# Deep Queue Learning: A Quest to Optimize Office Hours

**Avoy Datta**
Department of EE
Stanford University
`avoy.datta@stanford.edu`

**Yap Dian Ang**
Department of EE
Stanford University
`dayap@stanford.edu`

**Zheng Yan**
Department of CS
Stanford University
`yzh@stanford.edu`

## Abstract

Office hours at Stanford, especially among high enrollment classes, are typically subject to significant variance in student demand and causes stress for both students and TAs during time of high loads. We see this as a resource allocation problem, and we formulate a two-step approach to better allocate office hours. The first step is prediction, where we use data scraped from Queuestatus, Carta, and course syllabi to predict student demand at any office hours on an hourly basis. We experimented with different architectures such as Fully Connected Networks, Univariate and Multivariate LSTMs, and seq2seq models. The next step uses our prediction to generate recommendations for office hours assignments with modified Gibbs Sampling. Overall, our model is a promising source of information to augment existing office hours scheduling policies, both for new and existing courses. We report our model predictions and evaluation across different models, along with our suggestions for office hours assignments.

## 1 Task Definition

Among CS students at Stanford, the experience of queueing at office hours (OHs) is practically universal. Office hours is a highly valuable part of almost any class, allowing students to get valuable, one-on-one help from instructors. Unfortunately, student demand is prone to high variance, resulting in sometimes students waiting hours before receiving help, or conversely, teaching assistants (TAs) waiting hours before any students arrive. In particular, periods of overcrowding are a source of stress for both students and TAs, and are among the most commonly cited sources of negative experience on Carta. Thus, improvements in OH scheduling could significantly improve overall course experience for all parties involved.

To better optimize allocation of resources, we aim to predict peaks hours where more TAs can be assigned to to meet the surge in demands. Using hourly office hours data scraped from Queuestatus, course information taken from Carta, and major dates from class syllabi, we experimented with various models that predicts the hourly load influx for a given course and quarter. We define the load influx as the average serve time for the day times the number of student signups. This is equivalent to the expected aggregate TA time needed to satisfy all student demand over a given time period. Given these predictions, we then built a scheduler that outputs a suggested optimal TA schedule based on realistic TA constraints. Furthermore, we constructed a pipeline that, given basic course information, generates expected OH hourly load influx and TA recommendations for the whole quarter within a minute.

We note that after training models to predict load influx on these datasets, we do not predict hourly student demand for TAs, as is ideal. Rather, we predict hourly student demand for TAs given that office hours with some number of TAs is held. We determined that the number of TAs at office hours is uncorrelated with time of day ($p = 0.06$, cor.test in R). Furthermore, TAs seem to be scheduled approximately uniformly random throughout active hours. Therefore, we assume that the status quo scheduling of office hours is frequent and unbiased enough such that real student demand is, on average, proportional to the student demand given office hours is held. Thus, predicting student demand given office hours is held still gives useful information about the true distribution of student demand over the period of a quarter.

For both prediction and scheduling, we evaluate our models using a combination of quantitative and qualitative metrics. For the prediction component, the most logical quantitative metric is simply the average RMSE. Intuitively, this is the average amount of time, measured in minutes, that the model is off by when predicting student demand. Qualitatively, the most useful information that we could extract to inform OH schedules is the relative times in which student demand is high. In other words, we would prefer a model that has high RMSE but accurately predicts relative spikes in student demand, rather than a model has low RMSE but fails to capture important trends.

