Assignment 3 Part 2: Convolutional Neural Networks for Gesture Recognition

Deadline: Thursday, October 18, 2018 at 11:59pm

This assignment must be done individually. This assignment is out of **100 points**. You can find the mark breakdown in each section. You will be marked based on the correctness of your implementation, your plots, and your answers to the required questions in each section.

Learning Objectives

In this assignment, you will learn how to design, implement, train, validate and optimize a convolutional neural network whose purpose is to predict gestures recorded from a smartphone. In this assignment you will:

1. Gather gestures using a provided smart phone app (which was already completed in Part 1)
2. Preprocess the gesture data for classification
3. Reason about an appropriate *convolutional* neural network architecture for gesture detection and train said network.

You are expected to apply what you have learned in lab 2; the assignment has less guidance with essentially no startup code.

What To Submit

You should hand in the following files:

- A PDF file `assignment3-writeup.pdf` containing your visualizations (plots) and answers to the written questions in this assignment. **Graded questions are located in each section**, indicated by the heading *Questions*. **Please answer these questions and include them in your report** under the appropriate section headings.
- Your code for parts 2, 3 and 4, which should make use of the bare-bones structure given in the starter package `assign3.zip` provided with this assignment.
- Your saved model `model.pt`, it will be used to evaluate your best model against the test dataset.

You should package all of your files into a single zip file - `assign3-StudentID.zip` - where StudentID is your utorid. Upload your submission to the course Quercus page for this assignment.

