Neural Review Ranking Models for Ads at Yelp

Vishnu Purushothaman Sreenivasan
visp@stanford.edu

Florian Hartl
florianh@stanford.edu

Abstract

Predicting the click-through rate (CTR) of online advertisements lies at the heart of the business model of many of the world's biggest Internet-based companies like Google, Facebook, or Twitter. While Yelp's main source of revenue is also online ads, one characteristic which sets us apart from other companies is that we often will display a review snippet of the advertised business in the ad creative. Optimizing which review out of potentially thousands to present to the user has a significant impact on the CTR. We show that while a simple bag-of-words model provides a good baseline for selecting reviews, advanced neural models for natural language processing like Long Short-Term Memory (LSTM) networks substantially outperform such a baseline. We also report detailed results of the extensive hyper-parameter optimization we performed.

1 Introduction

Yelp's main source of revenue stems from online advertising. On all supported platforms - desktop, mobile web, and mobile app - business ads are presented to users in various parts of the Yelp product. Figure 1 showcases a sample mobile app ad creative. An advertiser only pays when a user clicks

Figure 1: Example Yelp advertisement.

on the ad, where the final cost of the click is determined by a "generalized second price" auction. Such an approach is formally described in Edelman et al. [1] and Varian [2] and was spearheaded by Google and Yahoo!. Being able to predict the click-through rate (CTR) of an ad accurately in order to get a good proxy for its relevance is essential for the efficiency of such an auction. While there are many factors which influence the CTR of an ad like business, time, and location-based properties, it is also important how the ad gets displayed to the user. In the example shown in Figure 1 a large part of the ad is occupied by a snippet of a review of the advertising business. This is representative of Yelp ads as most will highlight such a review snippet. In previous experiments we were able to prove that by optimizing the selection of a review for a given business, which might potentially have thousands of reviews, based on a simple bag-of-words model, we can significantly improve[1] the CTR of the advertisement.

In this paper, we investigate whether we can further enhance the review ranking by applying the recent state-of-the-art neural language models, like LSTMs. We examine multiple model architectures

[1]Because the CTR metric is used as a proxy for relevance, ideally we would like to select reviews which maximize CTR and therewith provide a better experience for both users and advertisers of Yelp.

and provide a performance comparison, including detailed results of the extensive hyper-parameter optimization we performed.

The rest of the paper is organized as follows. We begin by describing related work in Section 2. Section 3 outlines our approach and experimental setup, including descriptions of the data and the applied evaluation procedure. Results for all experiments are presented and discussed in Section 4 before we conclude the paper and report future work in Section 5.

2 Related work

Three areas of research are related to our work: CTR prediction for online advertising by applying machine learning techniques, learning to rank, and NLP with deep learning.

Industry leaders of online advertising like Google and Facebook have a strong incentive to invest in click prediction because on the one hand it's a central input to the ads auction mechanism as pointed out by He et al. [3] and on the other hand favoring ads with a high CTR leads to a better user and advertiser experience. Consequently, most published literature in this area of research stems from Google and the likes. LinkedIn for example outlines an adaptive approach to CTR prediction based on logistic regression in combination with Thompson sampling in Agarwal et al. [4]. Moreover, two of the most influential research publications for CTR prediction are by He et al. [3], describing a model at Facebook where logistic regression is stacked on top of gradient boosted decision trees, and by McMahan et al. [5] from Google, who provide an overview of a series of experiments they have conducted. None of these works fully apply to our special situation of displaying a review snippet of the advertised business in the ad creative. Since we already know which advertiser got selected and the process of choosing a review happens after the ads auction, one main difference is that we are not concerned as much about the calibration of our CTR predictions as we are about the correct ranking of the reviews.

With their introduction of RankNet, Burges et al. [6] describe a landmark approach for learning to rank. Unfortunately, this paper is not set in the domain of ad CTR prediction. Additionally, while they apply neural networks, the text-based features do not leverage the predictive power of word vectors or Recurrent Neural Networks (RNNs). Therefore, a more relevant research publication was made by Li et al. [7]. They approach the CTR prediction task as a ranking problem, in particular the proposed loss function is a pairwise and pointwise ranking loss. Moreover, word vectors are used as features. That said, they are focused on predicting the CTR of an ad as a whole and not on optimizing parts of the ad creative as it is the case for us with reviews and they also still don't apply RNNs to the text input.

The type of RNNs we examine in this paper are the LSTM and the Gated Recurrent Unit (GRU). LSTMs were first introduced by Hochreiter and Schmidhuber [8] and a thorough comparison between LSTMs and GRUs can be found in Jozefowicz et al. [9]. The jury is still out on which architecture generally performs better. An extension of LSTMs called bidirectional LSTMs is presented in Graves and Schmidhuber [10].