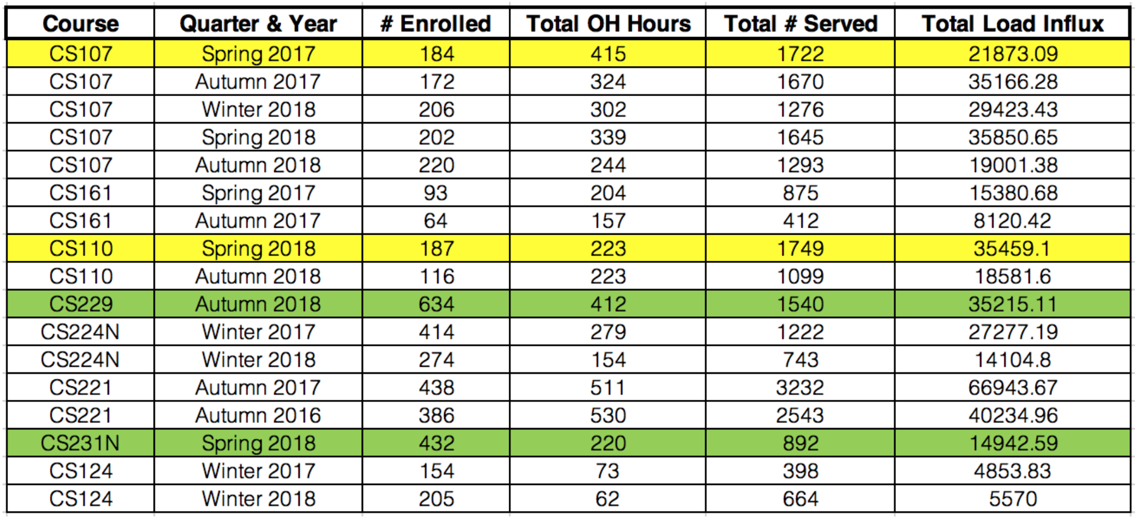
For scheduling, qualitatively we would like for schedules that match high TA allocation with relative peaks, but quantitative metrics are much trickier. At first, we opted to use cosine similarity between our optimized TA assignments and the actual load influx and compare it to the cosine similarity of the actual schedule and the actual load influx, since this intuitively gives us a measure of how well our schedule would have fit the actual student demand that quarter compared to the actual TA schedule. However, this method is flawed, simply because the actual student demand spike heavily depends on the intricacies of the schedule that was actually implemented. We saw that load influx for any slot was extremely correlated ($r = 0.34, p < 1e - 10$ by `cor.test()` in R) with the number of servers assigned, as well as extremely correlated ($r = 0.26, p < 1e - 10$) by `cor.test()` in R) with a binary figure representing whether or not an entry represented the first OH within the past six hours. Although we assume that these features are approximately uniformly random for any given hour, they still remain the source of considerable variation in load influx. Thus, we make the simplifying assumption that for the purposes of our scheduling component, our predicted load influx is the real load influx, and the cosine similarity between that and our proposed schedule will be our metric. We acknowledge this as a weakness of our study, and a weakness of assessing the efficacy of new policies using past human behavioral data in general.

*Note that this is a shared project between CS229 and CS221. For CS229, we focused on a more theoretical approach in predicting load influx by designing new loss functions catered towards data with high variance and fluctuations. We evaluated the new loss function with other commonly used loss functions such as mean squared error (MSE), mean absolute error (MAE) and Huber in regression. We also combine an ensemble of approaches to fine tune our prediction by using signal processing practices such as Hann window smoothing, early stopping and normalization, as well as multimodal classification using SVMs and random forest models. In CS221, we focus on LSTMs for prediction, modified Gibbs Sampling and error analysis. Through Gibbs sampling, we suggest TA allocations to surge timings given realistic schedule constraints.*

## 2 Infrastructure

Since our goal is specifically targeted towards Stanford's office hours, no public datasets were available. Also, in order to capture the intricacies of different classes/quarters as well as as many complex features that give information about why students would like to go to office hours in general, we needed our data to be high-dimensional. Thus, we manually set up complex pipelines to collect archived data from Queuestatus, with augmented features for each hour using per-course information taken from Carta, course syllabus, and the Stanford academic calendar. Through customized JSON parsing, we were able to obtain a combined 17 quarters' of data across 7 prominent CS classes. After preprocessing to remove all entries with zero serves and signups, we ended with 4672 hours', or just under 200 straight days' worth of OH data. A summary is shown in Figure 1 below.

We experimented with a plethora of features to augment our dataset with, and decided on the following predictors based on a combination of our own experience as students and significant correlation with load influx. On a per-class basis, we used: number of enrolled students, instructor rating, and proportion of freshman/graduate/PhD students enrolled. On a per-hour/day basis, we used: days until next assignment due, days after previous assignment due, days until an exam, hour of day, weekdays. For the hourly/daily features, validation testing found that one-hot bucket encodings were more effective for predictions. Day differences were bucketed in ranges of 10 to 5, 4 to 3, 2



