(Part A – Reinforcement Learning)

An Empirical Evaluation of Deep Q-Learning against Double Q-Learning

Matt Goodhead

School of Science, Engineering and Environment

University Of Salford

Introduction

S Vadera’s lecture slides [2] describe Deep-Q learning as an extension of Q-Learning where the Q function is estimated by using a Neural network. The Q value for an action represents how likely taking that action in the current state is to get you closer to getting a higher reward. For example, if moving down got the player closer to the goal then it would have a higher Q value. Or, if moving up in the current state ended the game then the action of moving up would have a very low Q value. Standard Q-Learning can be applied when there’s a known amount of states but Deep Q-Learning is needed when there’s an unknown amount of states. This is because if there’s an unknown amount of states then the Q function can’t be calculated mathematically so it has to be estimated.

In Hado Van Hesselt’s 2015 paper on deep reinforcement learning [3] he observed that the q_net_target is used both to estimate the initial Q values and for computing the target values. Hado Van Hessell noticed this and proposed use of the q_net to choose the action and the q_net_target when the next state is being evaluated. This change to the Deep-Q Learning algorithm is more widely known as Double Deep-Q Learning. In this paper I will be comparing Deep-Q learning to Double-Q Learning using the Gym python module created by OpenAI [4].

I decided to use the “MountainCar-v0” environment to compare Deep-Q, and Double Deep-Q Learning. OpenAI describes this model as a car on a one-dimensional track, positioned between two “mountains ” [4]. The goal is to drive up the mountain on the right; however, the car’s engine is not strong enough to scale the mountain in a single pass. Therefore, the only way to succeed is to drive back and forth to build up momentum. I decided to choose this environment because it would take a lot of episodes of training in order to be completed. Which means more data to compare the speed of Deep-Q and Double Deep-Q learning on.

Hardware

I decided to use Google Colab to train the reinforcement learning models. This is because Google Colab offers TPU (Tensor Processing unit) pools. Reinforcement learning is extremely slow so I wanted to make sure I had as much processing as I could available to speed up the research process. Normally when you use a TPU with a neural network you have to instantiate the neural network within the scope of the TPUs strategy. However, with keras-rl2 the DeepQ-Learning agent is designed to automatically work with pools of hardware without needing any additional code.

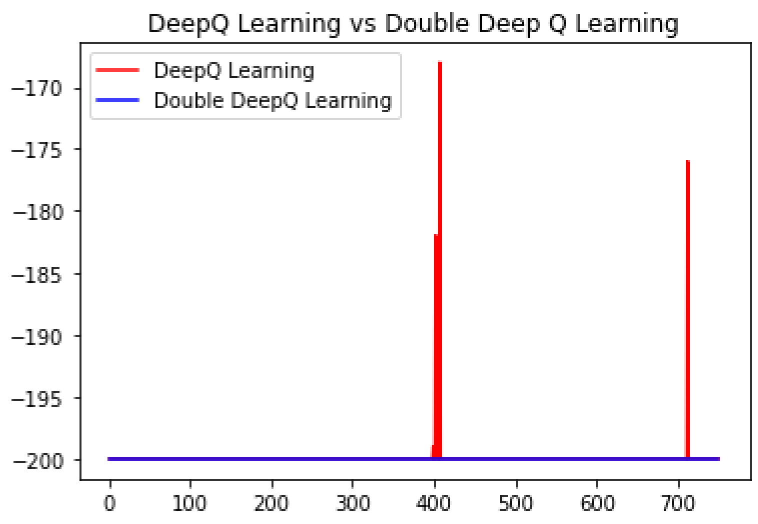
Model Design

I created two identical neural networks consisting of a flatten layer, 3 dense layers stacked on top of each other, and then a Dense layer which had a node for every possible action the neural network could take. I then connected both models to a Deep-Q Learning Network Agent. One agent had Double Deep-Q Learning disabled, and the other had it enabled. I did this so I could see the difference between what happens when you train the networks both with and without Double Deep-Q learning.

Steps Taken

I originally decided to train both networks for 150,000 steps which would have been 750 episodes of 200 steps. You can see the result of this training in Figure 1. During this training session the Deep-Q learning model managed to climb the hill twice, and the Double Deep-Q Learning model did not manage to climb the hill at all. After analysying this graph I decided that the model was under-trained so I decided to restart the training with double the amount of steps. This resulted in doing 300,000 steps per model which is equivalent to 1,500 episodes of 200 steps.

Figure 1 – Model Comparison without enough training



Review & Discussion

Quantitative Results

I trained two Deep-Q Learning models for the “MountainCar-v0” environment. The models were identical other than one using Deep-Q Learning and the other using Double Deep-Q Learning. Figure 2 shows the rewards during training for the Deep-Q model over 1,500 episodes of training. Figure 3 shows the rewards during training for the Double Deep-Q model over 1,500 episodes of training. Figure 4 shows both Figure 2 and Figure 3 plotted on the same graph for comparison.