A slightly tangential but related approach to deep learning for textual data stems from the use of Convolutional Neural Networks (CNNs). Kalchbrenner et al. [11] and Kim [12] investigated the efficacy and effectiveness of using CNNs for neural language modeling and propose novel model architectures which achieve state-of-the-art performance when juxtaposing against RNNs.

3 Approach

3.1 Feature pipeline

The inputs to all the models discussed in this paper are detailed in Figure 2. We use the review text as the primary feature source but also supply the models with other associated meta information about the review and the business. The four additional meta features which are not derived from the review text are (1) Review rating (2) Business rating (3) Business review count (4) Business expected CTR obtained from and in-house CTR model. Given that this in-house business CTR model is agnostic to reviews of a business just like the other business features, all of them only function as bias terms to remove the effect of the popularity of a business on the target CTR. Such a bias interpretation

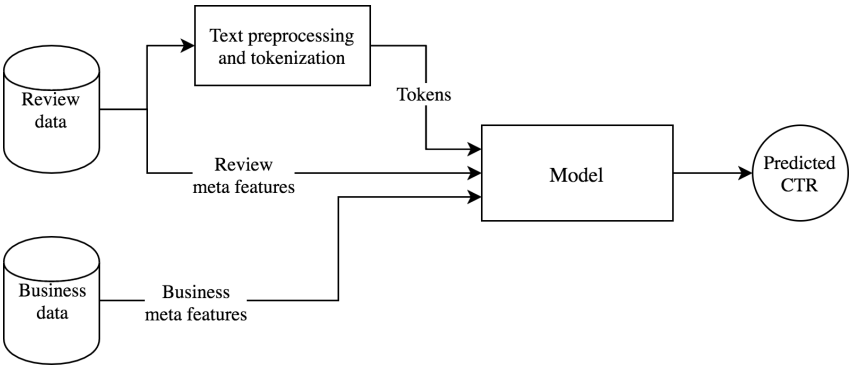


Figure 2: High level overview of the feature pipeline used across all models.

is only valid for simple linear models and does not generalize to non linear models. Overall, the described meta features lead to significant improvements on the baseline model (Section 3.2.1) and hence were used as features in all models outlined in this paper.

For the review text we perform very minimal pre-processing. Some salient points about the review pre-processing pipeline include

- No lower-casing is performed. The effect of lower-casing was studied for the baseline model and was seen to not produce any tangible improvements.
- We truncated the review to 300 characters to account for the shortened snippet shown on the screen. The snippet was tokenized using NLTK's `TweetTokenizer`[2].
- No stemming or lemmatization is performed. These operations were analyzed in the baseline model producing minimal to no differences in the model performance.
- Stop words and punctuations except for the question mark and exclamation point are removed.

3.2 Model architectures

3.2.1 Baseline: Logistic Regression

We already had a baseline model at our disposal which was a logistic regression trained on

- Unigram (~8000) and bigram (~5000) bag-of-words review text features.
- The four additional meta features discussed in the section 3.1.

3.2.2 Neural models

We approach the search over the model space in a two-pronged approach.

1. In the first prong, we search over the space of different model types, like RNNs and CNNs. This is documented in Table 1 and Appendix A contains further details on the different model architectures.
2. Along the second prong of the search space, we fix on a specific model, namely LSTM, and a specific hyper-parameter setting. We then vary each hyper-parameter independently to estimate its effect on model accuracy. Note that we are optimizing a non-convex loss function and hence the specific model performances corresponding to different parameter settings are not deterministic. The experiments for hyper-parameter optimization are documented in Table 2.

Review text representation for neural models
We enforce a maximum of 50 words for each review and pad it with zeros in case its word count

[2]See: https://github.com/nltk/nltk/blob/eac9799f29/nltk/tokenize/casual.py

Table 1: Examined neural models.

MODEL	DESCRIPTION
concat-wordvecs	Concatenate the word vectors of the review
gru	Unidirectional GRU
lstm	Unidirectional LSTM
bidir-lstm	Bidirectional LSTM
stacked-lstm	Two unidirectional, stacked LSTMs
lstm-convnet	CNN on word vectors along with a unidirectional LSTM

Table 2: Hyper-parameter settings (* marks default).