| Course | Quarter & Year | # Enrolled | Total OH Hours | Total # Served | Total Load Influx |
|---|---|---|---|---|---|
| CS107 | Spring 2017 | 184 | 415 | 1722 | 21873.09 |
| CS107 | Autumn 2017 | 172 | 324 | 1670 | 35166.28 |
| CS107 | Winter 2018 | 206 | 302 | 1276 | 29423.43 |
| CS107 | Spring 2018 | 202 | 339 | 1645 | 35850.65 |
| CS107 | Autumn 2018 | 220 | 244 | 1293 | 19001.38 |
| CS161 | Spring 2017 | 93 | 204 | 875 | 15380.68 |
| CS161 | Autumn 2017 | 64 | 157 | 412 | 8120.42 |
| CS110 | Spring 2018 | 187 | 223 | 1749 | 35459.1 |
| CS110 | Autumn 2018 | 116 | 223 | 1099 | 18581.6 |
| CS229 | Autumn 2018 | 634 | 412 | 1540 | 35215.11 |
| CS224N | Winter 2017 | 414 | 279 | 1222 | 27277.19 |
| CS224N | Winter 2018 | 274 | 154 | 743 | 14104.8 |
| CS221 | Autumn 2017 | 438 | 511 | 3232 | 66943.67 |
| CS221 | Autumn 2016 | 386 | 530 | 2543 | 40234.96 |
| CS231N | Spring 2018 | 432 | 220 | 892 | 14942.59 |
| CS124 | Winter 2017 | 154 | 73 | 398 | 4853.83 |
| CS124 | Winter 2018 | 205 | 62 | 664 | 5570 |

Figure 1: Summary of dataset scraped from Queuestatus. White: Training set. Yellow: Testing test(Seen courses). Green: Testing set (Unseen courses)

to 1, and 0. Hour of day was evenly bucketed into morning, noon, afternoon, and evening. Each entry corresponds to one hour of OH, and every entry in the same course/quarter shares the same course/quarter features.

As our ultimate goal is to predict entire unseen quarters, we separated our training/validation/test sets by entire quarters. Due to our limited sample size, we use K-fold cross validation to tune hyperparameters, where K is our number of quarters. Our test set consisted of 4 total classes: CS110 Spring 2018 and CS107 Spring 2017 as unseen quarters of classes we trained on, and CS224N Winter 2018 and CS231N Spring 2018 as entirely unseen courses. Our training set thus consisted of the remaining classes, totaling 13 quarters' of data between 5 unique classes.

## 3 Baselines, Oracles and Initial Model: Fully Connected Networks

We first implemented a linear regression model as a baseline, which takes in inputs (among which are days until next assignment is due, time of day, day of week, week number, days until exam, hours spent per week etc.) Our inputs are $x \in \mathbb{R}^n$ where we have $n = 30$ features, and we aim to predict load influx $\hat{y} \in \mathbb{R}$. Using the same input features, we designed a feed-forward fully connected network, with 2 hidden layers of size 15 and 8 respectively. ReLU activation was used on hidden layers with a final linear activation layer, compiled with Adam optimizer. We also tested features such as normalized inputs and early stopping, and experimented with different losses such as mean absolute error and mean squared error.

For oracles, all team members did 25 randomly chosen examples each, where we were presented with the same input features that the neural network receives, in addition to the load influxes before and after each of the examples. We had a root mean squared error of 94.1 for oracles, with the initial experiments reported below. From doing this oracle, we noticed that while load influx tends to follow a somewhat regular pattern, there were often large spikes that do not seem to correspond with any features. This tells us that even with a great model, we should expect high variance.

## 4 Sequence-to-sequence model for prediction using LSTMs

### 4.1 Initial experiment and motivation: Univariate LSTMs

We first experimented with univariate LSTMs as a baseline, where we train on the first 7 weeks of the course and test it on the final three weeks on the course since univariate LSTMs. Initially, we thought that the intricacies such as instructor methods, student interest in the quarter etc. could be learned in the first few weeks, which we then test on the unseen weeks. A quick demonstration is as shown below.

