COMPGW02: Web Economics

Individual Report

Alexander Cowen-Rivers
SN: 17036378
ucakaia@ucl.ac.uk

ABSTRACT

This report contains the individual data exploration for the project dataset, as well as my approach to the individual Real-Time Bidding strategy. The main contributions of my work are re-defining the bidding problem as a fraud detection problem; to make available tools for identifying outliers (fraudsters), which we redefined as click-through users. Secondly, creating a reinforcement learning environment from the bottom up, as well as creating the multi-agent environment which built upon this. Lastly, implementing a modified linear approximation version of a state-of-the-art Deep Reinforcement Learning to Bid model for the single, model-free agent, leading us to develop the novel model-free reinforcement learning to bid multi-agent algorithm.

Keywords

Real-time auction, CTR prediction, bid-landscape forecasting, bid optimisation

1. INTRODUCTION

Problem Context

Since 2009, Real Time Bidding (RTB) has become one of the dominant methods employed in the display advertising [11] industry. In this report, we were tasked with implementing a bidding strategy on the iPinYou dataset [17], in order to compete in an auction.

The RTB problem is an auction, where the 'auctioneer' first broadcast features of each user on their domain to the 'clients', who then process the features and return if they so choose, a bidding price. If the 'client' wins the auction, which is decided in various ways, then the user is displayed the 'clients' advertisement. Where clients are multiple companies wanting to advertise their product, and users are the atypical consumer.

There are many types of auctions systems, for example, in the English auction system you can implement two strands.

- **1st Pay Price**: This is typically seen online, whereby the winning 'client' pays the last bid you promised.
- **2nd Pay Price**: This method dedicates the winning 'client' pays the second highest price promised.

This assignment followed the 2nd Pay Price system. In creating a bidding strategy, for a particular auction system, you usually have two components. One which is utility estimation: approximating the likelihood a user will click through one of the advertisements. The second component is a bidding strategy, which takes your utility estimations and forms a bidding value out of this. A simple example of a bidding strategy would be the linear bidding strategy (1).

$$bid = base\_bid \times pCTR/avgCTR \tag{1}$$

For evaluating the utility estimation models, which produces the necessary probably of a click-through (pCTR), we use the below classification metrics.

- **ROC-AUC**: Receiver operating characteristic area under the curve. This is used to assess the diagnostic strength of a binary classifier.
- **Precision**: A measure of how many correct predictions made, over the total number of predictions.
- **Recall**: A measure of how many correct predictions made out of all possible correct predictions.

The report is structured as follows: in Section 2 we briefly explore the related work in the field. In Section 3.1, we explain the data exploration investigation and then discuss the development of the individual bidding strategy in Section 3.4. Section 3.4 also contains a summary of my individual best performing bidding strategy, best performing regarding a number of clicks on the validation set. I conclude with an analysis of the performance and summary of what I would like to focus on given more time.

The corresponding code [1] for this project can be found on GitHub.

2. RELATED WORK

In [17], the authors provide a basic statistical analysis of the full iPinYou dataset (note that in this project we only use a smaller subset - due to computational complexity and resource limitations). They also provide some benchmarking algorithms for CTR-prediction (logistic regression & gradient boosted decision trees) and bid optimisation (constant, random, linear, bidding below max eCPC).

The authors of [5], introduce an ensemble decision tree with a logistic regression model to predict clicks on Facebook adverts. They highlight the importance of good feature selection and describe how to sensibly down-sample and recalibrate a high dimensional dataset into one that is less computationally intensive to perform inference over.

[1] `https://github.com/uclwe/rtb`

Other papers focus on the problem of bid-landscape forecasting, which involves predicting the market price distribution of a give impression. Specifically [2] forecasts the bid for each sample using GBDT regression and then fits a GMM to generate a campaign level bid distribution.

