Deep Queue-Learning: A Quest to Optimize Office Hours

Avoy Datta **Dian Ang Yap** **Zheng Yan**

ad9697 **dianang7** **yzh**

Note: We are planning to use the dataset we collect in this project in CS221 as well, with the same group members and same big-picture goal. We currently plan to tailor most of the "prediction" portion of our project towards 229, and all of the "optimization' portion towards 221.

Motivation

Among CS students at Stanford, the experience of queueing at office hours is practically universal. Unfortunately, the wait time is prone to high variance, resulting in students sometimes waiting 4-5 hours before receiving help. Not only are these instances stressful for students, they are stressful for TAs as well. Therefore, minimizing the wait times and optimizing queue lengths could benefit all parties. While these instances are not completely unpreventable, we strongly believe that they could be reduced — using the information that is gathered every day on Queuestatus. What if we had better tools to predict when students go to office hours, and how long each student is expected to take? If we had more processed information, then a lot of doors open for informed recommendations. Our ultimate, big-picture goal is to still be able to suggest optimal office hour assignments for a course, given some number of TA constraints. The project is divided into two components: prediction and optimization. Here, we give an overview of our current intentions with each component, and their relationship to each other. Features, models, and general progress is detailed in the document below.

Methodology

Overview:

For the prediction component, we will address the following: given information about a time slot and a course offering, what is the expected "amount of help requested" at office hours for this course within this time slot? We define the expected "amount of help requested" as the average serve time per person for a given day times the number of signups for that time slot. For the sake of brevity, we refer to this as the **load influx**. Intuitively, this is the total time needed to help all of the students that queued up within the timeslot. Through predicting load influx, we obtain approximations of how much workload is being added into a course's office hours at any given time, informing optimal TA schedules accordingly.

For the optimization component, we will utilize the predictions for a given week and provide an optimal scheduling of TAs, given course constraints. Our current proposed set of constraints is also specified below.

Feature collection:

We have set up a complete pipeline for collecting features off of Queuestatus, Carta, and individual course syllabus. At the moment, we have a complete dataset of all CS107 office hours from January 2017 until November 2018 (~5 quarters), and are working on collecting data for other courses as well. Our current pipeline automatically processes and calculates most of the features we use to predict load influx, given a course name, quarter/year, and queuestatus page.

We currently store the following data for each one-hour period where office hours is active (either somebody was served or someone signed up). Bolded items are currently used as features to predict load influx:

1. **# Sign ups**, # Serves, # Servers, average wait time in the hour, average serve time in the hour
2. **Week number (1-11), Weekday (0-6), and time of day (1-24)**
3. **Days before/after the next assignment/exam is due (-1 if no assignments/exams are coming up)**
4. **Indicator variable if this is the first OH timeslot being held within the last 3 hours**

We store the following data for each quarterly offering of a course, taken from Carta:

1. **Instructor rating**
2. **Number of students enrolled**

We store the following data for each separate course offered, taken from Carta (these features will be used once we collect data for more classes):

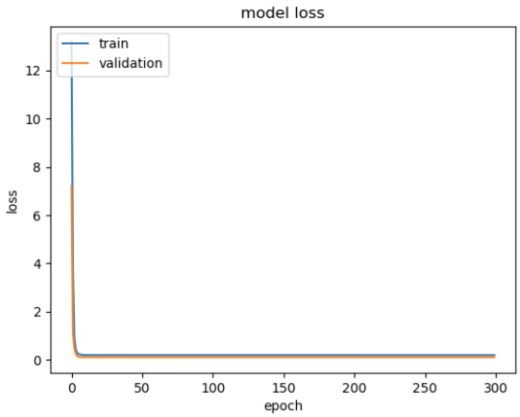
1. Course name (ex. CS107)
2. Avg. number of hours spent per week on class
3. Proportion of class that is freshmen, graduate students, and PhD students

Preliminary experiments

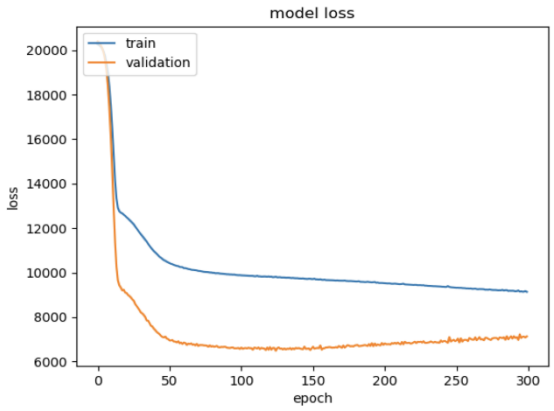
Predicting Hourly Load Influx with Neural Networks

We use features without any prior knowledge on the number of sign-ups, and without any prior knowledge on length of office hours-waiting time. Here, the features used are the same ones discussed earlier, indicative of the number of sign-ups for a given time slot. For evaluation purposes, we evaluate the mean squared error between the predicted value from the model as compared to the true value.

We initially randomly split the data into 85:5:10 in train/dev/test split, followed by normalizing the inputs between 0 and 1. We have an initial model with 3 hidden dense layers initialized randomly with He initialization, followed by a ReLU activation. In the final layer, we have a linear activation with one neuron, giving a total of 384 parameters in our fully-connected network. We experimented with both hinge loss and MSE loss while keeping other parameters ceteris paribus, and obtained the outputs below:



Hinge Loss. MSE on test set: 21521



MSE Loss. MSE on test set: 8971

After experimenting, we chose to optimize for MSE loss. In our experiments, we also see that normalizing the inputs help reduce MSE between predicted and test set, while allowing smoother gradient descent towards the optimum.