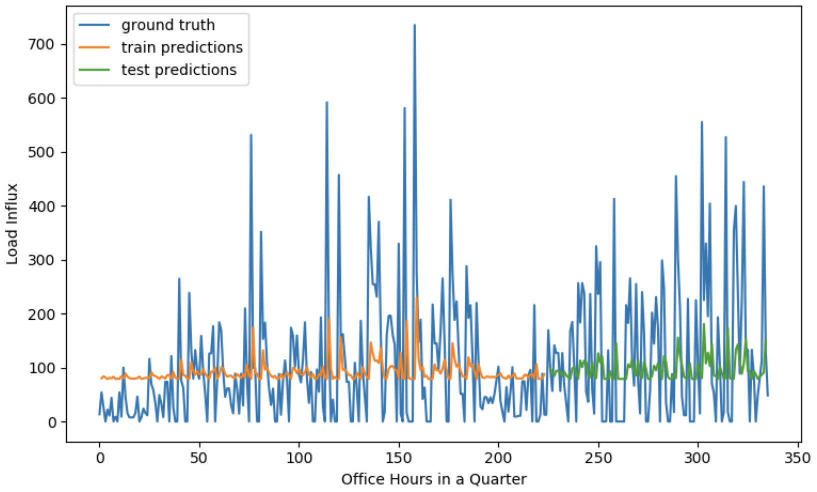


Figure 2: Example of univariate LSTM, where we train on first few weeks(orange curve) and test on the rest (green curve). We detect spikes of load influx, as spikes coincide at surge timings.

However, we quickly realize that it fails to generalize across different classes in different quarters as we made the assumption that we have starting data or seeds for the univariate LSTM. It is also biased as it has not taken into account surges during finals week, since it was trained for midterm season. When testing for a different class for an unseen quarter, we lack the data available. Therefore, we look towards other sequence models with autoregressive nature such as multivariate seq2seq models that would be more applicable to unseen classes.

### 4.2 Multivariate LSTMs Architecture

In this model, we make the assumption that each office hour is correlated to the few hours that precede it. We formulate the problem as such: let $k$ denote the window size and $n$ denote the number of features we have. Given a past window $[x_1, x_2, \ldots, x_k] \in \mathbb{R}^{k \times n}$, we seek to predict the load influx of the next hour, $\hat{x}_{k+1}$.

To encapsulate this sequential nature of the data, we use a **multivariate Long Short Term Memory (LSTM)** recurrent neural net architecture. The sequence-to-sequence (seq2seq) model consists of two parts - the encoder and the decoder. The encoder reads through an entire batch of data at once, encapsulating the information in hidden states, which it then passes to the decoder. The decoder makes load influx predictions given some set of features regarding the predictions. The model is both multivariate and autoregressive as inference is dependent on not just the prediction for the previous time step (i.e. time slot) but also for the set of features for the current time step. We set the window length for the encoder to 10, which corresponds to the max number of office hours that we assume can be scheduled per day, with the window length at the decoder set to 1, so the model only looks at the previous prediction and features for current time step at inference time.

### 4.3 Experiments

We tested the LSTM model on our full train set with the exception of CS107 Winter 2018, which was used as the test set. The results for different model hyperparameters have been tabulated. Because we expected the LSTM to overfit the limited dataset, we added a regularization term to the MSE loss, the the regularization parameter varied as a hyperparameter.

After tuning our hyperparameters, we compare the performance of our seq2seq model with the fully connected network and oracles as shown in Table 3, where multivariate LSTM achieves one of the lowest root mean squared error among the different model ensembles we experimented on. The FCN, which was implemented and discussed in detail in CS229, had a slightly better result.

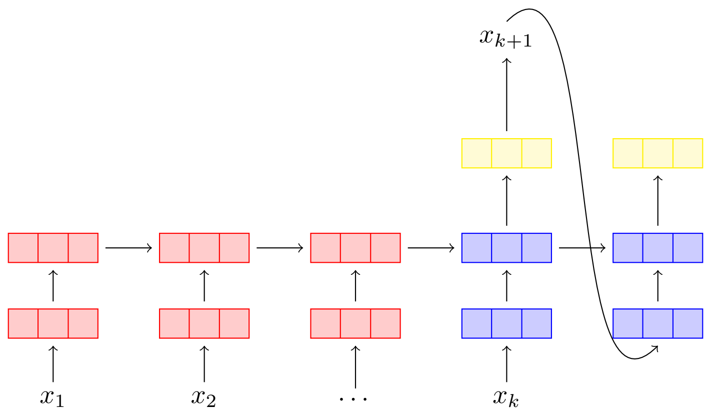


Figure 3: Illustration our our seq2seq model with many-to-one structure.