Non-linear bid optimisation strategies are described in papers such as [15], [16]. More specifically, in [16], the authors describe a strategy which takes into account the budget and the campaign lifetime, wanting to bid on more impressions instead of a subset of high valued ones. This strategy is cost effective as it bids on lower valued impressions, meaning that they save money and the chances of winning are higher. In [15], the authors challenge the idea of using value-based bidding, as they suggest that this strategy favours users with an already high action rate (e.g. click, or conversion). Instead, they suggest a lift-based bidding strategy that is based on the action rate lift i.e. focusing on users that would be more influenced by the ad specifically.

The authors of [1], [14] formulate the real-time bidding problem as a Markov Decision Process (MDP) and introduced model-based and model-free (respectively) reinforcement learning algorithms to develop bidding strategies that are able to additionally consider budget constraints.

The work of [7] is an extension to [1] and formulates the real-time bidding problem with competition as a multi-agent reinforcement learning problem, which they using the actor-critic framework to solve.

3. APPROACH & RESULTS

3.1 Data Exploration

We have various advertiser ID's, which equate to industry categories such as Milk powder and Software. The vast range of advertisement opportunities is one of the crucial factors that makes this dataset an interesting toy problem. All advertisers have a CTR of less than 0.1%, apart from the Mobile e-commerce app install industry, which has a CTR of nearly 0.5%. All advertisers, however, have a similar cost-per-click, which suggests that many of the click through on the mobile advertisements are of much lower value than the click through on other advertisements.

We then choose to analyse the CTR distributions amongst two similar advertisers; Chinese vertical e-commerce and Mobile e-commerce app install, in the hope of uncovering some knowledge of feature importance. We first looked to see if there was a correlation of features with CTR, such as looking at the bi-variate kernel density estimate of the slot width and height features for the impressions which provided a click, and those which did not. There didn't seem to be much difference in separating the data directly by clicks (6). Thus we choose to take a probabilistic approach to exploring the data and instead look at proportions of clicks per a given cluster.

Then we moved on to analysing some of the features we believed to be important, through visualisation.

- **OS:** As expected, IOS users have a much higher CTR of 0.05, due to IOS products being more expensive typically than non-IOS products. We observe the other OS's have a CTR of $\tilde{0}.01$.
- **Browser:** Following on from the OS analysis, we observe Safari 3.1 maintains one of the highest CTR's. Safari comes default with IOS products. Hence this was also to be expected.
- **Weekday:** It appears that weekdays 1-3 are of the higher CTR results, then as the week progresses the CTR slowly decreases.
- **Hour:** We observe the early hours of the day are some of the most valuable ones for RTB, as they maintain a fairly high CTR (reaching peaks of around $\tilde{0}.002$), this then stabilises as the day progresses.
- **Region:** Here we can see the different regions have varying CTR rates, with region 359 having the largest average CTR ($\tilde{0}.0024$) and variance.
- **Slot Size:** Certain slots seem to be of lower CTR performance than others, with the highest performing slot size being the longest rectangular slot of size [1000,90] with a CTR of $\tilde{0}.001$.
- **Ad Exchange:** Impressions data from ad exchange two appears to maintain a lower CTR than impressions which have appeared from ad exchange three. with exchange three having a CTR of $\tilde{0}.001$.
- **Tag:** While the smaller ID tags maintain a similar CTR, a few of the higher tags such as 38,44 and 65 have almost twice the CTR of others at $\tilde{0}.04$.

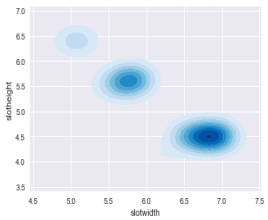

Figure 1: Non clicks ($\tilde{4}00$ examples)

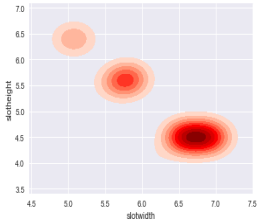

Figure 2: Clicks ($\tilde{4}00$ examples)

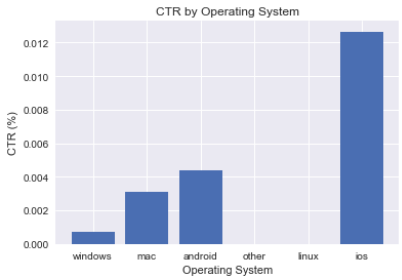

Figure 3: Browser CTR For Chinese E-commerce and Software Advertisers

We noticed through data exploration that the classes are heavily imbalanced, with around $\tilde{1}700$ clicks out of 2M impressions. When looking at the pay price distribution (3.1), we can see that there seem to be a few predominant distributions around winning pay prices, leaving me hopeful at producing a viable strategy.