HYPER-PARAMETER	VALUES
retrain word vectors	no*, yes
hidden layers	1, 3*, 5, 7
hidden layer size	32, 64*, 128, 256
dropout	none, first hidden layer, all hidden layers*
LSTM hidden state size	100, 200*, 300, 500
only word vectors of train set	no*, yes

is lower. The padded tokens are masked out when computing the loss and during backpropagation in the case of RNNs whereas for CNNs and concatenated word vectors they cannot be masked out. The number 50 was chosen after investigating the word count distribution of the review text snippet shown on ads. For each of these 50 tokens, we fetch its corresponding word vector from the pretrained, cased 300-dimensional GloVe word vectors which were trained on 840 billion tokens spanning a vocabulary of 2.2 million words[3].

Default model for hyper-parameter tuning
Figure 3 illustrates the architecture of the default `lstm` model we use to study the effects of hyper-parameters. The word vectors of the review tokens are fed into an LSTM with a hidden state size

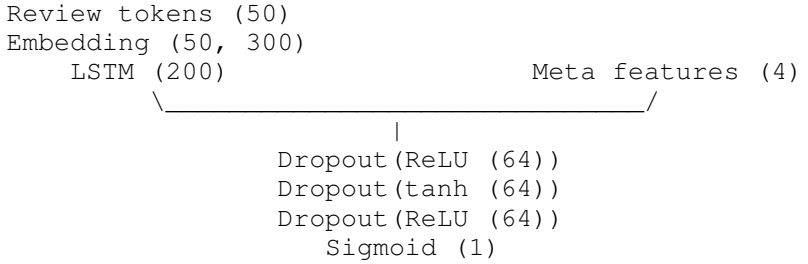


Figure 3: LSTM model with default hyper-parameters.

of 200 units. Thereafter, the output of the last time step of the LSTM is concatenated with our four additional meta features. This vector serves as the input to a three fully connected layers with 64 units each, all of which apply dropout at a rate of 0.5. Both using dropout for all hidden layers as well as the rate of 0.5 are recommended by Hinton et al. [13]. Finally, we use a sigmoid layer to obtain the CTR of the review.

We made a conscious decision to include at least one $tanh$ hidden layer. Since we are concatenating the meta features with the LSTM features, there are no guarantees that the features should be of the same scale. Having features with widely varying scales usually leads to slower gradient descent. Using a $tanh$ layer allows us to squish the inputs to a space of $(-1, 1)$. We later show the effect of

[3]Downloaded from: http://nlp.stanford.edu/projects/glove/ (March 4, 2017).

using only $ReLU$ layers (which do not offer sort of data squishing) and show that keeping the $tanh$ layers does indeed aid the model performance.

3.3 Data

The used data can succinctly be represented as a binomial model:

$$(\text{review\_features}) \rightarrow (\#\text{review\_ad\_impressions}, \#\text{review\_ad\_clicks})$$

Each data point is a review and the target variable is the number of clicks given the number of impressions. For our purposes we just divide the two to obtain the CTR and use the impression count to scale the weight of each data point for training and evaluation. The specific contents of `review_features` have already been discussed in Section 3.1. Overall, the dataset contains 1,350,517 reviews. We randomly divide it into a 50% train, 20% dev, and 30% test set. Because we care about the ranking order of reviews within each individual business we perform the split by business-ID.

3.4 Implementation and hardware

For the implementation of our models we used the Keras[4] API which enabled fast iterations and experimentation while using Tensorflow[5] for the low level optimizations under the hood. To train our models we found that the Adam optimizer with a learning rate of 0.001 and a mini-batch size of 1000 worked best. All our models were trained on a g2.2xlarge or, for more memory intensive models, a g2.8xlarge AWS EC2 instance, utilizing a single GPU.

3.5 Evaluation

For evaluating the models we use two metrics:

1. The Normalized Cross Entropy (NE) as Cross Entropy (CE) will be the loss the models will be minimizing. See He et al. [3] for a detailed explanation on the advantages of using NE over CE.

$$NE = \frac{\sum_{k=1}^{N} i_k \times (y_k \log(p_k) + (1 - y_k) \log(1 - p_k))}{\sum_{k=1}^{N} i_k \times (y_k \log(p) + (1 - y_k) \log(1 - p))}$$

   where, i_k is the number of impressions of a review k, y_k is the observed CTR of the review, p_k is the predicted CTR of the review, and p is the average CTR of the train set.

2. Since intrinsically the optimization problem we are dealing with is a ranking problem, we also employ a ranking metric: `expected_ctr_gain`. For this metric we first compute the sum of all the clicks and impressions in the dataset (`true_clicks`). We then sort the reviews of a business according to the model's predicted CTR in descending order. Next, we compute the expected number of clicks each business would have obtained if it had showed only the best review, according to the model, for all its impressions. For this we need to assume that the observed CTR for each review is the true click through rate. Finally, we sum the expected clicks across all businesses (`model_expected_clicks`) and compute:

$$\text{expected\_ctr\_gain} = \frac{\text{model\_expected\_clicks} - \text{true\_clicks}}{\text{true\_clicks}} \times 100\%$$

4 Results

In this section we will first describe the evaluation results for the different types of neural models listed in Table 1 before we address the effects of the hyper-parameters listed in Table 2.