1 Gesture Recognition - Problem Definition

1.1 Introduction

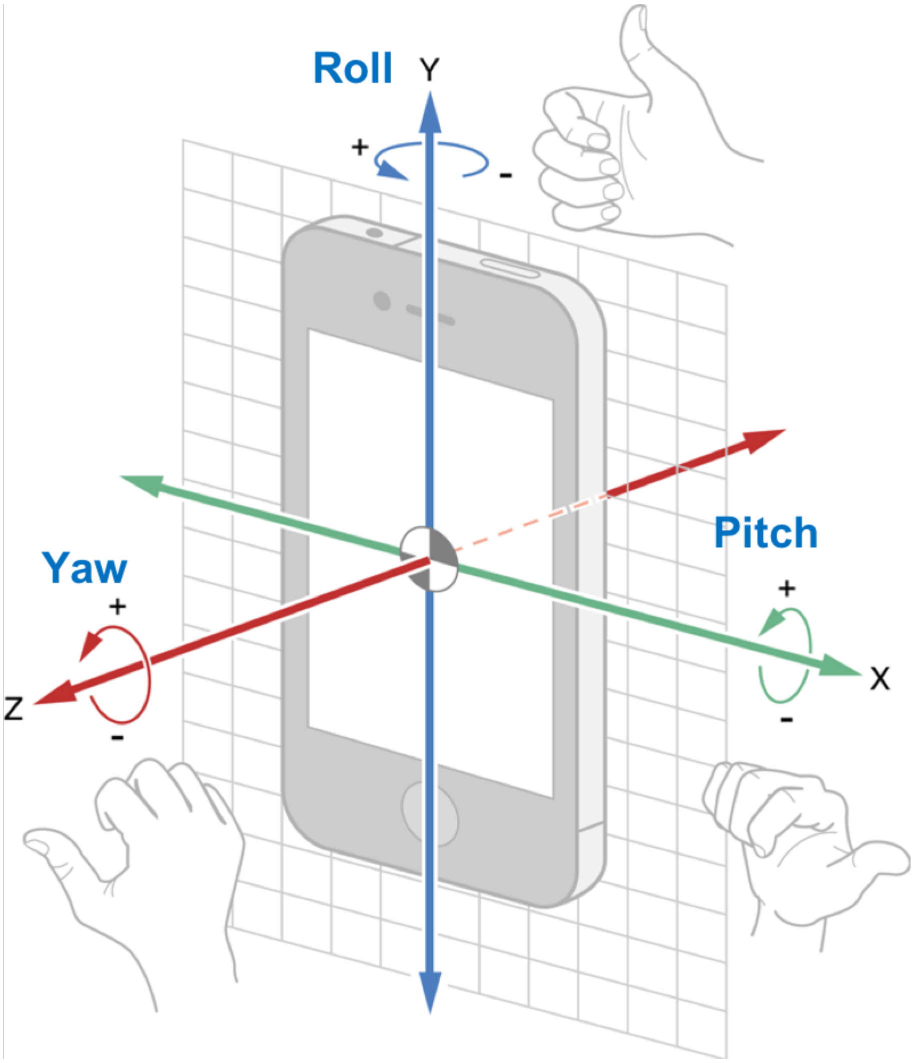

Figure 1: Translational and rotational coordinates of a smartphone. For those who know a little about airplanes or quadrocopters, this is the same coordinate system used there: x, y, z translation and pitch (green), roll (blue), yaw (red). Image courtesy of: `https://stackoverflow.com/questions/28857165/rotation-on-y-axis-in-degree-or-radians-in-ios`

Recall the work you did on Friday September 28th in the course Tutorial, for Assignment 3 Part 1: you moved the smartphone in a set of 26 gestures meant to represent the 26 letters of the alphabet. The application that you were using made use of the *accelerometer* and *gyroscope* to record the acceleration of the phone (in three dimensions) and the rate of rotation (also in three dimensions) of the phone while you were moving it. Everyone in the class did the same thing, “recording” those 26 gestures five times each. The goal of this assignment is to use that data to train a convolutional neural network to recognize (classify) those 26 gestures, and to obtain the highest accuracy possible.

Figure 1 illustrates the axes of measurement that are used in the phone. Acceleration is measured along the x, y, z axes, and is given in metres/s^2. Each measurement will be referred to as a_x, a_y, a_z The rate of rotation is measured in three dimensions as well, in units of radians/second. The axes used are traditionally called pitch, yaw and roll, which are illustrated in Figure 1. Each rate of rotation measure will be referred to as ω_x (pitch), ω_y (roll) and ω_z (yaw).

1.2 Accelerometer and Gyroscope Samples: Time Series Data

The way that the full motion of the gestures you made was recorded, was a sequence of acceleration and rotation measurements *over time*, which is called *time series data*. Go into the 'data' folder given in the zip file for this assignment and one sub-folder,and open up one of the `a_1.csv` files using a spreadsheet program so that you can look at it. You will see that each gesture consists of roughly 100 rows of data with seven numbers in each row. Each row is a *sample* from the accelerometer and gyroscope at a specific time during the gesture motion. The first column gives the number of milliseconds after the start button was pushed. The next three columns are the sampled values of a_x, a_y, a_z at that time, and the final three columns are ω_x, ω_y and ω_z.

Important Definitions: We refer to each `_.csv` file as a *gesture instance*. An individual row in that file, with three values of acceleration and three values of rotation, will be referred to as a *sample*.

Recall that, when you made each gesture, you were given two seconds to complete the motions. During that time, the App was configured to extract sample measurements of acceleration and rotation every 20 milliseconds, or 50 times per second. Thus there are a total of 100 samples (and so 100 rows in the each `_.csv` file) for each gesture instance. We can refer to the 50 Hz as the sampling frequency, f, and the total per gesture instance as, $T = 2$ seconds, and the total number of samples in each gesture instance, $L = f \times T = 100$.

Each student in the class has provided $26 \times 5 = 130$ gesture instances, and each person's 130 instances is provided in a separate folder, where each folder is numbered. Each folder contains the same 130 file names: `a_1.csv`, `b_1.csv`, ... `z_1.csv`, `a_2.csv`,..., `z_5.csv`. You will need to put these into a dataset that will need to be split, as usual, into training and validation sets, as was done in Assignment 2. Each file contains one gesture instance where the name of the file gives the correct label of the gesture instance. A single 'input' to the neural network should be the time series sequence of the 6-channel samples of gesture instance, beginning with sample 0 and ending with sample 99, as follows:

$$\{(a_x^0, a_y^0, a_z^0, \omega_x^0, \omega_y^0, \omega_z^0), (a_x^1, a_y^1, a_z^1, \omega_x^1, \omega_y^1, \omega_z^1), \ldots (a_x^{99}, a_y^{99}, a_z^{99}, \omega_x^{99}, \omega_y^{99}, \omega_z^{99})\}$$

where the superscript represents the sample number in the gesture instance.

Given the sequence of sample values as above, the goal of the neural network is to predict the class of the gesture, which is one of the 26 letters `a`, `b`, ... `z`.

