UNIVERSITY OF NEW SOUTH WALES

ZZEN9444 - ASSESSMENT 2

Writing Neural Networks - Language Processing

Alex Gould | z3062201

April 15, 2021

Contents

1 Executive summary **1**
1.1 Background 1
1.2 Results 1

2 Architecture, Algorithm and Enhancements **3**
2.1 Final model 3
2.2 Alternative models 4

3 Cost function, Optimiser **5**
3.1 Cost function 5
3.2 Optimiser 5

4 Word vectors, Metaparameters, Data preprocessing **6**
4.1 Word vectors 6
4.2 Metaparameters 7
4.3 Data preprocessing 8

5 Avoiding overfitting **10**

1 Executive summary

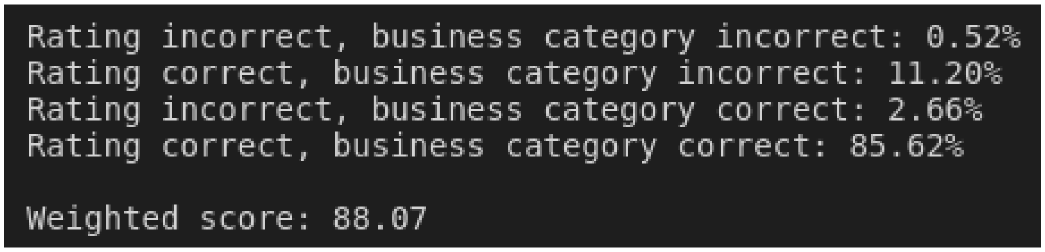
1.1 Background

This brief report provides an overview of the business review classification program, including how it works and the design decisions made. The program uses neural networks to predict user ratings and business categories based on online reviews in free form text format. The training data consists of 50,000 reviews which are split evenly between two ratings (Positive, Negative) and five business categories (Restaurants, Shopping, Home Services, Health & Medical, Automotive).

1.2 Results

The training data was split 80:10:10 between training, validation and a hold out test set. On the validation set, the model achieved a weighted score of **88%**. On the hold out test set and training sets, the model achieved weighted scores of 87% and 92% respectively.

Fig. 1 shows a breakdown of the accuracy of the rating and category models, with the model run on the default 80:20 split.



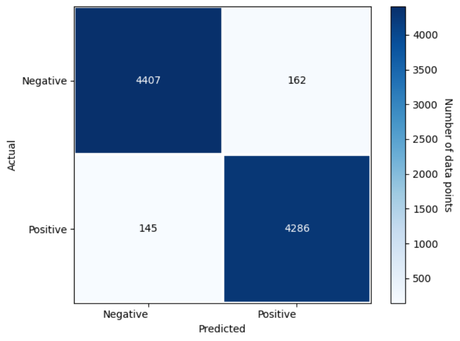
```
Rating incorrect, business category incorrect: 0.52%
Rating correct, business category incorrect: 11.20%
Rating incorrect, business category correct: 2.66%
Rating correct, business category correct: 85.62%

Weighted score: 88.07
```

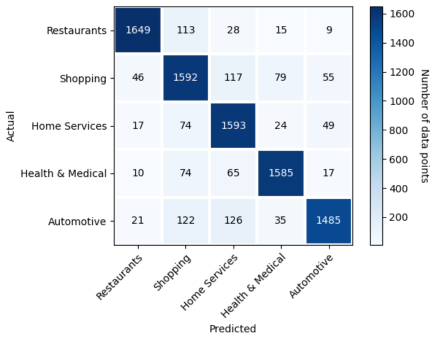
Figure 1: Breakdown of rating and category accuracy

The final submitted model was retrained on the entire dataset. This is expected to improve its performance on an unseen hold out set given more training examples are available for the model to learn general patterns.

Fig. 2 shows confusion matrices for the final model. The model is generally accurate across the categories. There is a tendency for the model to classify reviews under the more general category of "shopping" rather than e.g. "automotive", which reflects the generic nature of many of the reviews, as well as cross selling between categories (e.g. food or accessories being sold at a car wash or mechanic).



(a) Rating



(b) Category

Figure 2: Confusion Matrices

2 Architecture, Algorithm and Enhancements

2.1 Final model

Within the overall model, separate sub-networks were used for the category and rating tasks. This was the first design decision - whether to use a single network with separate outputs for each task, or use two more specialised networks. The latter approach is more computationally expensive, however performance is expected to be higher.