[4] See: https://github.com/fchollet/keras.

[5] See: https://www.tensorflow.org/.

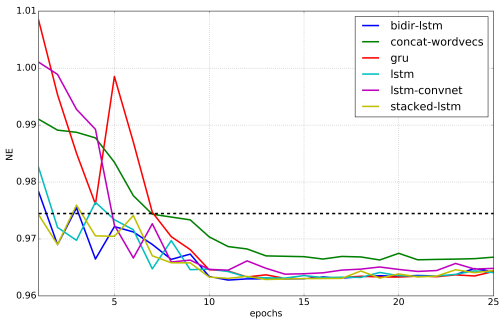


Figure 4: NE performance of different neural models.

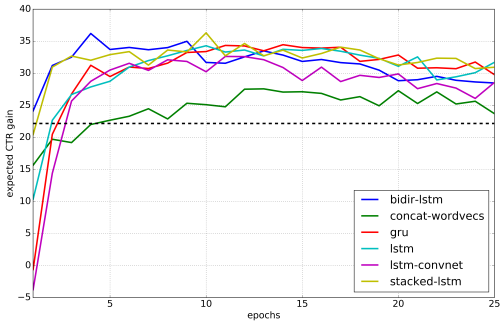


Figure 5: Expected CTR gain of different neural models.

4.1 Comparison of neural models

Figure 4 and Figure 5 illustrate the dev set NE and expected CTR gain performance of the various model types we experimented with and Table 3 provides an overview of the exact numbers, including test set results. Since the NE and the expected CTR gain metrics do not share the same best model across epochs and hence we select the best model for each metric separately based on the dev set and report the corresponding model's dev and test metrics in Table 3. Also note that in both Figure 4 and Figure 5 the baseline bag-of-words model is represented by a black dashed horizontal line.

When looking at the NE evaluation metric, we see that all neural models significantly outperform the previous bag-of-words approach. While RNN models perform significantly better than the concatenated word vectors and the baseline model, the specific type of RNN model doesn't make much of a difference. We do not report the metrics here but we found that using averaged word vectors also performed similar to the concatenated word vectors. Surprisingly, appending a convolutional max-pooled layer to the LSTM output hurts performance and requires further analysis and hyperparameter tuning in order to achieve better results. That the bidirectional LSTM ends up with the best NE score can be explained by the fact that the beginning of the displayed review snippet is more likely to be read then the rest of it. It's also worth mentioning that after about eleven epochs of training, the RNN model performances flatten out.

For the expected CTR gain metric, we see a similar pattern. All models are able to outperform the baseline with RNNs in the lead. The effect of overfitting is more pronounced than for the NE metric as we can see the expected CTR gain of all models decreasing in later epochs. Especially the bidirectional LSTM reaches its peak performance early before producing steadily declining expected CTR gains. Surprisingly, the expected CTR gain metric results differ from the NE metric in that the stacked LSTM is the best model. Since both metrics are reasonable choices, an online A/B test will have to determine which metric and therewith which model we should deploy in our production environment.

Table 3: Neural model and baseline performances.

MODEL	**DEV**		**TEST**	
	NE	EXP. CTR GAIN	NE	EXP. CTR GAIN
bag-of-words	0.97447	22.18	0.97474	25.22
concat-wordvecs	0.96631	27.57	0.96656	27.32
gru	0.96306	34.46	0.96327	36.13
lstm	0.96313	34.29	0.96331	34.79
bidir-lstm	**0.96278**	36.21	**0.96302**	35.57
stacked-lstm	0.96288	**36.30**	0.96315	**38.03**
lstm-convnet	0.96382	32.62	0.96411	34.01

4.2 Effects of hyper-parameters

The detailed graphs for each hyper-parameter can be found in Appendix B. An overview of the exact evaluation results of the best model for each hyper-parameter setting on the dev set and its corresponding metrics on the test set is provided in Table 4. Note that the expected CTR gain for the test set sometimes deviates substantially from the dev set results, e.g. for the dropout model variants, which is a sign that the metric is prone to high variance.

Table 4: Neural model and baseline performances.