Table 1: Results for training and testing the seq2seq model for load influx predictions

| training iterations | learning rate | batch size | hidden state dimensions | regularization parameter | RMSE (train) | RMSE (test) |
|---|---|---|---|---|---|---|
| 10000 | 0.01 | 256 | 256 | 0.003 | 42.2 | 126.9 |
| 1000 | 0.001 | 512 | 128 | 0.01 | 93.3 | 122.4 |
| 2000 | 0.001 | 512 | 128 | 0.01 | 99.1 | 115.6 |
| 5000 | 0.001 | 512 | 64 | 0.01 | 100.4 | 111.2 |
| 2400 | 0.001 | 512 | 512 | 0.01 | 93.3 | **107.4** |

### 4.4 Error Analysis and Final Evaluation on Test Set

We hypothesize that while LSTMs trained on MSE loss work well with temporal data, our fine-tuned FCN trained with mean absolute error loss slightly outperforms LSTMs. We hypothesize that training on mean absolute error lose makes the model more robust towards outliers. Moreover, we see a high variance in LSTMs while predicting on the test set. As we manually collected the data, we hypothesize that with more data across different quarters and classes, LSTMs would outperform FCNs due to the low bias-high variance phenomena we face in our LSTM models.

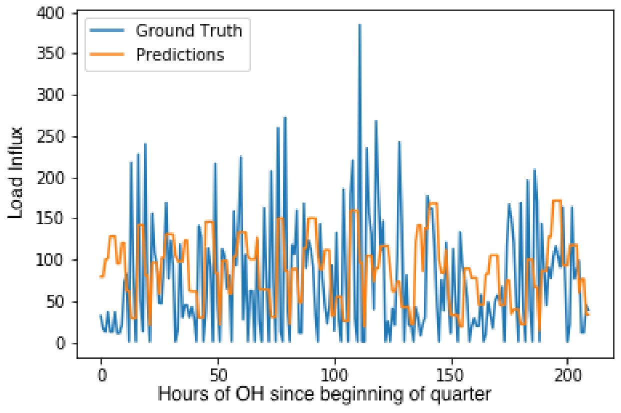


Figure 4: FCN predictions on CS107 Autumn 2018, in leave-one-out validation

Table 2: Evaluation of baselines, initial model and oracles.

| Model | RMSE (Load Influx) |
|---|---|
| Baseline (Linear Regression) | 125.2 |
| FCN, Normalized Inputs, Early Stopping | 112.4 |
| Multivariate LSTM | 107.4 |
| FCN, Log Transform Inputs, Mean Absolute Error, Early Stopping | **106.11** |
| Oracle | **94.1** |

Thus, we move to our testing set. With our best model in the leave-one-out validation phase (the FCN), we obtained an avg. RMSE of 124.466 for our set of seen courses in an unseen quarter, and 106.478 for our set of unseen courses in unseen quarters. Furthermore, similar to Figure 4), the relative locations of spikes were well captured in all four classes; however, in our unseen courses set, the magnitudes of the spikes were almost double in size of the spikes of the ground truth. From this, we see that our model remains relatively robust between both seen and unseen courses in terms of spike location and overall accuracy, but not spike magnitude. While testing on unseen classes, we notice that even our best fully-connected feed forward model fails to predict magnitude of surges and spikes in load influx.

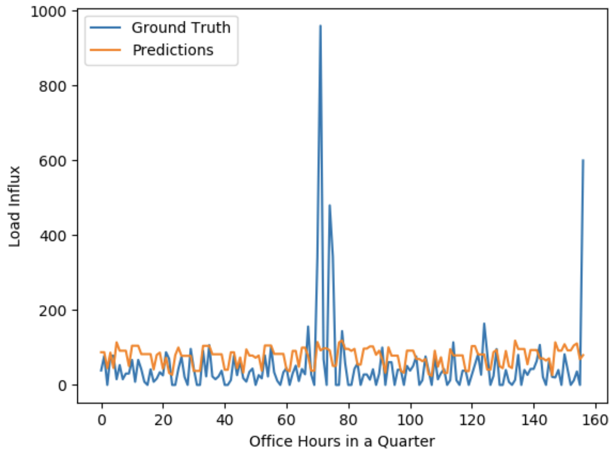


Figure 5: Case where FCN failed on extreme dataset (CS161 Autumn 2017).