Both networks used a similar Recurrent Convolutional Neural Network (RCNN) architecture, consisting of:

- 50% dropout applied to the input data
- Bidirectional LSTM (single layer)
- ReLU non-linearity
- Fully connected linear layer
- 50% dropout layer
- Max pooling and average pooling (concatenated)
- Linear output layer, with 5 outputs (followed by softmax) for the classification task and 2 outputs for the rating task

This architecture was inspired by a 2015 paper[1]. Key enhancements include:

- Initial dropout was added to further discourage overfitting; the task is small enough that the model can be trained for more epochs to compensate for this.
- ReLU was used in place of Tanh for non-linearity, as a more contemporary choice[2].
- Average pooling was used in combination with max pooling. In general, pooling is used with LSTMs to allow them to be location-invariant; without this, a negative comment in the middle of the review may have less weight than one at the end (or at the beginning for a bidirectional LSTM). Max pooling is used to detect features in this location-invariant manner, whereas average pooling detects an average level across the

whole comment. Both may be useful, and incorporating both allows the model to decide when and where to use each.

2.2 Alternative models

A range of alternative architectures were considered; the arch parameter in student.py can be used to toggle between these:

- Vanilla LSTM with a linear output layer - this was the first network I tried and was mainly used to gain an understanding of TorchText and the a3main.py scaffold
- Vanilla LSTM with an additional fully connected layer before the output layer - adding an additional layer on top of the LSTM improved accuracy significantly; from this point on most architectures produced weighted scores of around 85%
- Stacked LSTM with two layers followed by a fully connected layer - unfortunately stacking LSTM layers added more to the time spent debugging and making code flexible than it added to accuracy
- Bidirectional LSTM with an additional fully connected layer - the bidirectional LSTM increased accuracy somewhat but roughly doubled the computational time required
- Bidirectional LSTM followed by Bidirectional GRU - this architecture was inspired by a top performing entry in the Kaggle Jigsaw Toxic Comment Classification Challenge [3]. The stacking of LSTM and GRU, similar to using LSTMs with multiple layers, in theory allows for richer relationships to be learned, but in practice failed to impress. I note the original architecture included max and average pooling, which I only implemented with the final RCNN architecture.

In general, more complex architectures tended to perform slightly better "out of the box", but with an associated cost in terms of training time. Initial results suggested that the target accuracy targets would be challenging to achieve, and so the emphasis should be on performance rather than run time. To fully assess the range of architectures would require extensive tuning of hyperparameters for each option. I found there was a delicate balance between properly assessing a given architecture and knowing when to move on to something with more potential.

3 Cost function, Optimiser

3.1 Cost function

For the rating model, Binary Cross Entropy was used. This is a standard loss function to use for binary classification tasks. In particular, Binary Cross Entropy With Logits Loss was applied to the linear layer output. Initially, a Sigmoid layer was used with a Binary Cross Entropy Loss, however in initial experiments the Binary Cross Entropy with Logits Loss produced much higher accuracy. The documentation notes that the latter approach is numerically more stable [4].

For the category model, Cross Entropy was applied to the outputs of the softmax. This is a standard approach for multi class classification tasks.

3.2 Optimiser

In initial experiments, Adam and SGD were compared and Adam was found to produce better results "out of the box". Adam is known to work well in most applications with minimal changes to metaparameters [5] and only the learning rate was tuned along with other metaparameters.

It is possible that with proper tuning, SGD with Momentum may provide better results for the final model, given its greater depth and larger training time [6]. However, given the relatively long training time, it would also be challenging to optimise.

4 Word vectors, Metaparameters, Data preprocessing

4.1 Word vectors

Glove 300 word vectors were used in the final model. My initial expectations were that the higher dimensionality of the Glove 300 vectors would allow the model to capture more meaning at the cost of greater computational time.

Fig. 3 and Fig. 4 show the accuracy and run time of the LSTM-GRU architecture during development. At the time, I was optimising the learning rate and hidden size metaparameters for the category model, with a minimal rating model held constant.