2 Data Pre-processing (33 Points)

The data collected from the whole class is provided to you in sub-folders of the `data` folder given with this assignment, as described above. Each gesture instance contains raw sensor values from the smart phones. **The columns come in this order:** time from start (in milliseconds), x-acceleration, y-acceleration, z-acceleration, pitch, roll, and yaw. The values include the effects of gravity and sources of noise. We have held out some of the data as a hidden test set. The data held out is all the gesture instances from a subset of the class and a few other people. Once you've submitted your completed lab, your models will be evaluated against this set.

2.1 Converting the dataset to numpy files (7 Points)

The first step is to convert all the data `.csv` files in all the sub-folders and put it into a form that can be worked with as numpy arrays. Write that code in the file `csv2numpy.py`; it should create two separate numpy arrays: the sensor values for each gesture instance, and the label for each instance. (Note that the labels are given in the file names). Once this code works (and you should test it carefully), you should save the data in numpy form as `.npy` files (recall you did this in Assignment 0) - one file should be called `instances.npy` and the other should be `labels.npy`, and should be stored in the `data` folder.

2.2 Understanding the dataset (9 Points)

It is always import to begin visualizing the dataset and trying to understand it before building a model to classify it. The cartoon below lampoons the unthinking nature of model building in the deep learning era, versus the carefully reasoned machine learning of prior approaches.

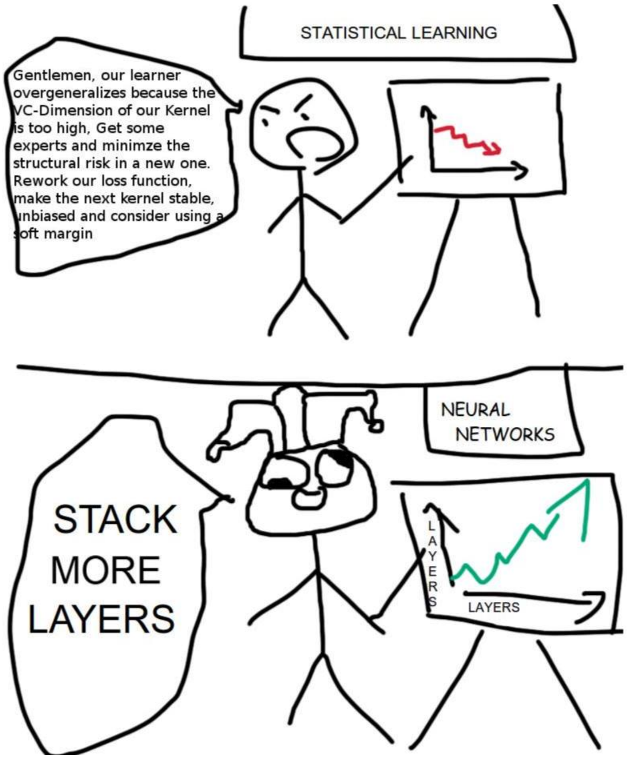
STATISTICAL LEARNING
Gentlemen, our learner overgeneralizes because the VC-Dimension of our Kernel is too high, Get some experts and minimze the structural risk in a new one. Rework our loss function, make the next kernel stable, unbiased and consider using a soft margin
NEURAL NETWORKS
STACK MORE LAYERS
LAYERS
LAYERS

Figure 2: Dank meme. Image courtesy of: `https://www.reddit.com/r/ProgrammerHumor/comments/5si1f0/machine_learning_approaches/`

The criticism of 'unthinking' work in deep learning has arisen because optimization of models is often seen as a "black art" because to get it to work, we often have to tune a set of hyperparameters without systematic understanding of what is being changed. This is unlike “shallow” machine learning where there are usually clearer mathematical motivations for model choices. We would like make a compromise between these two approaches: While we have to accept hyperparameter tuning as a way of life, we can also try to understand our data to guide our hyperparameter search with intelligence rather than chance.