We also note that in some extreme cases as per figure 5, the model fails to predict the surge timings and does not show any different trends in different weeks. For instance, for CS161 Autumn 2017, we hypothesize that assignments this term take less time than most classes in the training set, and more attention is paid towards the midterm which explains the huge spike mid quarter. Furthermore, CS161 offers group many group office hours for problem sets, which do not use Queuestatus. However, this model fails to predict the spike during midterm season. This fits our expectation that our model does not perform as well when generalizing to other quarters and instructors, due to the limited dataset that we manually collected.

As a sanity check of the real-life efficacy of our model, we implemented a GUI and demo pipeline that produces hourly predictions for any hypothetical course and quarter in 2019, provided basic course information (exam dates, assignment dates, students, proportion freshmen, etc. as described in features). As a first test, we inputted a custom, freshman-dominated class that has 1000 students, weekly assignments, two midterms, and a final.

We see that this model gives completely reasonable results, which align with our preliminary statistical tests that load influx correlates positively with week number and negatively with weekday/days left until assignments are due. Surprisingly, midterms did not have a significant effect. Interestingly, we see a significant jump in the final few weeks of class: this may be because many classes (such as CS107) tend to have significantly more difficult assignments then. Changing the parameters of the class, we notice that the mean load influx increases marginally with number of students, and

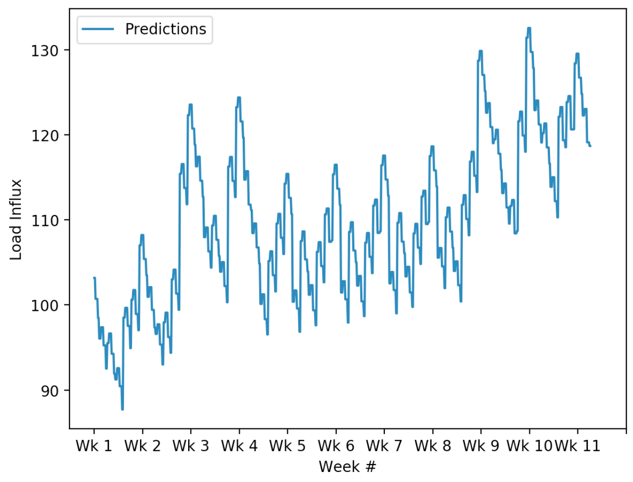


Figure 6: Weekly load influx predictions made on hypothetical course

decreases with proportion of PhD students (perhaps they have too much research going on, or just already know everything...).

## 5 The Scheduler

We have thus far 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.

### 5.1 Defining optimality of an assignment

To estimate how 'valid' a TA schedule is, i.e. how much a schedule is capable of meeting the student demand fr office hours for a given quarter, we tried to come up with a metric that optimized how much the array of TAs assigned per time slot overlapped with the distribution of predicted loads for a quarter. We first convert the TA assignments to an array that lists the number of TAs assigned to a given hour in the quarter, which we defined as $\overrightarrow{T_a}$, while the vector of predicted loads is defined as $\overrightarrow{\hat{Y}}$ Thus, we can define the overlap as the **cosine similarity** of the vectors,

$$similarity = \frac{\overrightarrow{T_a} \cdot \overrightarrow{\hat{Y}}}{||\overrightarrow{T_a}|| * ||\overrightarrow{\hat{Y}}||}$$

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 until the end to obtain a complete quarterly schedule.

### 5.2 Modeling the problem as a CSP

#### 5.2.1 Framework

We initially tried to 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 divided the quarter into 2-week intervals, and assign individual schedules for every biweekly interval. The 2-week period was set 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 goal would be to solve for each 2-week period independently, then append the results at the end.

The domain of each TA was a *2k*-tuple of time slots, with *k* being the maximum number of hours clocked per week. Each element of the tuple was a time slot in which the TA was available. TAs had slots removed according to 2-3 recurring slots randomly selected from popular class patterns (MWF, TT, MW) and sample from random times, as well as and 4-5 random 1-2 hour commitments.

#### 5.2.2 Obstacles

The major obstacle when attempting to solve the CSP was computational efficiency. For a 2-week period, the domain size for each variable scaled proportional to $O(N)$, with N being the number of time slots for which the TA was assigned. We analyzed the computational efficiency of this model, and found it has a runtime of $O(|n!|^N)$, where n is the number of hours a TA is available. Assuming each TA has availability for roughly 40 hours a week, with 10 TAs, this easily becomes exponentially difficult to compute.