Qualitative Analysis

Out of both Figure 2 and Figure 3, the first model to get a reward higher than –200 was the DeepQ Learning model after roughly 750 episodes. However, after this neither network managed to get a reward higher than –200 for more than 200 episodes. Due to this, I think that the first rise in reward on the Deep-Q Learning model was an anomaly, and I have decided not to take it into account when evaluating the results of the experiment.

At roughly 950 episodes into the training session both models started to finish climbing the hill. At this point in the training the Deep-Q learning model was managing to complete the task more often than the than the Double Deep-Q Learning. However, as the training continued the Double Deep-Q Learning model started to successfully climb the mountain more successfully than the Deep-Q Learning model. You can see this by comparing the gaps in Figures 2 and 3. Towards the end of the two figures you can see the gaps in the Double Deep-Q Learning graph are smaller than the gaps in the Deep-Q Learning graph.

Figure 5 & 6 shows the time taken per episode during the training process. The Deep-Q Learning model took roughly 1.8 seconds per episode whereas the Double Deep-Q Learning took rough 2 seconds per episode. This small (0.2 second) difference in time might not seem like much but that is the time difference per episode rather than for the entire training session. As there is 1,500 episodes total that means that 300 seconds or 5 minutes of extra time was needed to train the Double Deep-Q model.

Conclusion

My research has shown that Deep-Q Learning can learn faster than Double Deep-Q Learning initially but over more training Double Deep-Q Learning will eventually become more reliable. This supports Hado Van Hasselts observation that Q-Learning is more likely to overestimate Q-Values than Double Q-Learning. This is because, overestimating the Q-Values would cause the model to be less reliable after lots of training compared to the Double Deep-Q Learning model. My research has also shown that training a Double Deep-Q model takes longer than training a Deep-Q model. For me, the impact of this was only 5 minutes but for someone not using a pool of hardware this difference may be more significant and a deciding factor between the two models.

Figure 2 – DeepQ Learning Reward Graph

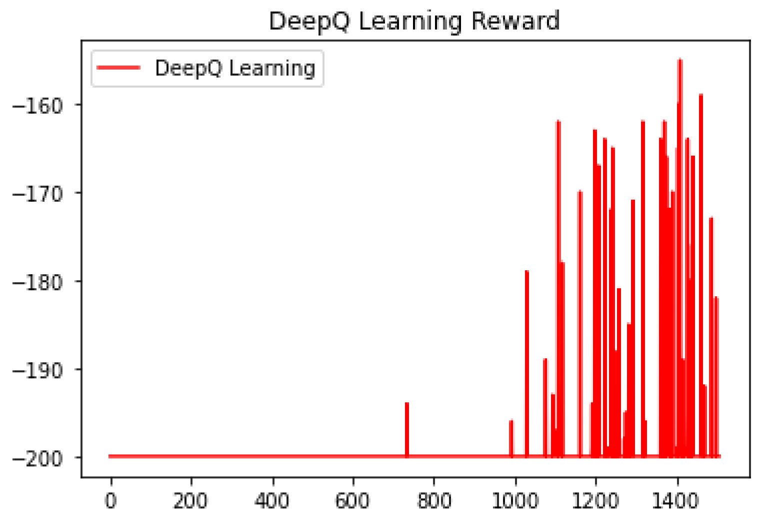


Figure 3 – Double DeepQ Learning reward graph

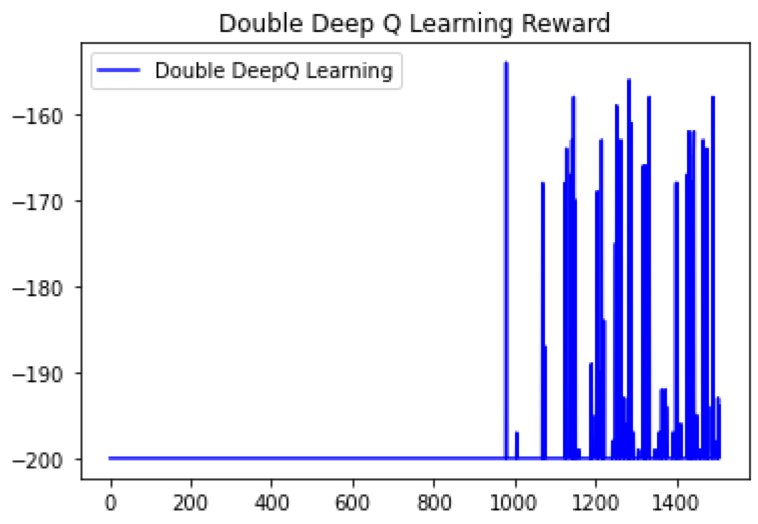


Figure 4 – DeepQ Learning and Double Deep Q Learning rewards plotted on the same graph.