1. (4 points) First, visualize some of the data creating code in the file `visualize_data.py`. Choose two gestures classes (i.e two of a, b, c etc.). For each of those, choose 3 gesture instances (from among the input data, from different people) and plot each of sensor values (a_x, a_y, a_z, ω_x, ω_y, and ω_z) versus sensor value number on a graph. Use six different colors and make sure to include a legend. You should have 6 graphs in total, three for each gesture. Each plot should have 6 curves corresponding to x-acceleration, y-acceleration, z-acceleration, pitch, roll and yaw. **Include these plots in your final report.** Group the graphs by gesture so that three graphs of the same gesture are together and can be visually compared. What patterns distinguish one gesture from the other? Note: we’re using 3 plots per gesture to make sure any qualitative conclusions we make are (more) reliable than if we just used a single graph per gesture.

   Questions

1. (2 points) Can you spot any patterns that distinguish one gesture from the other? Describe them (try to talk about at least 2 curves). Feel free to use informal language like “gesture 1 has 3 bumps on its x-acceleration curve but gesture 2 only has 2” or “both gestures have 2 bumps for the y-acceleration but the bumps occur ealier for gesture 2 than gesture 1”.
2. (1 points) Do your observations make sense with the patterns you had to trace on your smartphone for each gesture? Why or why not? (the answer can be “no” as long as you provide some plausible reasons).
3. (2 points) Visualizations like these are often useful for helping decide the architecture of the neural network. The concept of the *receptive field* of a CNN arises from this question. The receptive field of a kernel (on any layer of the CNN) is the size of the *original input region* that the output from the kernel is affected by. The receptive field of a kernel on the first layer is simply the size of the kernel itself. Kernels on subsequent layers indirectly compute on larger regions of the raw input because of things like the pooling layer. What do our graphs suggest about the receptive field that our CNN needs to have in order to classify the gestures?

2.3 Basic statistics (8 points)

Next let’s look at some basic statistics of our dataset.

1. (4 points) In `bin_data.py` write a function to find the *average* sensor values over **both time and gesture instances** for each gesture. For example, if you have m instances of a gesture, you should start with a $m \times 100 \times 6$ numpy array for that gesture. After the averaging, you should have a 1×6 numpy array of average sensor values for the gesture. In total, you should have C, 1×6 numpy arrays, where C is the number of gestures. Repeat this calculation for the *standard deviation* (over **both time and instances**). You should now have C, 1×6 numpy arrays for standard deviation. Choose 3 gestures. For each gesture, plot the average

features in a bar graph. **Plot the standard deviation as error bars**. **Include these plots in your final report.** We now have a very coarse way of comparing 3 gestures: by their average sensor values and standard deviation. Can you differentiate the 3 gestures just be looking at these bar graphs?

You may have noticed that some bars in your plot are always larger than others. For example, the largest bar on your plot is probably the third one. This corresponds to z-acceleration and it is large because of the force due to ever-present gravity.

Questions

1. (2 points) Looking at your bar graphs, can you classify the gestures by eye? Explain your answer.
2. (2 points) Do you think a neural network could classify the gestures just from the average sensor values rather the original samples? Why or why not?

2.4 Normalizing the dataset (5 Points)

As discussed in class, data values are often normalized to prevent a pre-bias in the training process towards features that start with higher values. Although normalizing data may seem as straightforward as subtracting the mean and dividing by the standard deviation, as we did in Assignment 2, it gets more complicated with time series data: we need to decide between two types of normalization: normalization over time steps, and normalization over time steps *and* gesture instances.

Our first option is to normalize the data over time only. For example we can calculate the mean and standard deviation of x-acceleration for each gesture instance, i out of the total of N in the dataset (recall that μ is used to indicate mean, and σ is used to indicate variance):

$$\mu^i_{a_x} = \sum_{l=1}^{100} a^i_{x,l}$$

$$\sigma^i_{a_x} = \sum_{l=1}^{100} (a^i_{x,l} - \mu^i_{a_x})^2$$

Here a_x is the x-acceleration and $a_{x,l}$ refers to the l^{th} (out of L) value in the time series. The sum is over the $L = 100$ time steps. Let's call these the *local* mean and standard deviation, local because they are calculated from each sample without considering the other samples. We subtract each sample by its local mean and divide by the local standard deviation.

Our second option is to normalize the data over both samples and time. For example we can calculate the mean and standard deviation of the x-acceleration as,

$$\mu_{a_x} = \sum_{i=1}^{N} \sum_{l=1}^{100} a^i_{x,l}$$

$$\sigma_{a_x} = \sum_{i=1}^{N} \sum_{l=1}^{100} (a^i_{x,l} - \mu_{a_x})^2$$

The difference is we're also summing over the samples. Let's call these the *global* average and standard deviation, global because we sum over all the samples. We subtract each sample by the global mean, which is the same for all samples, and divide by the global standard deviation.