### 5.3 Gibbs sampling to maximize correlation

To make the problem solvable within a reasonable amount of time, we ease the bounds on how accurately we want to solve the final problem. We initiate each variable with a list of time slots chosen randomly. For each iteration of Gibbs Sampling, we iterate over each of the *2k* values and sample a new value based on the **change** in full weight of the assignment made by assigning a value.

$$P(X_{ij} = x) = \delta(\frac{\overrightarrow{T_a} \cdot \overrightarrow{\hat{Y}}}{||\overrightarrow{T_a}|| * ||\overrightarrow{\hat{Y}}||})/(\delta x).$$

Plotting the progress in the correlation between the two final vectors, we observe the plots in Figure 6:

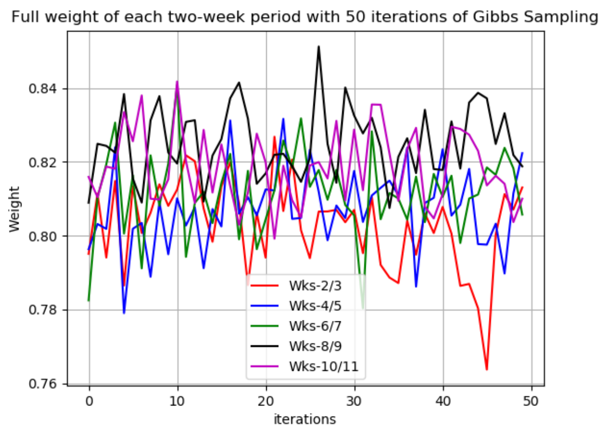


Figure 7: Correlation value for each iteration of Gibbs Sampling

The correlation for different weeks converge onto a solution within a small number of iterations. Overall, we observe that assignments for the scheduler roughly match the predictions for load influx made by our predictor. We observe that our proposed TA schedules have approximately equal ( 0.79) cosine similarities to the proposed schedule as the optimized schedule does to the real schedule (A random assignment of TAs corresponds to  0.5 cosine similarity). This is fairly impressive — as mentioned before, the real load influx is heavily dependent on the real schedule.

### 5.4 Analysis of scheduler

As shown in **Figure 8**, the cosine similarity between load influx predictions and assigned slots for our schedule is almost identical to the similarity between these two metrics for the ground truth schedule. This reveals an important insight about the pattern of load influx for *any* quarter: the load influx is

| Cosine similarity, actual schedule | Cosine similarity, optimized schedule |
|---|---|
| 0.794 | 0.789 |

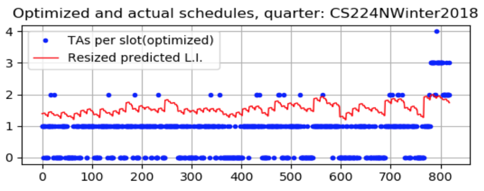


Figure 8: Predicted load influx aligns with Gibbs-sampling produced TA schedule

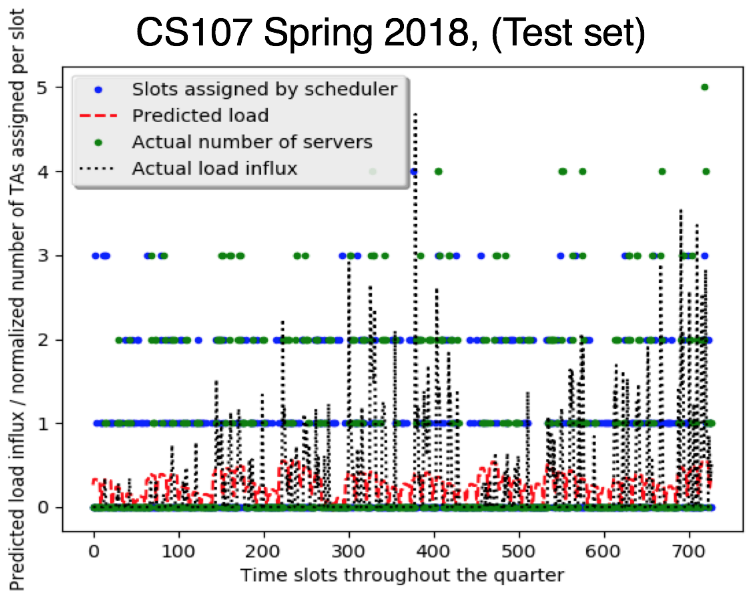


Figure 9: FCN predictions and Gibbs scheduling performs well on unseen test quarters, producing TA assignments that align roughly with spikes in student demand