HYPER-PARAMETER	DEV		TEST	
	NE	EXP. CTR GAIN	NE	EXP. CTR GAIN
retrain word vectors: no	**0.96313**	**34.29**	**0.96331**	**34.79**
retrain word vectors: yes	0.96626	30.71	0.96648	31.89
hidden layers: 1	0.96327	34.05	0.96352	**37.61**
hidden layers: 3	**0.96313**	**34.29**	**0.96331**	34.79
hidden layers: 5	0.98373	25.47	0.98380	27.25
hidden layers: 7	0.99594	24.94	0.99579	23.43
hidden layer size: 32	0.96364	33.05	0.96383	33.14
hidden layer size: 64	0.96313	34.29	0.96331	34.79
hidden layer size: 128	0.96323	**35.76**	0.96339	**36.29**
hidden layer size: 256	**0.96288**	33.74	**0.96329**	35.62
dropout: none	**0.96267**	33.98	**0.96293**	35.16
dropout: first hidden layer	0.96495	33.63	0.96524	**37.86**
dropout: all hidden layers	0.96313	**34.29**	0.96331	34.79
LSTM hidden state size: 100	0.96336	34.10	0.96357	**37.27**
LSTM hidden state size: 200	0.96313	**34.29**	**0.96331**	34.79
LSTM hidden state size: 300	0.96330	34.28	0.96354	37.23
LSTM hidden state size: 500	**0.96313**	34.09	0.96338	33.91
only word vectors of train set: yes	0.9633	33.18	0.9745	**36.31**
only word vectors of train set: no	**0.96313**	**34.29**	**0.96331**	34.79
hidden layer activations: $(ReLU, ReLU, ReLU)$	0.96432	34.17	0.96455	**35.81**
hidden layer activations: $(ReLU, tanh, ReLU)$	**0.96313**	**34.29**	**0.96331**	34.79

Retrain word vectors. Not retraining word vectors but instead keeping them fixed results in significantly better performance for both NE and expected CTR gain. We most likely don't have enough training data for retraining to be favorable.

Number of hidden layers. One and three hidden layers perform much better than the other options of five or seven layers with three hidden layers having a slight edge over the one layer model, especially for expected CTR gain. This shows that adding more layers doesn't necessarily lead to more accurate models and instead might even lead to the contrary effect. In our case, using five or seven layers doesn't even reach the bag-of-words model performance for NE.

Hidden layer size. This hyper-parameter doesn't have much of an influence on the quality of the model predictions as all variants perform similarly well. While one might slightly prefer values of 64 or 128 based on the expected CTR gain graph, the most useful takeaway is that for higher numbers of units per hidden layer, the training converges faster.

Dropout. While using dropout only in the first hidden layer performs much worse than applying it to all fully connected layers, not using dropout at all leads to surprisingly good results. The model without dropout even beats the all dropout version on the NE metric and also seems to be more resistant to overfitting than expected. Therefore, the conclusion from experimenting with dropout in our case is that this hyper-parameter has a significant impact on the model performance but doesn't follow a clear pattern. Still, the best practice of applying dropout to all fully connected layers worked well.

LSTM hidden state size. There is no noticeable trend as to which setting would be better than others. Therefore, the lowest number of LSTM hidden state units of 100 should be chosen in order to reduce the model training time.

Word vectors from train set only. The idea behind using only the word vectors contained in our train set instead of the full set of pretrained word vectors is to save resources when operating the model in a memory restricted production environment. Following this procedure reduced the memory footprint by an order of magnitude. Since both options exhibit similar performances for NE as well as expected CTR gain, it turns out that only saving the word vectors of the train set is a viable approach for our case.

5 Conclusion

We compared various types of neural models against an already existing bag-of-words approach on the task of ad CTR prediction for reviews of an advertiser. RNNs were able to capture the information contained in the review text much better than the bag-of-words model and significantly outperformed it for both the NE as well as the expected CTR gain metric. The best architecture for NE turned out to be a bidirectional LSTM, whereas for the expected CTR gain metric a stacked LSTM scored the highest. Both models, however, were only marginally better than a simple LSTM or GRU model. In addition to these findings, we presented detailed insights into the effects of various hyper-parameters on model performance and demonstrated the immense importance of finding a good configuration. As due to cost and time restrictions it usually is not possible to try all combinations of hyper-parameter values, modelling experience and having a good intuition about the effects of said hyper-parameters go a long way.

Looking forward, we plan to expand our hyper-parameter search; for nstance employing the plethora of text pre-processing techniques. Thereafter, a meaningful way to utilize the gained knowledge from our hyper-parameter search would be to apply it to the best neural models from our comparison by retraining them with optimized configurations. An additional direction for future work is to further explore the combination of convolutional neural networks with RNNs since we currently don't have a good understanding why this would have a negative effect on model performance. Finally, and most importantly, given that each of our two metrics currently feature a different winning model, the jury is still out on which metric and consequently which model we should prioritize. Therefore, production environment A/B tests for the bidirectional and stacked LSTMs in order to answer this question should be the focus of immediate next steps.