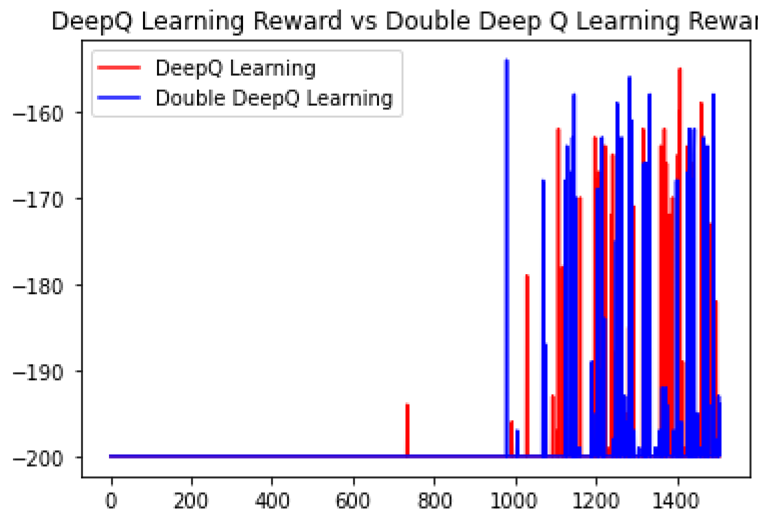


Figure 5 – Time taken for Deep-Q Learning

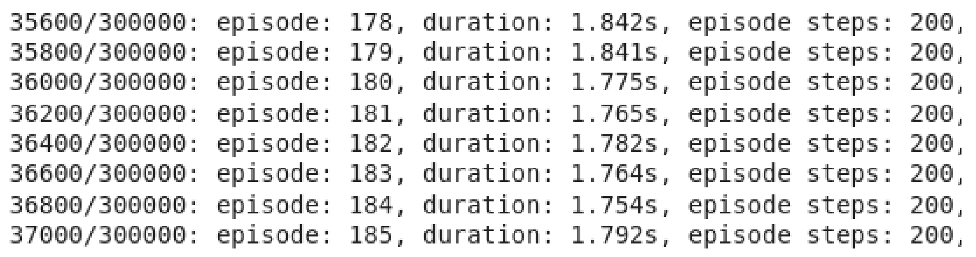
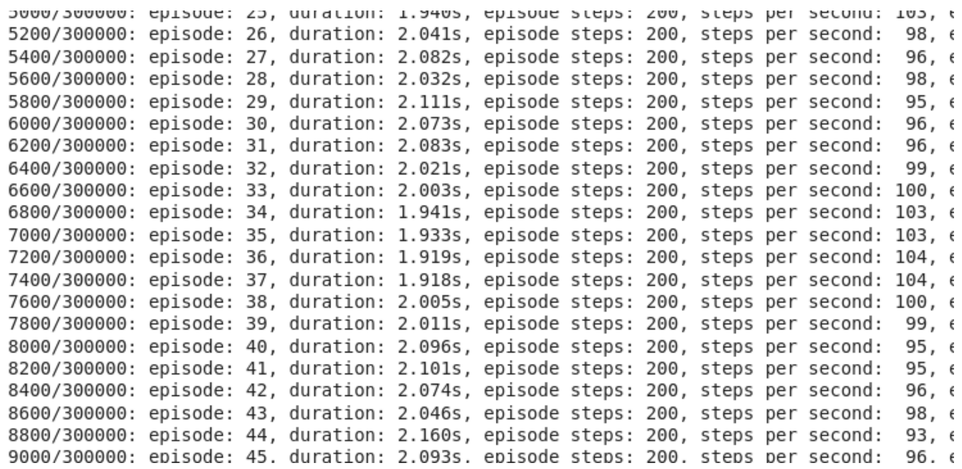

```
35600/300000: episode: 178, duration: 1.842s, episode steps: 200,
35800/300000: episode: 179, duration: 1.841s, episode steps: 200,
36000/300000: episode: 180, duration: 1.775s, episode steps: 200,
36200/300000: episode: 181, duration: 1.765s, episode steps: 200,
36400/300000: episode: 182, duration: 1.782s, episode steps: 200,
36600/300000: episode: 183, duration: 1.764s, episode steps: 200,
36800/300000: episode: 184, duration: 1.754s, episode steps: 200,
37000/300000: episode: 185, duration: 1.792s, episode steps: 200,
```

