

Deep Queue-Learning: A Quest to Optimize Office Hours

CS221 Progress Report

Avoy Datta
ad9697

Dian Ang Yap
dianang7

Zheng Yan
yzh

Note: We are planning to use the dataset we collect in this project in CS229 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, but may be slightly changed based on upcoming lectures on neural networks. We want to ensure that we’re putting material more suited for CS221 into the “CS221 part” of our project, and same with CS229.

Goals:

As unchanged from our project proposal, 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.

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? Current, 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 additional 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. Currently, we are still discussing the best way to formulate this constraint satisfaction problem (ex. for how many weeks at a time should TAs have the same weekly schedule for, if there is a minimum at all?), and how the load influx predictions will be used to prioritize each timeslot. 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 the next assignment is due (-1 if no assignments are coming up)**

4. **Days after previous assignment was due (-1 if no previous assignment)**
5. **Days before the next exam (-1 if no exams are coming up)**
6. **Indicator variable if this is the first OH timeslot being held within the last 3 hours**

For the above information, we plan to experiment for predictions of different-sized timeslots (load influx per 2 hours, 4 hours, etc.), as predictions for one hour only may be too variable.

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
4. Proportion of class that is graduate students
5. Proportion of class that is PhD students

As a concrete example of how our classifier works so far: given Monday of Week 7, at 5PM, a day before the assignment is due (also 7 days after the last assignment was due), 17 days before the exam, and the first OH held within the last 3 hours, in a class with an instructor with a 4.0 rating and 300 students enrolled, the classifier may predict that there will be an influx of 160 minutes*students (avg. time it takes to serve a student * number of students) within the next hour.

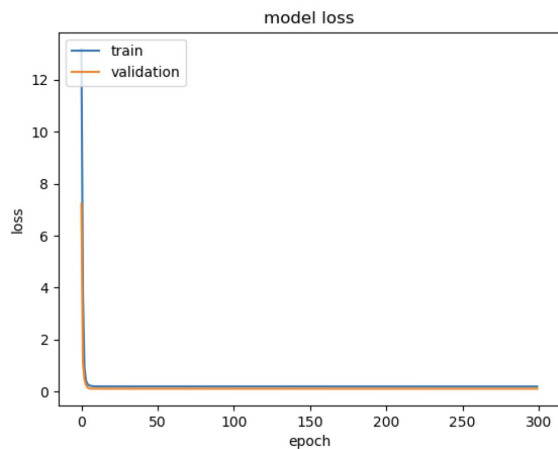
Predicting Load Influx with Neural Networks

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 extracted that could be indicative include number of TA as servers, day of week, week of quarter, days until assignment due/after assignment due, days till exam, number of students, instructor rating, average time spent on class, and proportion of students of different years. (frosh, grad students etc). For evaluation purposes, we evaluate the mean squared error between the predicted value from the model as compared to the true value.

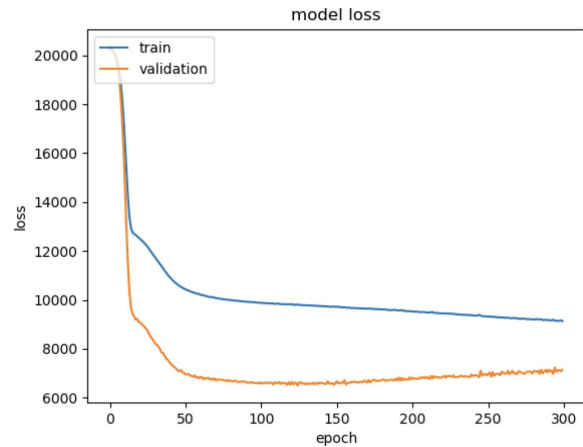
We initially randomly split the data into 85:5:10 in train/dev/test split, followed by normalizing the inputs between 0 and 1. We have an initial model with 3 hidden dense layers initialized randomly with He initialization, followed by a ReLU activation. In the final layer, we have a linear activation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

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 15)	225
dense_1 (Dense)	(None, 8)	128
dense_2 (Dense)	(None, 3)	27
dense_3 (Dense)	(None, 1)	4
Total params: 384		
Trainable params: 384		
Non-trainable params: 0		
Train on 1333 samples, validate on 71 samples		

We experimented with both hinge loss and MSE loss while keeping other parameters *ceteris paribus*, we have the outputs below:

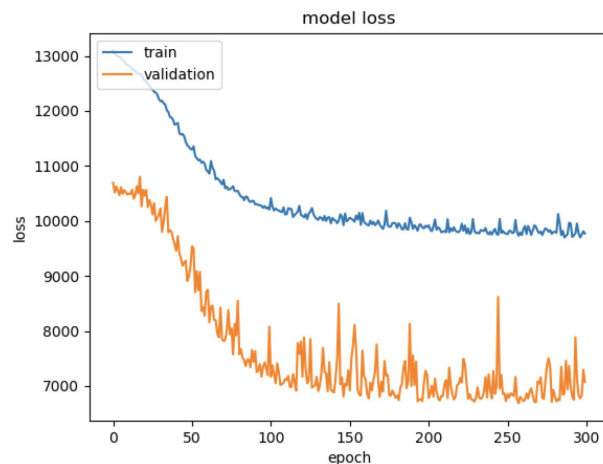


Hinge Loss. MSE on test set: 21521

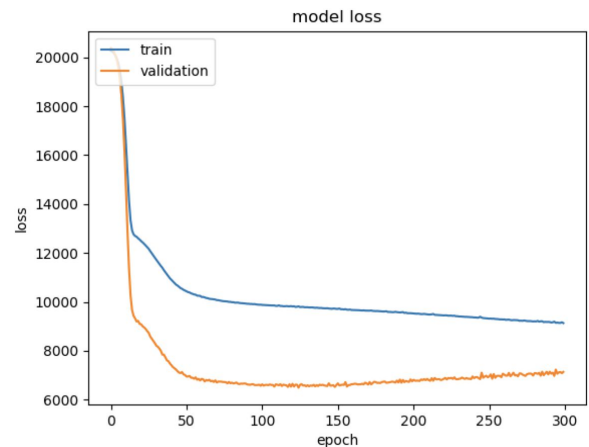


MSE Loss. MSE on test set: 8971

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.



Without normalizing inputs, train on MSE Loss.



Normalized inputs, evaluated on MSE loss.

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). In CS229, we plan to use LSTMs and improved NN models to improve prediction (hence not the focus on this project in CS221).

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 -
 - $\text{sum}(X_i) \leq \text{maxHours}$ &&
 - $\text{sum}(X_i) \geq \text{minHours}$
- 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.