNet dataset. In this problem, we will use the pretrained model on the CIFAR10 dataset.

1. (5 pts) Load the CIFAR10 dataset and display one image of each class. Set up your dataloader using *torchvision.tranforms*. Take into account the required preprocessing transformations expected by the model (ResNet18). Additionally, include a data augmentation strategy and report it.

2. (5 pts) Download an image of any of the classes present in the ImageNet dataset and make sure the pretrained model is able to classify it correctly. Then, try a transformation or noise injection that can trick the network.

3. (5 pts) Now we will retrain the model for CIFAR10. Freeze all of the layers except for the last fully connected layer. Allow this last layer to adapt to the CIFAR10 dataset during training. Create a validation set and implement an early stopping strategy. Plot the validation accuracy vs epochs and report your selection of hyper-parameters.

4. (5 pts) Finally, retrain the network in a similar way as in the previous part. In this case, unfreeze the first convolutional layer and the first convolutional layer present in the first "*BasicBlock*", in addition to the final fully connected layer. Plot and compare the filters of the first convolutional

layer for the original model with the new filters learned after you retrain it. Again use early stopping on your validation set and plot the validation accuracy vs epochs. Report how you selected your hyperparameters.

For an explanation on how to load ResNet18 and CIFAR10 you can watch the tutorial video on convolutional networks on Canvas. Also, code is available at: `https://github.com/KevinMoonLab/STAT-6685/blob/master/PyTorchDemo/pytorch_demo_Conv.ipynb`

References

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