Figure 6 – Time taken for Double Deep-Q Learning



```
5000/300000: episode: 25, duration: 1.940s, episode steps: 200, steps per second: 103,
5200/300000: episode: 26, duration: 2.041s, episode steps: 200, steps per second:  98,
5400/300000: episode: 27, duration: 2.082s, episode steps: 200, steps per second:  96,
5600/300000: episode: 28, duration: 2.032s, episode steps: 200, steps per second:  98,
5800/300000: episode: 29, duration: 2.111s, episode steps: 200, steps per second:  95,
6000/300000: episode: 30, duration: 2.073s, episode steps: 200, steps per second:  96,
6200/300000: episode: 31, duration: 2.083s, episode steps: 200, steps per second:  96,
6400/300000: episode: 32, duration: 2.021s, episode steps: 200, steps per second:  99,
6600/300000: episode: 33, duration: 2.003s, episode steps: 200, steps per second: 100,
6800/300000: episode: 34, duration: 1.941s, episode steps: 200, steps per second: 103,
7000/300000: episode: 35, duration: 1.933s, episode steps: 200, steps per second: 103,
7200/300000: episode: 36, duration: 1.919s, episode steps: 200, steps per second: 104,
7400/300000: episode: 37, duration: 1.918s, episode steps: 200, steps per second: 104,
7600/300000: episode: 38, duration: 2.005s, episode steps: 200, steps per second: 100,
7800/300000: episode: 39, duration: 2.011s, episode steps: 200, steps per second:  99,
8000/300000: episode: 40, duration: 2.096s, episode steps: 200, steps per second:  95,
8200/300000: episode: 41, duration: 2.101s, episode steps: 200, steps per second:  95,
8400/300000: episode: 42, duration: 2.074s, episode steps: 200, steps per second:  96,
8600/300000: episode: 43, duration: 2.046s, episode steps: 200, steps per second:  98,
8800/300000: episode: 44, duration: 2.160s, episode steps: 200, steps per second:  93,
9000/300000: episode: 45, duration: 2.093s, episode steps: 200, steps per second:  96,
```

[1] : Moore, A. W. (1990). Efficient memory-based learning for robot control. [Accessed 9/5/2020]

[2]: Vadera, S. & University of Salford. (2021). *22 – Deep Q Learning* [Slides]. Salford Blackboard. https://blackboard.salford.ac.uk/bbcswebdav/pid-5363071-dt-content-rid-25219919_1/xid-25219919_1 [Accessed 9/5/2020]

[3]: van Hasselt, H., & Sutton, R. S. (2015). Learning to predict independent of span. *arXiv preprint arXiv:1508.04582*. [Accessed 9/5/2020]

[4]: Gym: A toolkit for developing and comparing reinforcement learning algorithms. (2021). Retrieved 9 May 2021, from https://gym.openai.com/envs/MountainCar-v0/ [Accessed 9/5/2020]

[5]: *Google Colaboratory TPU Guide*. (n.d.). Google Colab. https://colab.research.google.com/notebooks/tpu.ipynb [Accessed 9/5/2020]

(Part B: Sequence Learning)

An Empirical Comparison of LSTM and GRU cells for classification models on the TripAdvisor Hotels dataset

Matt Goodhead

School of Science, Engineering and Environment

University Of Salford

Abstract

GRU and LSTM cells can be used to improve NLP (Natural Language Processing) models by helping them remember the words before it in the sentence, so a model can understand the context of a word in the sentence. My comparison shows the steps taken to use both types of cells to classify TripAdvisor reviews. After that, I discuss the pros and cons of both GRU and LSTM cells, and which should be used to improve the performance of NLP classification models.

Introduction

According to research in 2017 [1], TripAdvisor first emerged in 2004 as a web application for the tourism domain. The website is populated by user-generated reviews of hotels, restaurants, and tourism spots. TripAdvisor also allows users to walk around Google Maps and view the TripAdvisor ratings of nearby businesses (this can be seen in Figure 1. In 2017 TripAdvisor was ranked as the most popular site for trip planning, with millions of tourists visiting the site when arranging their holidays. However, Similar Webs site rankings for Travel and Tourism show that TripAdvisor has more recently been overtaken by Booking.com [2].

Figure 1 – 3D Visualization of TripAdvisor Reviews in Granada (Paseo de los Tristes). In 2017 TripAdvisor was the most popular site for planning a trip. [1]