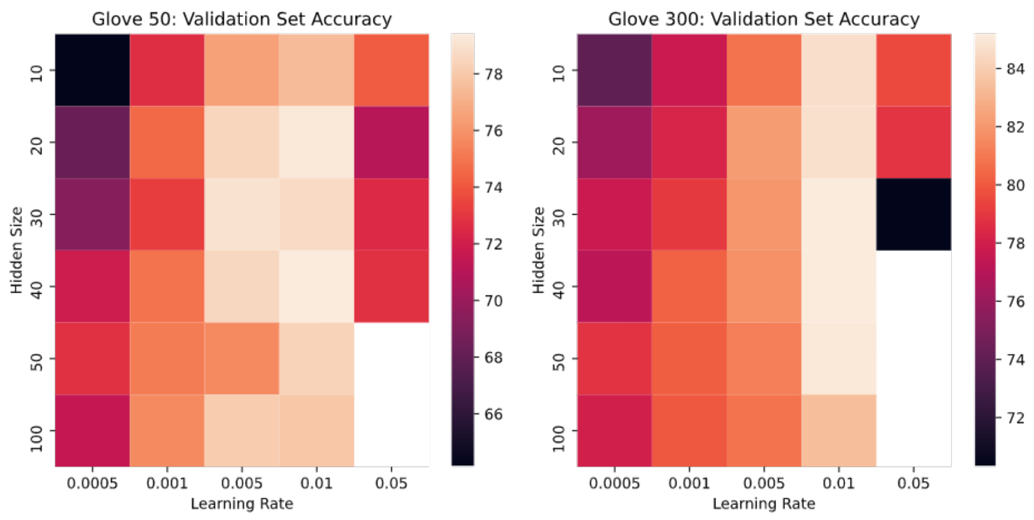


Figure 3: Glove 50 vs Glove 300: Accuracy

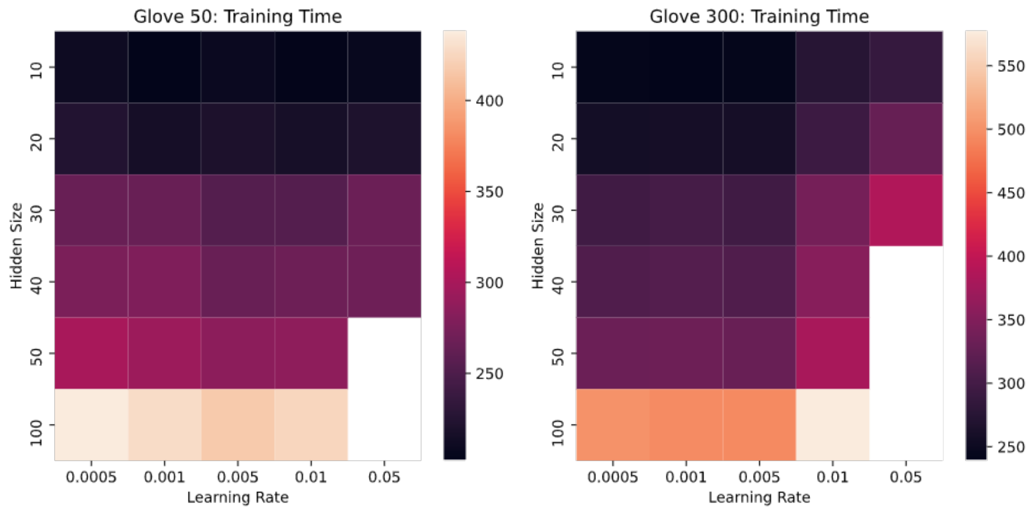


Figure 4: Glove 50 vs Glove 300: Training Time

Consistent with my initial expectations, accuracy and training time were generally higher when using the Glove 300 word vectors compared to using the Glove 50 word vectors (note the difference in the colour scale limits in the figures above). Given the emphasis on accuracy over computational time discussed previously, the Glove 300 vectors were used in the final model.

4.2 Metaparameters

The key metaparameters were:

- Epochs
- Learning rate
- Hidden size
- Drop out rates

Throughout the experimentation phase, epochs were limited to 5, as learning curves suggested that most of the gain in accuracy occurred in the first epoch with some small improvements up to around epoch 5, and limiting the number of epochs allowed for more rapid experimentation.