Acknowledgments

We'd like to thank Danqi Chen for mentoring this project and the course instructors, Chris Manning & Richard Socher, together with all teaching assistants for delivering an excellent, highly informative course.

References

[1] Benjamin Edelman, Michael Ostrovsky, and Michael Schwarz. Internet advertising and the generalized second price auction: Selling billions of dollars worth of keywords. Working Paper 11765, National Bureau of Economic Research, November 2005. URL http://www.nber.org/papers/w11765.

[2] Hal R. Varian. Position auctions. *International Journal of Industrial Organization*, 25 (6):1163–1178, December 2007. URL https://ideas.repec.org/a/eee/indorg/v25y2007i6p1163-1178.html.

[3] Xinran He, Junfeng Pan, Ou Jin, Tianbing Xu, Bo Liu, Tao Xu, Yanxin Shi, Antoine Atallah, Ralf Herbrich, Stuart Bowers, and Joaquin Quiñonero Candela. Practical lessons from predicting clicks on ads at facebook. In *Proceedings of the Eighth International Workshop on Data Mining for Online Advertising*, ADKDD'14, pages 5:1–5:9, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-2999-6. doi: 10.1145/2648584.2648589. URL http://doi.acm.org/10.1145/2648584.2648589.

[4] Deepak Agarwal, Bo Long, Jonathan Traupman, Doris Xin, and Liang Zhang. Laser: A scalable response prediction platform for online advertising. In *Proceedings of the 7th ACM International Conference on Web Search and Data Mining*, WSDM '14, pages 173–182, New York, NY, USA, 2014.

ACM. ISBN 978-1-4503-2351-2. doi: 10.1145/2556195.2556252. URL http://doi.acm.org/10.1145/2556195.2556252.

[5] H. Brendan McMahan, Gary Holt, D. Sculley, Michael Young, Dietmar Ebner, Julian Grady, Lan Nie, Todd Phillips, Eugene Davydov, Daniel Golovin, Sharat Chikkerur, Dan Liu, Martin Wattenberg, Arnar Mar Hrafnkelsson, Tom Boulos, and Jeremy Kubica. Ad click prediction: A view from the trenches. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '13, pages 1222–1230, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-2174-7. doi: 10.1145/2487575.2488200. URL http://doi.acm.org/10.1145/2487575.2488200.

[6] Chris Burges, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton, and Greg Hullender. Learning to rank using gradient descent. In *Proceedings of the 22Nd International Conference on Machine Learning*, ICML '05, pages 89–96, New York, NY, USA, 2005. ACM. ISBN 1-59593-180-5. doi: 10.1145/1102351.1102363. URL http://doi.acm.org/10.1145/1102351.1102363.

[7] Cheng Li, Yue Lu, Qiaozhu Mei, Dong Wang, and Sandeep Pandey. Click-through prediction for advertising in twitter timeline. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '15, pages 1959–1968, New York, NY, USA, 2015. ACM. ISBN 978-1-4503-3664-2. doi: 10.1145/2783258.2788582. URL http://doi.acm.org/10.1145/2783258.2788582.

[8] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, November 1997. ISSN 0899-7667. doi: 10.1162/neco.1997.9.8.1735. URL http://dx.doi.org/10.1162/neco.1997.9.8.1735.

[9] Rafal Jozefowicz, Wojciech Zaremba, and Ilya Sutskever. An empirical exploration of recurrent network architectures. *Journal of Machine Learning Research*, 2015.

[10] Alex Graves and Jürgen Schmidhuber. Framewise phoneme classification with bidirectional lstm and other neural network architectures. *Neural Networks*, 18(5):602–610, 2005.

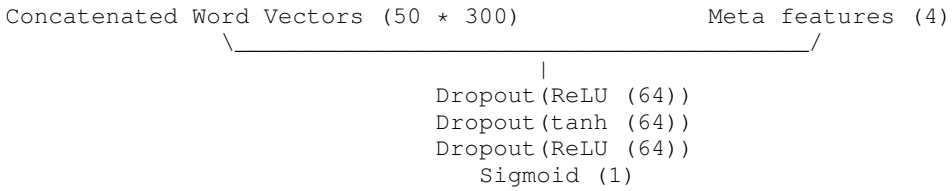
[11] Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. A convolutional neural network for modelling sentences. *arXiv preprint arXiv:1404.2188*, 2014.

[12] Yoon Kim. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*, 2014.