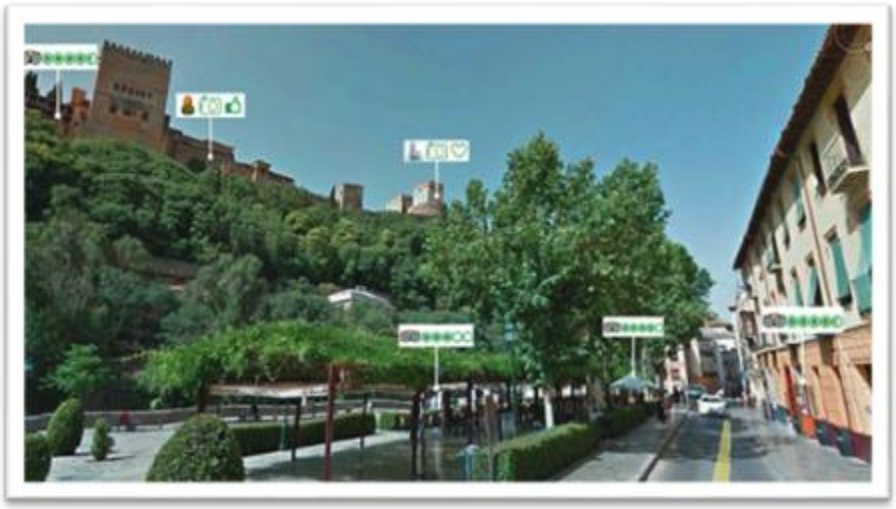
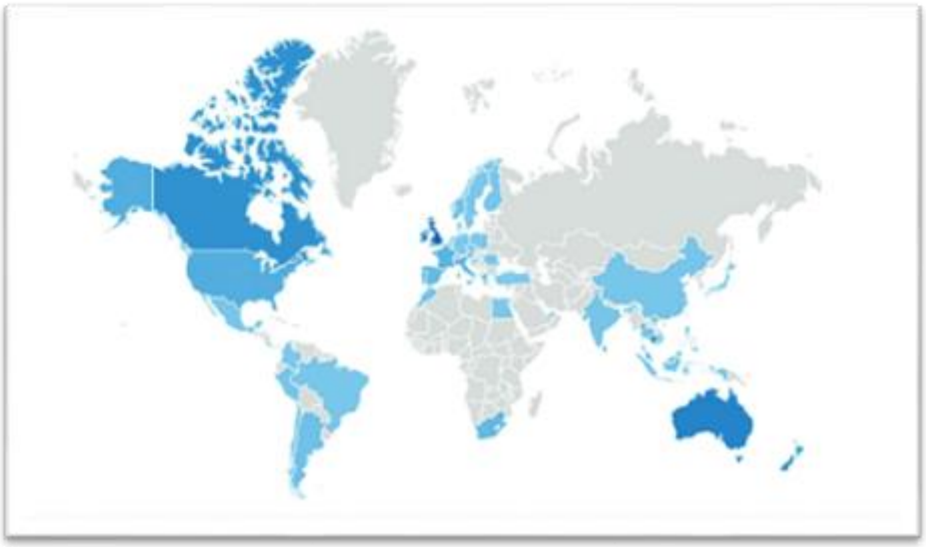


Figure 2 – Map of the popularity of TripAdvisor searches on Google between 2012-2017[1]

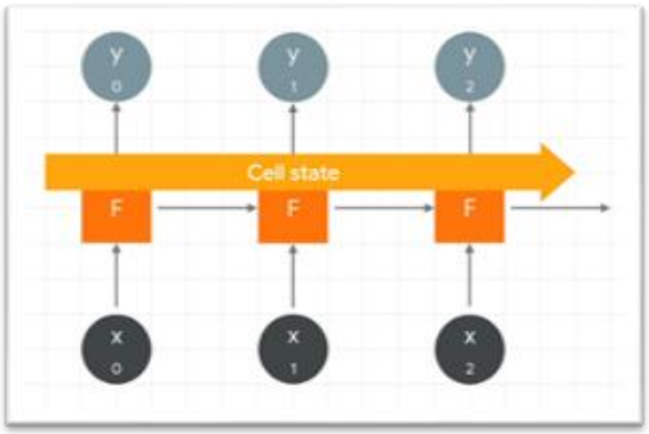


A study 2021 study [3] shows that the reliability of hospitaliy reviews can be biased by user-given-ratings. This means that fake reviews can made to improve/decrease a potential customers opinion of a business. To maintain their integrity companies such as TripAdvisor need to combat biased/fake reviews. However, TripAdvisor receives thousands of reviews every day from across the world, this is too many reviews to have each review individually moderated. Reviews can be flagged and reported for moderation. However, this still leaves many biased reviews ignored.

As shown in the 2021 study [3], biased user-given-ratings can be identified by predicting the rating from the textual content of the review. The study concluded that unreliable reviews can be detected using deep learning models to predict the rating and then comparing the predicted rating to the users rating. The models from this paper were successful at finding reviews that may cause inconsistency and ambiguity to readers with a high accuracy. Similar models could be used to warn users of biased reviews to improve their experience on hospitality review sites such as TripAdvisor.

An important part of learning the meaning of a review is the context of a word in a sentence. In part five of TensorFlow's “NLP Zero to hero” video series on YouTube [4], it is explained that in a recurrent neural network a neuron can learn something and then pass on context to the next time step. The problem with this is that over a long distance the context can be deluded or forgotten, this means that the network might not be able to see how meaning in words that are far apart impact overall meaning. The LSTM architecture (diagram in Figure 3) is the solution to this problem. The LSTM gate has a cell state which is a context that can be maintained across many time steps. As a result of this it can understand links between the beginning and the end of a sentence. An extension of this is to have the cell state passed in both directions, this is called a Bidirectional layer.