Initially, learning rate and hidden size metaparameters were selected using various targeted grid searches, similar to those shown in section 4.1. Hidden sizes were selected using the same value for both rating and category networks, and for multiple layers where relevant. A subsequent search around this space (e.g. first and second layer hidden sizes +/- 10 separately) was used to confirm that there was no significant gain to be had by adjusting these values further. I explored hidden sizes and learning rates together as I expected there might be some relationship between the two, although this was not as evident as I had expected (e.g. mainly vertical bands evident in the figures shown in section 4.1)

Minimal tuning was applied to drop out rates; these are discussed in more detail in section 5.

Having tuned the hyperparameters using this approach, the weighted score reached a plateau at around 86% (for various architectures). At this point, I decided to take my best performing model at the time (the RCNN) and go deep. I increased the hidden size from 64 to 100, increased the epochs from 5 to 40 and added 50% initial drop out (which was not present until this point). The intention was to make it more difficult for the model to learn (more parameters, more dropout) but also give it more time to learn. This approach led to the weighted score reaching the target level of 88%. However, run time increased from around 15 minutes to around 1.5 hours, which made it difficult to fine tune the model further.

4.3 Data preprocessing

To enable effective preprocessing, I created a pipeline which showed a selection of examples of misclassified data points, including the original text and the processed text that the model converted into word embeddings. This allowed me to compare my rating as a human with what the algorithm was producing and consider whether it was possible to make it more obvious to the model when I could see things it could not. Based on this approach, I added the following preprocessing steps:

- While emoji did not seem to be present in the data, there were emoticons such as :) and D: which I converted to either ”positive good” or ”negative bad” to try to signal the emotional content to the model. I used 8 of the most frequently used emoticons [7].
- Many reviewers would explain why they gave a particular rating (e.g.

"I gave this place two stars because...". For higher star numbers I converted to "positive good" while for lower star numbers I converted to "negative bad". While the model may have picked this up, the importance of such a feature seemed particularly high.

- I removed most punctuation and digits, to reduce noise in the training data, but retained ! ? , and . as these seemed to indictate emotion quite often. Negative reviews were often accompanied with a high amount of emotion, and hence more ! and ? in particular. Consecutive punction such as ... was converted to prevent e.g. "so...so...bad" from becoming "sosobad".
- I picked out a handful of phrases where I thought the word embeddings might not capture the meaning well, such as where there are several meanings (e.g. "sucks" could involve drinking, but within a review context tends to indicate negative sentiment) or where the word was clearly aligned to one of the business categories (e.g. "vehicle" probably indicates automotive). Given I only picked out a handful of phrases I think this was more of a proof of concept than something which impacted model accuracy significantly. In a production environment, such hand crafting of features might be worthwhile.

The model framework included the capacity to remove a list of "stop words", however I did not make use of this feature. The idea behind removing stop words is to make it easier for the model to pick up on the "signal" by removing low value words. Initially I did remove a list of stop words taken from NLTK; however, using my pipeline above, I realised phrases such as "not good" were becoming "good", changing the meaning entirely. At this point I decided to not use any stop words as it was difficult to predict what impact it might have on attention based language models.

5 Avoiding overfitting

When a model is assessed on the same data it has trained on, it can simply ”memorise the answers”, leading to high performance on the training data and a poor ability to generalise to unseen data. For this reason, a3main.py includes the ability to split the data into training and validation sets, so the model assessment is more ”fair”. However, in selecting metaparameters and creating preprocessing features, we run the risk of inadvertently overfitting to our validation set. This is particularly the case if random seeds are not carefully controlled, as the data used for training in one run may be used for validation in another. For this reason, I separated out 10% of the data as a fixed ”test set” which was used alongside the random validation set to assess model accuracy and better gauge the extent to which the model would generalise. A stratified split based on the target variables was used to ensure both sets contained the same proportions by rating and category.

Dropout layers encourage the network to learn more general patterns rather than relying on specific features within the training dataset (i.e. overfitting). Several forms of dropout were considered:

1. Dropout can be applied to the inputs before any other layers, reducing the relevance of memorising specific training examples.
2. Dropout layers can be introduced at any point in a network, forcing the network to ”share the load” between hidden features rather than specialising too much.
3. Dropout can be introduced into layers of a stacked LSTM or GRU, where the dropout applies to the output of layers besides the final layer.

The final RCNN architecture used both the first and second options. A drop out rate of 50% is used in both cases. This is the default within Pytorch and provides the greatest amount of regularization [8].