In conclusion, there seems to be a lot of separation between various clusters of features.

3.2 Basic Bidding Strategy

PLEASE REFER TO GROUP REPORTS.

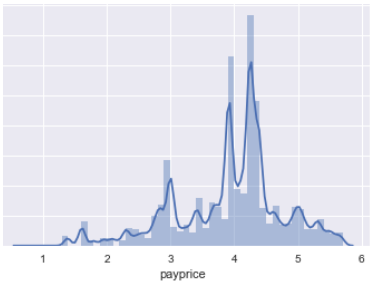

Figure 4: Log Pay Price Distribution

3.3 Linear Bidding Strategy

PLEASE REFER TO GROUP REPORTS.

3.4 Best Bidding Strategy

For my bidding strategy, I chose to first investigate the linear bidding strategy, with focus on improving the pCTR prediction component, to later investigate reinforcement learning methods. As we had already used Gradient Boosting trees and Logistic Regression for the group sections previously, I decided to investigate both neural methods for pCTR prediction. Using one-hot encoded features as my input to the neural models, with the impression click value field as my target. I trained all neural models on three epoch with the same default learning rate, for a fair comparison. I also experimented with class weights being used in the model and not, this proved beneficial for all neural models hence it was a consistent setting throughout architecture exploration. Below I present the various model specifications as well as best click rate achieved after performing a grid search over the base bid values for the linear bidding strategy.

3.4.1 Neural Methods

Model One

I started off with a basic neural network, consisting of two dense layers of 100 units and 50 units. With relu and sigmoid non-linearities, to try to predict the pCTR of an impression. I used the Adam optimiser [8] with a held out validation set to monitor training performance and to perform early stopping if need be on the validation loss. The performance of this model was extremely promising, as it achieved 145 clicks on the validation set and I had barely done any architecture optimisation. I was able to reconstruct impression profiles to a validation accuracy of $\tilde{4}3\%$, using mean squared error as my loss metric.

Model Two

Thus I decided to implement a deeper model, consisting of a five-layer 200,100,50,10 units network. I also added batch normalisation [6] on each layer, due to the overfitting issues that deep networks carry with them. Xavier initialisation [3] was also used for neuron weights. Overall, this model improved on the previous neural model by a fairly large amount, reaching 156 clicks on the validation set.

Model Three

I then thought of using an autoencoder style model in the hope of finding a hidden representation of the click-through users. I used four layers of size 40,30,30,40. Applying dropout [12] to the latent dimensions with probability 0.25. This received a validation score of 152 clicks.

Model Four

Firstly I would like to introduce the reader to a very similar problem I stumbled upon while investigating alternative methods to create a CTR prediction model. In fraud detection, the vast majority of users are not committing fraudulent activity, similar to how the vast majority of impressions in an advertisement bidding will produce no clicks.

After researching standard methods online, I found an interesting example [13] where the authors used an auto-encoder to first reconstruct the input of users who were not fraudulent, then once the model can reconstruct user vectors well you set a threshold with you tune and any user profile which is above the reconstruction error threshold you label as fraudulent. I adapted this by first getting the auto-encoder from model three to recreate impression vectors with their click field marked as 0.

I then plotted a histogram of reconstruction errors for the two classes.