Figure 3 – LSTM Architecture [4]



A less performance intensive alternative to an LSTM is a GRU. S Vadera's slides [6] describe how an LSTM is a cell made of a forgetting gate, an updating gate, and a cell output gate. Every time-step the hidden state and cell state is passed on allowing context to be remembered. The GRU cell architecture merges the cell state and hidden state, as well as combining the forget and input gates into a single reset gate. In this paper I use the TripAdvisor rating prediction as a benchmark to compare the performance of the LSTM cell and the GRU cell for predicting the review score from the review text. This analysis will help improve the models used to find biased hospitality reviews.

Dataset Preprocessing

A csv file containing two columns (review text, and rating) was provided by S Vadera for this comparison [7]. This dataset contained 20,491 reviews, and the longest review contained 2,275 words. I parsed this dataset using the "read_csv" function from the panda's library [8]. An immediate problem I noticed with this dataset was that it contained punctuation. This meant that a tokenizer would think that "Hello" and "Hello." were two completely different words. However, I did want to remove punctuation as it is extra information to use to understand the context of the reviews. For example, a review with lots of exclamation marks (!) is extremely passionate so it is more likely to be a 0-star rating or a 5-star rating. To solve this problem, I used regex to make each punctuation mark its own word. For example, "I love bananas!" would be changed to "I love bananas !". With this improvement punctuation will now be treated as a word by the tokenizer so it will help add more context rather than removing it like it previously did. This problem is also described in a solution on the Stack Overflow website [10].

Another step I took to preprocess the data was to add start and end tokens to the start of each sentence, I found this technique from an article on the TensorFlow website [9]. This addition helps the network find the context of a word depending on whether it is near the start of the end of a review. To implement this, I added the word token "<start>" to the start of every review and added the word token "<end>" to the end of every review. I then applied the preprocessing to every review in the dataset using a list comprehension [11]. After this I converted the review texts to numerical sequences by using a tokenizer and then padded the sequences so they were the same length and could be inputted into a neural network. To do this I used the Tokenizer class and the "pad_sequences" function from the TensorFlow machine learning framework [12].

Word Embeddings

To get the best results possible for all the networks in this comparison I decided to use the GloVe word embeddings [13]. GloVe is a dataset of vector space representations of words. The higher the similarity between two words is the smaller the cosine difference will be between the word vectors. Using GloVe helps the neural network to create relationships between the meaning of a word and the TripAdvisor score. To implement this, I imported the glove embeddings and then extracted the word vector for each word that is in the dataset and stored them in a matrix which I passed to the embedding layer for it to use as its initial weights.

Neural Network Design

For this comparison I created 4 different models. The models are almost identical other than the type of long-term-short-term-memory cell used. I decided to test with a LSTM, a Bidirectional LSTM, a GRU, and a Bidirectional GRU. The first layer of the networks was an embedding layer which used the 300d GloVe weights previously mentioned. Then I added, two of the type of cell I was testing stacked together. To stack them together I had to set "return_sequences" to true so the sequences would be passed on to the next layer. I then added a "GlobalMaxPool1D" layer, then a Dropout layer followed by a Dense layer with 50 nodes followed by another Dropout layer, followed by a Dense layer with 6 nodes. I then compiled the network using the "sparse_categorical_crossentropy" loss function and used the Adam optimizer.

Why sparse_categorical_crossentropy?

Sparse categorical cross entropy is for when the classes are mutually exclusive (e.g. when each sample belongs to exactly to one class), and categorical cross entropy is for when one sample can have multiple classes or for when the labels are soft probabilities (like [0.5, 0.3, 0.2]) [14]. In the case of TripAdvisor reviews the ratings are mutually exclusive because each review only has a single rating. Another difference is that for categorical cross entropy the output of the network is a one-hot encoding, but for sparse categorical cross entropy the output of the network is just a single integer. However, if I eventually wanted to train the network further, categorical cross entropy could be useful as it outputs the percentage chance of the review being each rating. For example, I could look at the percentages for the reviews that are not correctly classified and find out what type of reviews the network needed more training on.

Adaptive Learning Rate

As the gradient descent algorithm gets closer to the weights needed to correctly classify the reviews training can stagnate because the learning rate is too high to get to the needed values [15]. Imagine you were walking towards a small target on the floor. Initially you would want to take big steps to get close to the target but then when you got close you would want to reduce your step size, so you do not step over the target. An adaptive learning rate does this by slowly reducing the learning rate during the training process. I implemented this for all the models by using an Exponential Decay curve as a learning rate schedule which I then passed to the Adam optimizer.

Training Hardware

Training the networks on my own computer was possible but extremely slow, as I had to use a low batch size due to video memory limits. To vastly speed up the training of the network I used a TPU pool (tensor processing unit) from Google Colab. A TPU pool is a pool of lots of different CPUs and TPUs working together. To use a pool with TensorFlow you must define a strategy which tells TensorFlow how to interact with all the pooled hardware. Google Colab has a guide on how to set this up [16].