Learning curves were employed to get a sense of how much a model might be overfitting. When the training and validation performance diverges, our model is overfitting. Fig. 5 and Fig. 6 show examples of such learning curves, for models without dropout and with dropout respectively. When validation performance plateaus then further training will often lead to overfitting.

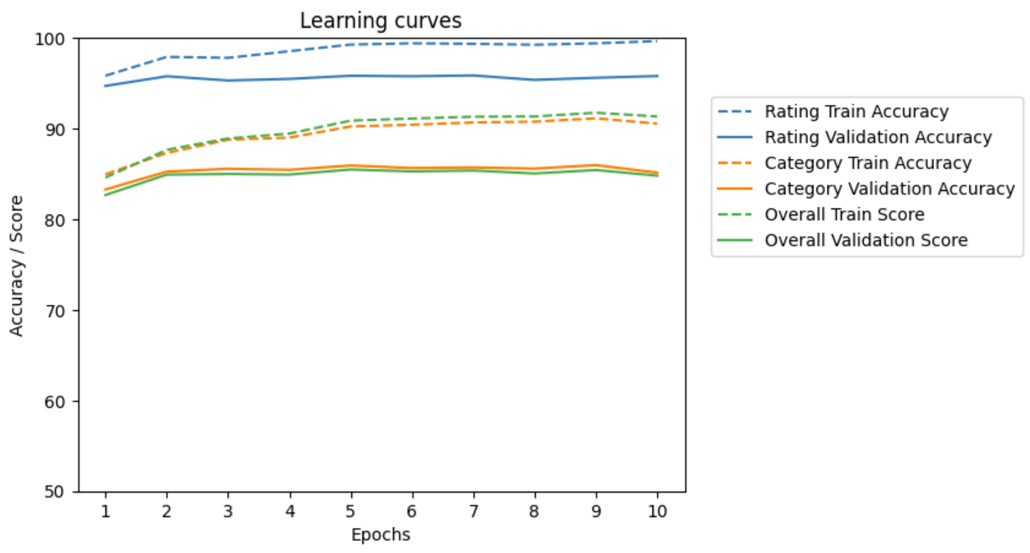


Figure 5: Learning curve - without dropout

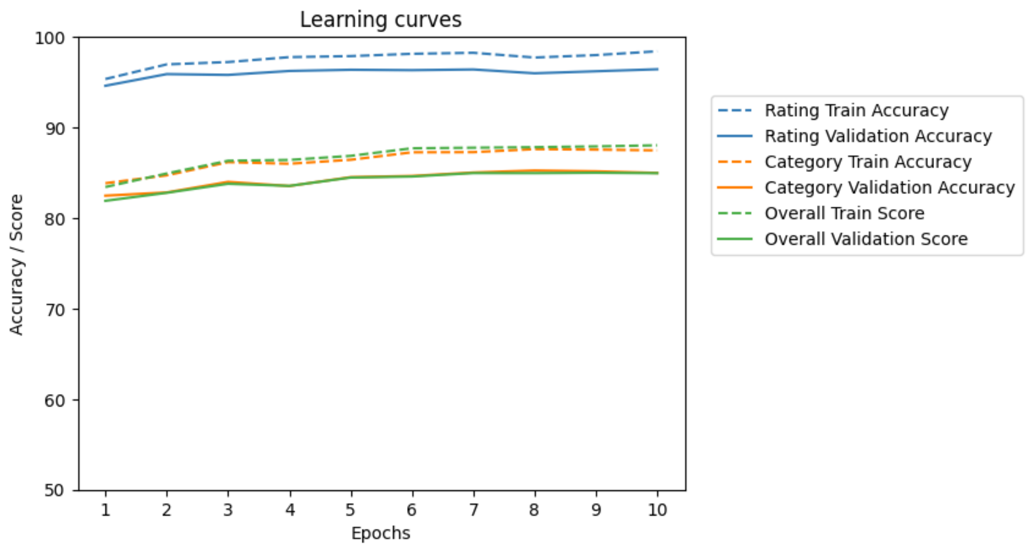


Figure 6: Learning curve - with dropout

Fig. 7 shows the learning curves for the final model. The overall validation curve continues to increase for at least 30 epochs, particularly for the category model, suggesting it is appropriate to continue to train the model for at least 30 epochs. Between 30 and 40 epochs the validation score plateaus, so there's no argument for extending it beyond 40 epochs, and possibly an argument for reducing the epochs from 40.