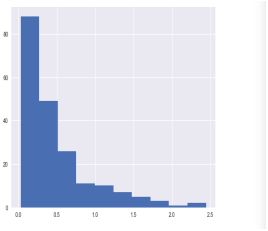

Figure 5: Reconstruction error of click impressions

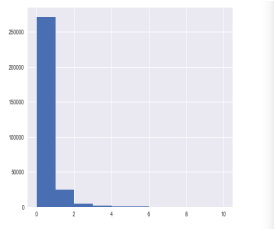

Figure 6: Reconstruction error of impressions without clicks

I first tried searching for a threshold value, which proved extremely poorly, due to the difficulty separating click impressions purely on reconstruction error, achieving only $\tilde{6}0$ clicks.

I decided to try to normalise the reconstruction error so that I could use it in my linear bidding strategy. I logged the reconstruction error and then passed the values into a sigmoid function. The best result I was able to achieve from this method was 77 clicks on the validation set. I also tried other experiments to normalise the reconstruction errors to values between [0,1] to use a pCTR values. However, I have only presented the best performing method.

3.4.2 Reinforcement Learning Methods

Environment Specification

First, we had to create the RL environment, which was a difficult task in itself. We defined the auction environment, named FeatureGrid, as an adaptation of the classic grid world environment. **State Representation** To reduce the features for each state to a reasonable number, we used the feature importance function within the groups highest performing Gradient Boosting Trees CTR prediction model, to select the most important 60 features for state representation, from our $\tilde{9}00$ feature state representation. To our surprise, the model suggested that $\tilde{8}00$ out of our features has an important of $\tilde{0}$.

The auction environment would also need to produce the **initial state**, **reward**, **discount** and **next state** for the

Table 1: Comparison of individual strategies

Approach	Precision	Recall	ROC-AUC	Cost	Clicks	CTR	CPC
Basic Feed Forward Network	0.0194	0.0190	0.8197	93.97%	145	0.001636	40.50
Deeper Feed Forward Network	0.0194	0.0185	0.8434	99.51%	156	0.001302	39.8
Basic Auto-encoder	-	-	-	99.06%	152	0.001283	40.73
Advanced Auto-encoder	-	-	0.5360	100.00%	77	0.000544	81.16
Single Agent DRLB w Linear Approx	-	-	-	99.97%	74	0.0004532	83.27

agent. To satisfy these needs, we first saved a copy of the shuffled training set in each initialisation of the environment, which initialised a memory of the budget consumed by the agent per episode.

When calling the environment.step() function it now produces the first row of the impression matrix as the initial state, sequentially progressing through the impression matrix until episode termination.

Episode termination is achieved by a full pass through the impression matrix, or when the agent has only less than 60 CNY left in its budget.

The **discount** value was set to be 1, as we consider the immediate reward from winning each auction, longer-term rewards are also taken into consideration through agent construction. The reward outputted from the auction environment was set to be the impression value, which in our case was the pCTR. It is this reason why we can use the agent at test time to dynamically adjust its response without ever receiving a true reward.

```
for i in range(number_of_episodes):
    action = agent.initial_action()
    for step in range(number_of_steps-1):

        #Terminates episode
        if AuctionEnv.budget<60:
            break
        reward, discount, next_state = AuctionEnv.step(action)

        action = agent.step(reward, discount, next_state,step)

    #Shuffle data for stochasticity
    X_new, y_new = shuffle(train_X, train_y)
    #Reinitalise enviroment for next episode
    AuctionEnv=FeatureGrid(X_new,y_new,impression_values,budget=6250*10)
```

Figure 7: Single Agent Training Procedure

Agent Specification

The main task of this section was implementing the step function for my agent (3.4.2), to be used in the previously defined environment (3.4.2). I started off by implementing a version of the Deep Reinforcement Learning to Bid (DRLB) model [14], which takes much of the work from the Deep Q-Network model [10], modifying the structure so that it is appropriate for the advertisement bidding domain. The DRLB model-free agent takes the view of not wanting to produce a bid, but rather to adjust an already well performing linear bidding strategy by modifying λ shown in (5).

$$bid\ value = \frac{impression\ value}{\lambda} \tag{2}$$

In the original paper lambda could be adjusted by 11 values, however, in the interest of faster model convergence we reduced this to the following lambda adjustments (3).

$$actions[-0.5, -0.15, -0.08, 0, 0.08, 0.15, 0.5] \tag{3}$$

.