heavily dependent on the actual assignment of office hours - students tend to congregate more when more TAs are assigned to a given time slot. Thus, the best application of our scheduler would be to act as a recommender system of TA schedules for a new CS course where prior data on office hour patterns is unavailable. Our scheduler offers the best heuristic available for Stanford CS courses when it comes to scheduling office hours based on expected demand.

**Figure 9** shows the results our scheduler would have produced for a quarter not present in the training set, that of CS107 Spring 2018. We see the prediction algorithm manages to accurately predict the location of surges in demand, with the scheduler assigning more TAs for those slots. Overall, it seems a more even distribution of TAs maximizes our similarity metric.

## 6 Related Work

Unfortunately, predictions for time-series behavior of students are not very well studied. In an interview with senior CS109 lecturer Chris Piech, he expressed that status quo OH scheduling is somewhat arbitrary (loosely based on assignment dates), and the department is currently not basing OH scheduling on any rigorous data. As a first source for technical work, we looked to the approaches of a CS229 project that had a similar goal.

The authors used custom feature extractors to come up with a model that could predict wait times at the LaIR. They made a few interesting observations regarding prediction tasks. The first was that multimodal classification with equi-depth buckets tended to outperform regression approaches in terms of accuracy, indicating that focusing part 1 as a classification problem might be more fruitful. Another observation was that adding layers to the neural network or nonlinearities did little to improve performance. We thus focus on emphasis on feature engineering by combining and augmenting Queuestatus with course data. In contrast to this project's dataset, QueueStatus eclipses the LaIR framework in terms number of courses served, which allows us to generalize more. One limitation of QueueStatus is it collects less problem-specific information than LaIR does, so we will have to rely on other forms of feature extraction in our model. Also we provide an orthogonal approach: instead of using equi-depth buckets for classification, we focus on regression and predicting spikes with different models such as LSTMs instead.

*Multi-task sequence to sequence learning*

Luong et al. explored different setting for sequence to sequence models that uses recurrent neural networks to map variable-length input sequences to variable-length output sequences. Although not specific to our task, we are interested in how many-to-one and many-to-many are structured, where encoders and decoders are applied in the realm of machine translation. We are interested in using these models to predict office hours influx and spikes, by also training encoders and decoders with LSTM (Long Short-term memory) as recurrent neural network units that are more suitable to predict outputs in the time domain. This work is thus complimentary as we generalize it to the domain of office hours and education.

## 7 Conclusion and Future Works

Overall, our project provides the first general-use model for predicting student demand at Stanford CS office hours. Using hourly Queuestatus data and course information, we were able to generate realistic predictions for office hours load in a wide range of CS classes, and implement a scheduler that does reasonably well.

To build a more robust model to counter the high variance problems we face, we hope to include more classes in the dataset besides CS classes (generalizing to other classes in different departments) as well. Increasing the quantity and distribution of our dataset would be a key step next in prevent overfitting.

Furthermore, we constructed a basic GUI in R that, given basic course information, generates OH hourly load influx for the whole quarter within a minute. So far, Chris Piech has expressed interest in using our model next spring. Given more time, we would like to extend our predictions to more classes, and perhaps even other universities using Queuestatus.

## 8 Implementation

Implementation is uploaded to Codalab at

https://worksheets.codalab.org/worksheets/0x289187773d624274bbc65bf7a0dd39fd/

Full work is uplaoded to Github at

https://github.com/Zheng261/DeepQueueLearning

## 9 References

[1] Troccoli, N., Capoor, B. & Troute, M. (2017) Predicting CS106 Office Hours Queueing Times. *Past CS229 Project*

[2]. Luong, M. T., Le, Q. V., Sutskever, I., Vinyals, O., & Kaiser, L. (2015). Multi-task sequence to sequence learning. *arXiv preprint arXiv:1511.06114.*