Results & Discussion

Quantitative Results

After training all four networks for 20 epochs I plotted the accuracy and validation accuracy of all four networks onto the same graph so they can be compared. Figure 4 shows the accuracy of all four networks. Figure 5 shows the validation accuracy of all four networks. Figure 6 shows an epoch of training for the bidirectional LSTM model. Figure 7 shows an epoch of training for the bidirectional GRU model. I took a screenshot of the epochs so I can compare the time taken per epoch for the networks.

Qualitative Analysis

By comparing the accuracy graph (Figure 4) to the validation accuracy graph (Figure 5) we can see that the accuracy for all the models continues to increase past epoch 10 even though the validation accuracy does not. This is called overtraining the neural network is starting to remember rating for each sentence rather than learning how to determine the ratings. We can also see that the Bidirectional GRU overtrains faster than the Bidirectional LSTM, and the GRU network overtrains faster than the LSTM network. By examining this we can see that GRUs overtrain faster than LSTMs. We can also see that Bidirectional cells overtrain faster than regular cells.

By evaluating Figure 5 the performance of each of the 4 networks can be found. The GRU network had the worst validation accuracy (roughly 62% at the highest point). Whereas the LSTM network managed to reach a higher validation accuracy (roughly 65% at the highest point). Both Bidirectional LSTM and GRU networks managed to get roughly the same validation accuracy as the LSTM network. However, the Bidirectional networks are more complex and took almost double the amount of time to train. For this task a none-bidirectional LSTM is the best choice if we don’t care about how long the network takes to train.

However, the time and resources taken to train the network is an important factor then a GRU would best option. By comparing figures 6 & 7 we can see that the time taken to finish an epoch with the Bidirectional GRU (7 seconds) is almost half the time taken to complete an epoch with the Bidirectional LSTM (13 seconds). For my experiment the time taken to complete an epoch didn’t really matter as I was using a TPU pool. However, if someone had hardware constraints then a GRU may be the best option.

Conclusion

In this work, I analyzed the performance of LSTM and GRU cells and compared their accuracy, validation accuracy, and time taken to perform an epoch. From the qualitive analysis both LSTMs and GRUs are good for different tasks. If the goal is to minimize hardware use or to train the network as fast as possible then a GRU would be an excellent choice. However, if the goal were to get the best validation accuracy possible then an LSTM would be the best choice. Finally, I can also conclude that making the LSTM/GRU cells bidirectional did not improve validation accuracy and overtrained faster than the regular GRU/LSTM cells making them the worst choice for this job.