I also added additional sequential features to the state vectors such as time left ratio, left budget ratio and total reward R_total gained as a ratio over the theoretical optimal reward $R*$ of 1793, (1793 possible clicks, each with a maximal probability value of one).

I chose the approximating networks to be as simple as possible; a linear regression model, instead of a three-layer neural network which was the recommended architecture. The justification for this simplification was due to each episode took around 20 minutes to process, and after exchanging correspondence with the lead author of DRLB; Di Wu. I was informed DRLB performed 20,000 training episodes, which was infeasible given the computation resources I had available. My implementation consisted of the various components listed below, with further details followed shortly.

- I create an approximation to the reward function, called RewardNet.
- DynaQ agent with linear approximation, which will sample from its replay memory before each update concerning the temporal difference error on that experience. Note this only ever uses rewards obtained from RewardNet, never updating based on the impression direct reward.
- A reward dictionary, which stores the max episode return for each state-action pair, this is used by the RewardNet to sample from as targets for its SGD updates.

Our RewardNet has a special type of memory replay that only takes action after one episode has been completed, thus a total episode reward value has been collected. The RewardNet keeps a dictionary of the of the max episode return acquired from each state-action pair, to later be sampled as (state, action, total return) tuples for stochastic gradient descent, to optimise its long-term target prediction capabilities. The reason for using total reward was so that we are maximising the $return = \sum_{i=1}^{T} directed\ return_i$. To make this dictionary storable and training computable, We had to downsample the data to 200,000 examples. We implemented a linear model for this. However, it was in sight for us to implement a neural model for this task.

We implemented the adaptive ϵ-greedy policy which selections actions ϵ-greedily concerning equation 4.

$$\epsilon = max(0.95 - r * step, 0.05) \tag{4}$$

Where step is the total step count across all episode and r is the annealing rate hyperparameter, usually set to either 1e-5/2e-5 due to the empirical performance was seen in [14]. We used 1e-6 in our case due to having a much larger time

frame to anneal over as we didn't implement batch processing. The benefits of the Reinforcement Bidding model is that it can change lambda each time step, thus considerably changing its bidding strategy into concerning time left, budget left and total expected reward acquired.

To combat divergence, we standardised all input features as well as clip gradients before an update between -1 and 1. For both the approximating models we use linear regression as a baseline.

For validation purposes, we run the agent through the impression matrix sequentially as normal, however using previous bid/win ratios and bid/pay price rations, we can create a counterfactual budget which we can adjust at each step, reducing it at a set rate proportional to expected pay price.

```
def step(self, reward, discount, next_state,step_num):
    s = self._state.reshape(self.number_of_features,1)
    a = self._action
    r = reward
    self.total_r+=r
    g = discount
    next_s = next_state

    #Update RewardNet
    self.RewardModel.update(s,a,r,step_num,self.total_r) #update rewardnet
    self._action=a=self.behaviour_policy(self.q) #take action at

    #get expected total episode reward from RewardNet
    r=self.RewardModel.transition(s,a)
    self.replaybuffer.append((s,next_s,a,r)) #store s,next_s,a,r
    pred_q = np.dot(self.params[a,:],s)

    #Store TD error
    self.replayerror.append(np.sum(np.abs((pred_q-r))))

    #sample wrst TD error
    n=len(self.replaybuffer)
    self.num_offline_updates
    replayprobs=self.replayerror/np.sum(self.replayerror)
    rand_int=np.random.choice(np.linspace(0,n-1,n,dtype=int),p=replayprobs)
    s,next_s,a,r=self.replaybuffer[rand_int]

    #gradient descent
    reward_est=r+self.target_policy(next_state)
    pred_q = np.dot(self.params[a,:],s)
    dt=np.dot((pred_q-reward_est),s.T)
    dt=np.clip(dt,-1,1)
    self.params[a,:]=self.params[a,:]-self.step_size*dt
    self._state=next_state

    return self._action
```