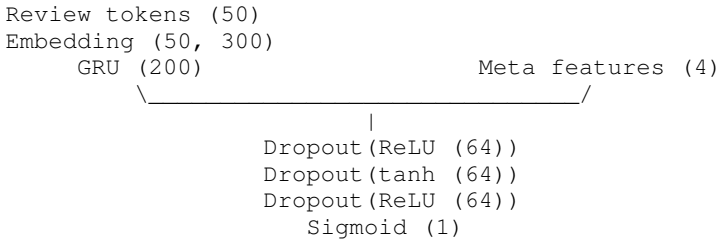
[13] Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *CoRR*, abs/1207.0580, 2012. URL http://arxiv.org/abs/1207.0580.

A Detailed description of model architectures

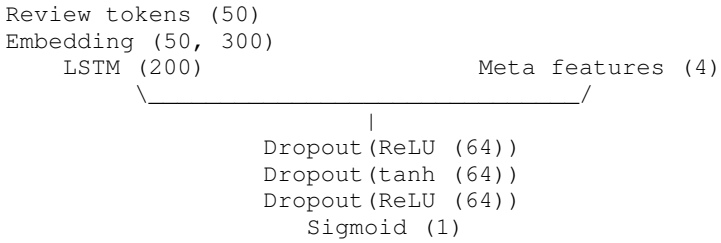
Concatenated word vectors



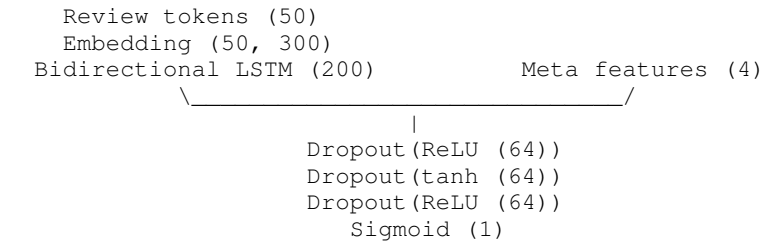
GRU



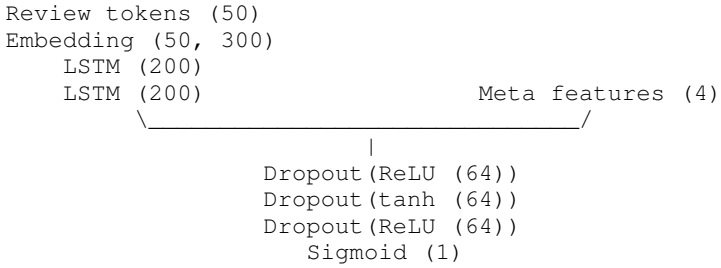
LSTM



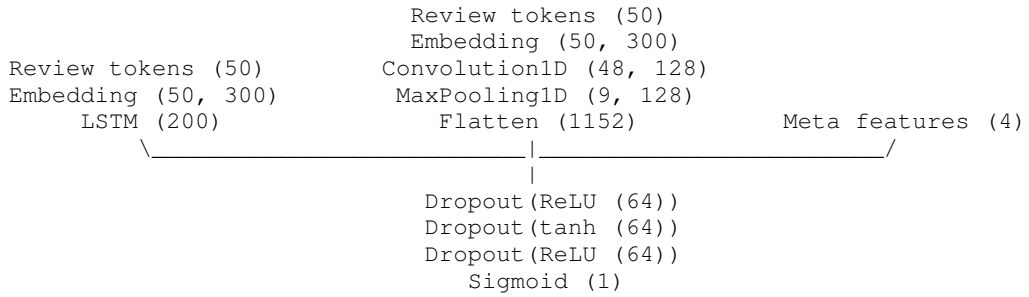
Bidirectional LSTM



Stacked LSTM



LSTM + convolutional neural network



B Hyper-parameter comparison graphs

Retrain word vectors

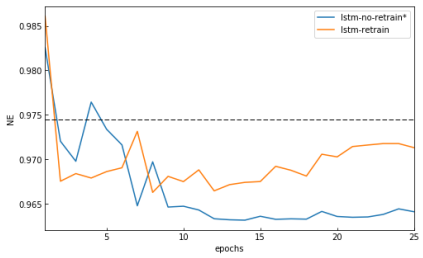


Figure 6: NE for retraining word vector variants.

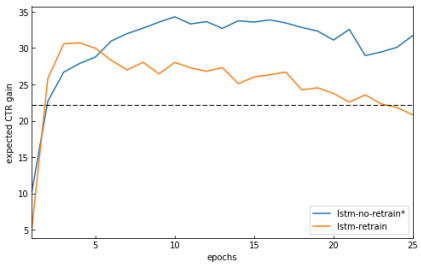


Figure 7: Expected CTR gain for retraining word vector variants.