Figure 4 – Training Accuracy for all 4 networks

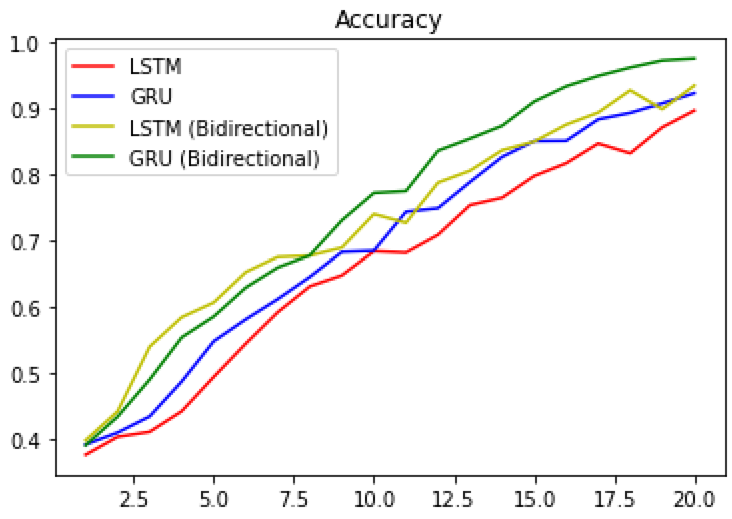


Figure 5 – Validation accuracy for all 4 networks

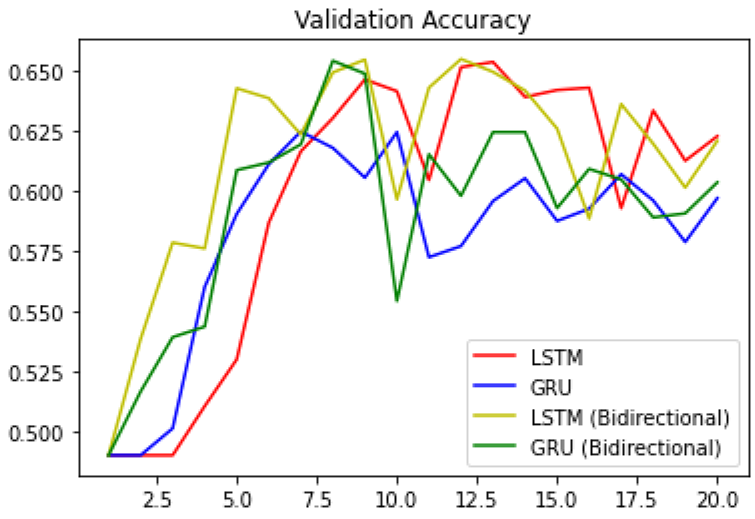
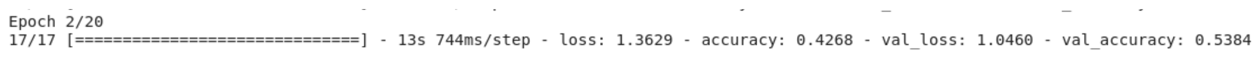
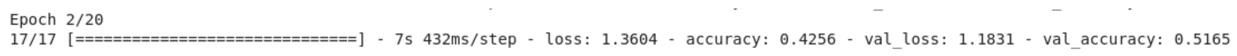


Figure 6 – Epoch from Bidirectional LSTM Model



```
Epoch 2/20
17/17 [==============================] - 13s 744ms/step - loss: 1.3629 - accuracy: 0.4268 - val_loss: 1.0460 - val_accuracy: 0.5384
```

Figure 7 – Epoch from the Bidirectional GRU Model



```
Epoch 2/20
17/17 [==============================] - 7s 432ms/step - loss: 1.3604 - accuracy: 0.4256 - val_loss: 1.1831 - val_accuracy: 0.5165
```

[1]: Valdivia, A., Luzón, M. V., & Herrera, F. (2017). Sentiment analysis in tripadvisor. *IEEE Intelligent Systems*, *32*(4), 72-77. (from https://w.sentic.net/sentiment-analysis-in-tripadvisor.pdf) [Accessed 8th May 2020 12:06]

[2]: SimilarWeb. (n.d.). *Top sites ranking for Travel And Tourism in the world*. Retrieved 8 May 2021, from https://www.similarweb.com/top-websites/category/travel-and-tourism/

[3]: Zheng, T., Wu, F., Law, R., Qiu, Q., & Wu, R. (2021). Identifying unreliable online hospitality reviews with biased user-given ratings: A deep learning forecasting approach. *International Journal of Hospitality Management*, *92*, 102658. [Accessed 8th May 2020 12:47]

[4]: TensorFlow. (2020, May 7). *Long Short-Term Memory for NLP (NLP Zero to Hero - Part 5)*. YouTube. https://www.youtube.com/watch?v=A9QVYOBjZdY [Accessed 8th May 2020 at 13:02]

[5]: Gruber, N., & Jockisch, A. (2020). Are GRU cells more specific and LSTM cells more sensitive in motive classification of text?. *Frontiers in Artificial Intelligence*, *3*(40), 1-6. [Accessed 8th May 2020 13:12]

[6]: Vadera, S. & University of Salford. (2021). *16 - Long Short Term Memory Networks (LSTMs)* [Slides]. Salford Blackboard. https://blackboard.salford.ac.uk/bbcswebdav/pid-5145907-dt-content-rid-21025712_1/xid-21025712_1

[7]: TripAdvisor, & Vadera, S. (2021). *TripAdvisor Hotel Reviews* [Dataset]. Sunil Vadera for use in the Deep Learning Module at Salford. https://blackboard.salford.ac.uk/bbcswebdav/pid-5342678-dt-content-rid-24732019_1/xid-24732019_1

[8]: P. (n.d.). *pandas-dev/pandas*. GitHub. https://github.com/pandas-dev/pandas

[9]: *Neural machine translation with attention | TensorFlow Core*. (n.d.). TensorFlow. https://www.tensorflow.org/tutorials/text/nmt_with_attention#write_the_encoder_and_decoder_model

[10]: *python: padding punctuation with white spaces (keeping punctuation)*. (2010, September 5). Stack Overflow. https://stackoverflow.com/questions/3645931/python-padding-punctuation-with-white-spaces-keeping-punctuation

[11]: Oliphant, T. E. (2007). Python for scientific computing. *Computing in Science & Engineering*, *9*(3), 10-20.

[12]: T. (n.d.-b). *tensorflow/tensorflow*. GitHub. https://github.com/tensorflow/tensorflow

[13]: Pennington, J., Socher, R., & Manning, C. D. (2014, October). Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)* (pp. 1532-1543).

[14]: *Sparse_categorical_crossentropy vs categorical_crossentropy (keras, accuracy)*. (2018, December 1). Data Science Stack Exchange. https://datascience.stackexchange.com/questions/41921/sparse-categorical-crossentropy-vs-categorical-crossentropy-keras-accuracy

[15]: Zeiler, M. D. (2012). Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*.

[16]: *Google Colaboratory TPU Guide*. (n.d.). Google Colab. https://colab.research.google.com/notebooks/tpu.ipynb