Figure 8: Agent Step Function

Experiments

The agent was initialised to have a $\lambda_0 = \frac{1}{75275.275275}$, as this was found to empirically perform the best with the impression values used. Performance of the agent was stable. However, it seemed to perform better (3.4.2) at the beginning when undergoing random fluctuations to the bidding strategy rather than when $\epsilon - greedy$ had reduced down to 0.05 where the model was meant to take more exploitative actions. This experiment the epsilon value was set to $1e-6$, for reasons discussed previously.

I chose to re-run the experiment using $r = 3e-7$ in my adaptive ϵ-greedy policy, plotting the epsilon values calculated at each time-step within the policy. This validates my previous hypothesis that the model performance was proportional to the epsilon value, thus the linear approximation adaptation of DRLB was insufficient for learning purposes within this setting.

Analysis

The best performing model I produced was the deep neural network, which when submitted solely achieved a test score of 166 clicks, the validation performance of the various models I created is shown in the table (3.4).

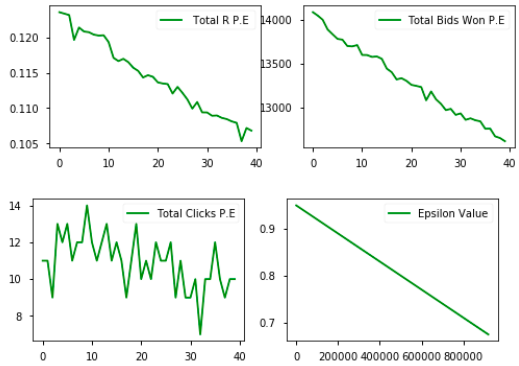

Figure 9: Learning curves with annealing rate 3e-7

3.5 A Further Developed Bidding Strategy

PLEASE REFER TO GROUP REPORTS.

4. CONCLUSION

4.1 Summary

Overall I learnt a great deal from this project, spanning RTB strategies to creating a reinforcement learning environment from the bottom up, and the basic concepts of multi-agent reinforcement learning.

4.2 Future Work

Firstly, I would suggest following the author's recommendation [14] of a multi-agent implementation of DRLB, using non-linear function approximation. To avoid reward overestimated, it would be useful for further implementations of my adapted DRLB algorithm to include double Q-Learning [4].

Regarding the Model 4 (Autoencoder), we believe there was more potential for improvement as the learning curve had not converged fully in the constrained three epochs. It would also be interesting to replace the autoencoder with a variational autoencoder [9], due to it's stronger representation powers.

4.3 Contribution

All other contributions by team members were equal towards the group aspects of this project.

ACKNOWLEDGMENTS

Thanks to Yin Wing for making the API and my group members for useful discussions around the domain. Special thanks to Di Wu, the lead author of DRLB paper, offering implementation advice. Tim Warr for the helpful discussions leading to redefining the bidding problem as one within the fraud detection domain. We would also like to thank DeepMind for providing the GridWorld code for the model environment from previous assignments, which we used as the base to create both the single and multi-agent sequential auction environments.

5. REFERENCES

[1] H. Cai, K. Ren, W. Zhang, K. Malialis, J. Wang, Y. Yu, and D. Guo. Real-time bidding by reinforcement learning in display advertising. In *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*, pages 661–670. ACM, 2017.

[2] Y. Cui, R. Zhang, W. Li, and J. Mao. Bid landscape forecasting in online ad exchange marketplace. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 265–273. ACM, 2011.

[3] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In Y. W. Teh and M. Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR.

[4] H. V. Hasselt. Double q-learning. In J. D. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. S. Zemel, and A. Culotta, editors, *Advances in Neural Information Processing Systems 23*, pages 2613–2621. Curran Associates, Inc., 2010.