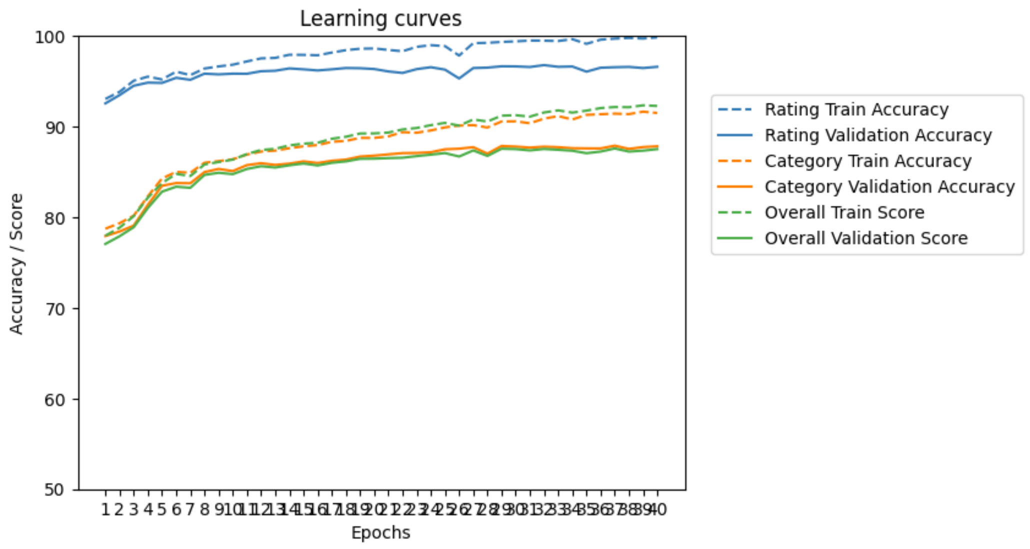


Figure 7: Learning curve - final model

References

[1] S. Lai, L. Xu, K. Liu, and J. Zhao, “Recurrent Convolutional Neural Networks for Text Classification,” Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, Jan. 2015, Accessed: Apr. 16, 2021. [Online]. Available: `https://www.aaai.org/ocs/index.php/AAAI/AAAI15/paper/view/9745/9552`.

[2] Wikipedia Contributors, “Rectifier (neural networks),” Wikipedia, Apr. 02, 2019. `https://en.wikipedia.org/wiki/Rectifier_(neural_networks)` (accessed Apr. 16, 2021).

[3] A. Burmistrov, ”Toxic Comment Classification Challenge — Kaggle”, Kaggle.com, 2018. [Online]. Available: `https://www.kaggle.com/c/jigsaw-toxic-comment-classification-challenge/discussion/52644`. [Accessed: 14- Apr- 2021].

[4] Torch Contributors, “BCEWithLogitsLoss — PyTorch 1.7.1 documentation,” pytorch.org, 2019. `https://pytorch.org/docs/stable/generated/torch.nn.BCEWithLogitsLoss.html` (accessed Apr. 16, 2021).

[5] K. Eckhardt, “Choosing the right Hyperparameters for a simple LSTM using Keras,” Medium, Nov. 29, 2018. `https://towardsdatascience.com/choosing-the-right-hyperparameters-for-a-simple-lstm-using-keras-f8e9ed76f046` (accessed Apr. 16, 2021).

[6] Vitaly Bushaev, “Adam—latest trends in deep learning optimization.,” Towards Data Science, Oct. 22, 2018. `https://towardsdatascience.com/adam-latest-trends-in-deep-learning-optimization-6be9a291375c` (accessed Apr. 16, 2021).

[7] H. Wang and J. A. Castanon, “Sentiment expression via emoticons on social media,” IEEE Xplore, Oct. 01, 2015. `https://ieeexplore.ieee.org/abstract/document/7364034` (accessed Dec. 10, 2020).

[8] P. Baldi and P. Sadowski, “Understanding Dropout,” 2013. Accessed: Apr. 15, 2021. [Online]. Available: `https://proceedings.neurips.cc/paper/2013/file/71f6278d140af599e06ad9bf1ba03cb0-Paper.pdf`.