Let's examine the advantages and disadvantages of both methods. First consider global normalization. The gesture classes (a vs. b vs. c etc.) may differ in some systematic way, for example the accelerations may always be larger for the B gesture than the O gesture, maybe because you need to draw B's faster. Global normalization can exploit this because it maintains the relative intensities of different gestures. Local normalization will not be able to exploit systematic differences between gestures which might make it harder for the model to learn. However, local normalization should give us better generalization. If you draw a gesture differently across samples, then local normalization will help deal with that.

There is always a trade off between normalizing data one way or another. We would like to exploit differences between gestures but also receive the generalization benefits of local normalization. Unfortunately we can't have it both ways. For gesture recognition, we care a lot about generalization. We want our recognition system to be robust to different users. Therefore, we will use local normalization.

1. (5 points) Complete `normalize_data.py` to perform local normalization on the dataset. Save the normalized data in `./data/normalized_data.npy`.

2.5 Train-validation split (4 Points)

Machine learning datasets are usually split into 3 sets: train, validation, and test. As mentioned above, part of the dataset is being held out to form the (hidden) set that will be used to grade your final result.

Questions

1. (2 points) Why is it necessary to evaluate a model on a test set instead of only relying on the validation set?

While you don't have access to what you'll eventually be evaluated on, it is still a good idea to tune your network to be as good as possible on the validation set. As discussed in class and in Assignment 2, when deciding on the train-validation split, there's an inherent trade off between the amount of data you can train on, and the reliability of the validation. In Assignment 2, we chose a 80-20 training-validation split. In that dataset, there were roughly 24,000 'instances' of each Adult in the balanced set. In this dataset there are (roughly) 40 people who generated 130 instances each.

1. (2 points) Please choose an appropriate data split for this assignment and split your data into a `train_data.npy` and `val_data.npy`. Note that you can come back to this section and change the train to validation ratio later on if you feel that a different setting would work better. In your report, explain how you arrived at your final choice of split.

3 Training (38 Points)

You will be given marks in this section for correctly implementing the Dataset class, a working model (one which does better than guessing at random), and the training and evaluation loops.

We will evaluate the performance of your model on a held out test set after you've submitted your assignment (refer to section 4).

3.1 PyTorch Dataset (7 Points)

In `dataset.py` write a PyTorch Dataset class to serve data during training. Instantiate Dataset and DataLoader objects for both the train and validation sets in `main.py`.

3.2 Model design (7 points)

In `model.py` implement a convolutional neural network to classify gestures. There are no restrictions on your architecture other than it must be a CNN. Note: you earn points in this section by having a **working** implementation (defined as better than random guessing), *it doesn't have to be a good implementation*. Therefore, it is advisable that you train something simple first to secure points for this section. In general it's always a good idea in deep learning to start with a simple model to make sure your training pipeline is error-free before moving onto more sophisticated models.

You will have to consider the batch size, learning rate, loss function, optimizer, etc.

3.3 Training (7 Points)

In `main.py` write a training loop to iterate through the training dataset and train your model. It is recommended that you include some useful logging in the loop to help you keep track of progress, and help in debugging.

3.4 Evaluation (7 Points)

In `main.py` write an evaluation loop to iterate through the validation dataset to evaluate your model. It is recommended that you call the evaluation function in the training loop (perhaps every epoch or two) to make sure your model isn't overfitting. Keep in mind if you call the evaluation function too often, it will slow down training.

3.5 Plotting (6 Points)

Include a plot of the training and validation accuracy of your model as a function of training steps. **Include these plots in your report**. Report the best validation accuracy the model achieves. Apply smoothing as needed.

3.6 Saving and loading your model

Validation performance will oscillate or begin to decrease (due to overfitting) during training so we should keep track of the best model. To do this, keep track of the lowest validation error seen so far and everytime the current model beats the lowest error so far, save the model using `torch.save(model, 'filename.pt')`. You can load a saved model using `model = torch.load('filename.pt')`. Alternatively, you could save the model at every epoch and go back to find the best model in hindsight when you plot validation error against number of epochs. This approach can rapidly use up space in the file storage system on your computer.