[5] X. He, J. Pan, O. Jin, T. Xu, B. Liu, T. Xu, Y. Shi, A. Atallah, R. Herbrich, S. Bowers, et al. Practical lessons from predicting clicks on ads at facebook. In *Proceedings of the Eighth International Workshop on Data Mining for Online Advertising*, pages 1–9. ACM, 2014.

[6] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015.

[7] J. Jin, C. Song, H. Li, K. Gai, J. Wang, and W. Zhang. Real-time bidding with multi-agent reinforcement learning in display advertising. *arXiv preprint arXiv:1802.09756*, 2018.

[8] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[9] D. P. Kingma and M. Welling. Auto-Encoding Variational Bayes. *ArXiv e-prints*, Dec. 2013.

[10] e. a. Mnih. Human-level control through deep reinforcement learning. *Nature*, 518(7540).

[11] S. Muthukrishnan. Ad exchanges: Research issues. *Lecture Notes in Computer Science Internet and Network Economics*, page 1âĂŞ12, 2009.

[12] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15(1):1929–1958, Jan. 2014.

[13] V. Valkov. Credit card fraud detection using autoencoders in kerasâĂŁ-âĂŁtensorflow for hackers (part vii), Jun 2017.

[14] D. Wu, X. Chen, X. Yang, H. Wang, Q. Tan, X. Zhang, and K. Gai. Budget constrained bidding by model-free reinforcement learning in display advertising. *arXiv preprint arXiv:1802.08365*, 2018.

[15] J. Xu, X. Shao, J. Ma, K.-c. Lee, H. Qi, and Q. Lu. Lift-based bidding in ad selection. In *AAAI*, pages 651–657, 2016.

[16] W. Zhang, S. Yuan, and J. Wang. Optimal real-time bidding for display advertising. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1077–1086. ACM, 2014.

[17] W. Zhang, S. Yuan, J. Wang, and X. Shen. Real-time bidding benchmarking with ipinyou dataset. *arXiv preprint arXiv:1407.7073*, 2014.

6. APPENDIX

Model 4

The learning curve for the auto encoder, clearly displaying an un-converged model.

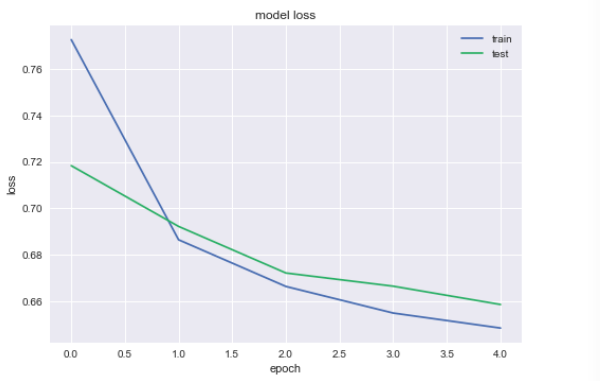

Figure 10: Learning Curve for Model 4

Evaluation of the threshold approach to the auto-encoder reconstruction error, clear difficulty separating clicks based on a large count of false positives and false negatives, leading to poor recall and precision.

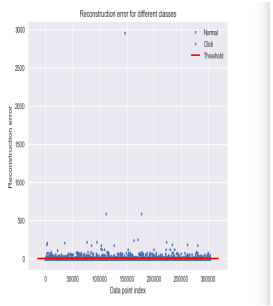

Figure 11: Plot of impressions with threshold imposed

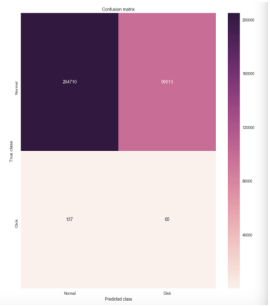

Figure 12: Confusion Matrix Of Best Threshold Value 0.5

DRLB Agent

The equation used to perform the lambda adjustment provided an action a_t selected by the agents policy at step t, is given below.

$$\lambda = \lambda_{initial} + \lambda_{initial} * a_t \tag{5}$$