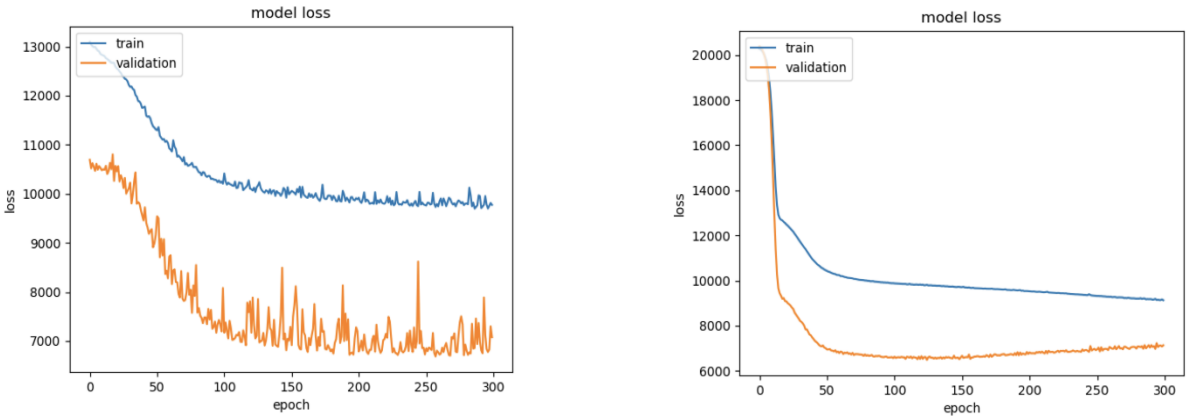


Figure 1: Loss without normalized inputs vs loss with normalized inputs

We see that there are signs of overfitting as the validation loss starts increasing after 150 epochs. Therefore, we use early stopping to prevent overfitting in models in which we ran for 300 epochs with a batch size of 64 with Adam optimizer. Overall, we have a good baseline for a fully connected network in predicting load influx. (with MSE of 8971, that gives an approximate error of 90 minutes estimate error for load influx).

Predicting Hourly Load Influx Category with a Random Forest Classifier

```
               Type of random forest: classification
                     Number of trees: 3000
No. of variables tried at each split: 2

        OOB estimate of  error rate: 33.5%
Confusion matrix:
     High Low class.error
High  449 255   0.3622159
Low   268 589   0.3127188
```

Using the same features as above, we try to predict the hourly load influx as a categorical variable using a random forest classifier (randomForest package in R) as a baseline model. We defined a new variable LoadInfluxCategorical for each entry, which can take on one of two values:: high load influx (> 1 hour load influx) and low load influx (<= 1 hour load influx), which gives approximately ~700 entries in each bucket (random forests are generally better when training on data with categories of equal size). We found that the classifier had an OOB (cross-validation) error of 33.5% when trained on this new variable, which is fairly bad.

Since we have such a large error on even binary classification, load influx as a quantitative variable on an hourly basis may be inherently too variable to predict fruitfully using our current features and a non-recurrent model, and to improve our predictions from above, we may need to set different goals or utilize better features.

Next steps

Scheduling office hours as a Constraint Satisfaction Problem

We have trained a deep neural network to predict load influx for office hours during specific times in the quarter. Given this model, we now strive to recommend office hour schedules to a group of TAs for a given class, based on expected load influx of students during different days in the quarter. We model this as a constraint satisfaction problem (CSP), with the TAs for the quarter as variables with time slots as values to assign.

When assigning schedules, we divide the quarter into 2-week intervals, and assign individual schedules for every biweekly interval. This is done to account for classes that have assignment deadlines in intervals of 10 days or 2 weeks as well as classes that have weekly assignments. The CSP is thus solved once for each 2-week interval.
When looking at a 2-week interval, we treat each individual hour between 8 am and 10 am (the assumed bounds for office hours) as a time slot. We then look at the time slots on which TAs are available (based on their listed weekly availabilities). The domain for each variable is all the possible combinations of hours during which the TA is available.

Consider a quarter with N TAs, with each TA (indexed *i*)having a list of time slot availabilities T_i. We define *minHours* and *maxHours* as the minimum and maximum number of hours the TA is allowed to work during the 2-week period, respectively. We propose the following factors for a set of variables $\{X_i\}_{i=1}^{N}$:

- Unary factor on each X_i: constrains -
    - sum(X_i) <= maxHours &&
    - sum(X_i) >= minHours
- *N*-ary factor over all variables: constraints-
    - presence of at least one time slot on every week day

These are our proposed constraints for the problem. We plan to initially construct the CSP with loose weights just to make sure the CSP returns complete assignments, and then add more constraints if need be.
We have thus far obtained a list of assignments for the two-week period. We define a **weight function** for each assignment such that:

$$W(\{X_i\}) = \overrightarrow{T_{assigned}} \cdot \overrightarrow{T_{expected}}$$

where:

- $\overrightarrow{T_{expected}}$ is a **unit** vector of time slots for the two-week period, with each element representing the scaled **predicted** load influx for that slot
- $\overrightarrow{T_{assigned}}$ is the vector of timeslots representing the number of TAs occupying each timeslot.

The weights assigned are proportional to the number of high-demand time slots filled by the complete assignments. The algorithm returns the assignment with the **highest** weight, which is the optimal schedule for the provided TA *availability, conditioned* on the *anticipated* load influx for that interval. We repeat this process for each two-week period till the end to obtain a complete schedule for the quarter.

Contributions

Avoy - CSP design and implementations; LSTM-model for predictions
Dian Ang - FC neural network and LSTM-model design and implementations
Zheng - Data collection/parsing (e.g.) queuestatus scraping, feature calculation. Random forest implementation.