Number of hidden layers

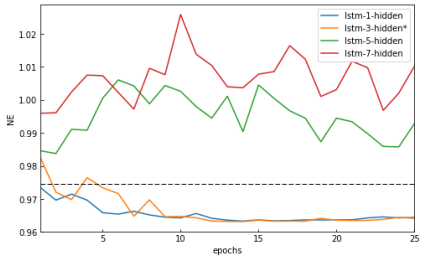


Figure 8: NE for hidden layer variants.

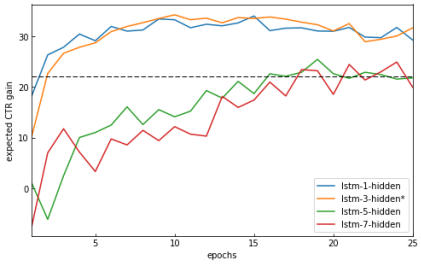


Figure 9: Expected CTR gain for hidden layer variants.

Hidden layer size

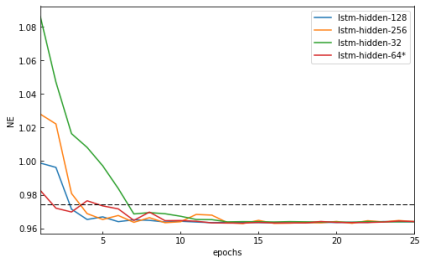


Figure 10: NE for hidden layer size variants.

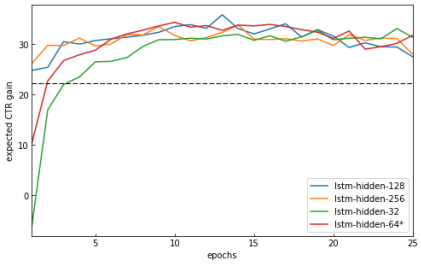


Figure 11: Expected CTR gain for hidden layer size variants.

Dropout

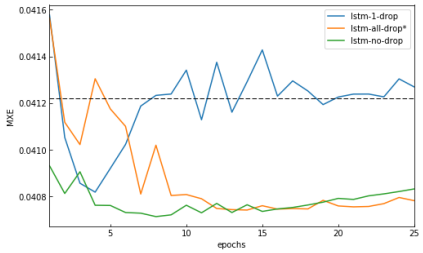


Figure 12: NE for dropout variants.

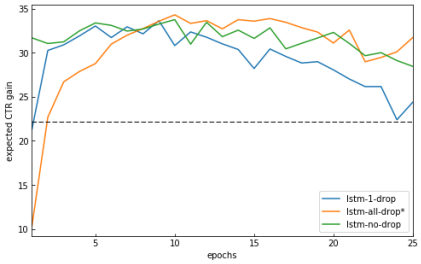


Figure 13: Expected CTR gain for dropout variants.

LSTM hidden state size

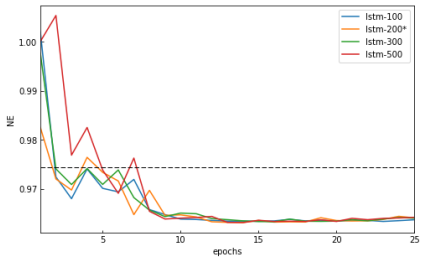


Figure 14: NE for LSTM hidden state size variants.

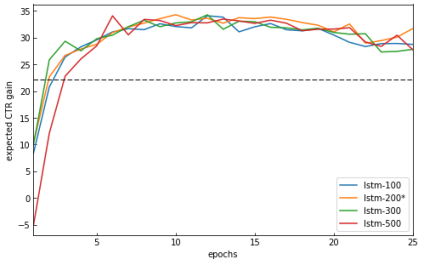


Figure 15: Expected CTR gain for LSTM hidden state size variants.

Word vectors from train set only

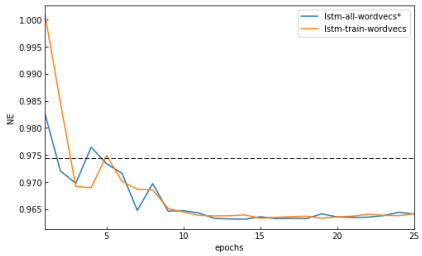


Figure 16: NE for word vector variants.

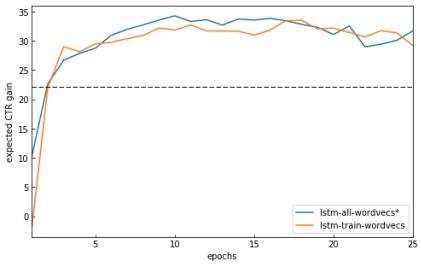


Figure 17: Expected CTR gain for word vector variants.