4 Test performance (25 Points)

Your model will ultimately be tested on a set of people whose gestures you have not been given. Your goal is to improve the accuracy of your results, as best you can, such that you will have the best chance of achieving high accuracy on that unseen test set. This may require some thinking about how you divide up the given data among among possible training, validation and test sets.

4.1 Absolute Accuracy (15 Points)

A total 15% of your grade on this assignment is based on your model's accuracy on the test dataset as shown in the figure below.

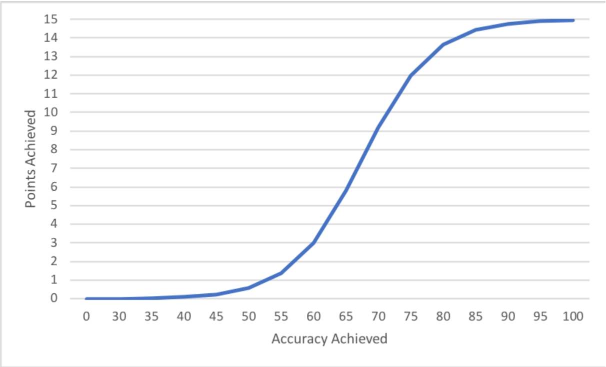

Figure 3: Grade for Given Accuracy

The purpose of striving for a high accuracy is to give you practice thinking about the correct architecture for the data as well as try some tricks deep learning practitioners use to improve training. However, you **are not allowed to collect more training or validation data**; You can, if you wish, use data augmentation methods.

4.2 Explanation/Source of Accuracy (10 Points)

Describe how you achieved the accuracy that you did - provide a clear description of each technique you used, and what it individually achieved. More points will be awarded for more attempts to improve.

4.3 Best Accuracy Prize

In addition to the grading for the grade for the test result, we will have prize for the best accuracy achieved.

4.4 Submission instructions

We will need the following to grade your model:

1. A ready-to-run model: `model.pt`. You should save the entire model (including architecture) rather than just the weights.
2. A list of hyperparameters, settings you chose, and anything else you need to train the model (e.g. if you used data augmentation then either the extra data or your script to generate the extra data).

3. Instructions for training your model (so we can potentially replicate your results).

Include your very best model in your submission for this assignment. Please try saving, loading, and evaluating your model on the validation set before handing it in! We will not debug your model if it's not working after the submission.

4.5 Some Recommendations

Below we provide a number of tricks and methods, that might improve your results. They are taken from general knowledge of neural networks, and are specific to this assignment, so it is entirely possible they will not be successful! The suggestions are listed roughly in order of difficulty of implementation. As always, you may need to experiment with the number of epochs to train for, batch size and learning rate.

- Your model's performance may vary between different runs. This is a problem because the validation performance may not reflect test performance. To get a more reliable evaluation, run the same model using different random seeds and take the model with the best average performance.
- Different activation functions.
- Optimizer using Stochastic Gradient Descent (SGD) with momentum.
- Adam optimizer, or some other variant of SGD (`http://ruder.io/optimizing-gradient-descent/`).
- Use weight decay.
- Use Dropout.
- Batch normalization (`https://arxiv.org/abs/1502.03167`).
- Change number of CNN layers, size of kernels, stride, etc.
- Try convolution layers without activation functions.
- Dilated convolutions (`https://www.inference.vc/dilated-convolutions-and-kronecker-factorisation/`).
- Change the size of your training set relative to the validation set (be careful to not make the validation set too small as it won't reflect the performance on the test set!).
- Implement cross validation. k-fold cross validation is usually not used in deep learning because it makes training slower. You need to train your model k times instead of one. However, if your model trains quickly, cross validation is a good idea to improve your model's performance..
- Remove some of the features like z accelerations, as perhaps the additional features are causing overfitting.
- Integrate the x-y acceleration data into position data. Use position instead of, or in addition to, acceleration in the neural network. (finding position from acceleration is actually non trivial, you may need to look into Kalman filters `https://www.wikiwand.com/en/Kalman_filter`).

5 Feedback (2 Points)

1. Describe your experience with assignment 3:

   (a) How much time did you spend on assignment 3?

   (b) What did you find challenging?

   (c) What did you enjoy?

   (d) What did you find confusing?

   (e) What was helpful?