1. Motivation

Scenario: A startup company wants to design a subscription based website for movie buffs. The site allows users to host watch parties for their favorite movies. The company would like to work on their recommender system, and they need a way to extract a numerical rating from a written review of a movie. You are a Data Scientist and are writing a proposal to work on this task.

The success of streaming services such as Netflix, Paramount Plus, Hulu, and Youtube TV have transformed the entertainment industry forever. An integral part of the success of these services is their recommendation systems. The machine learning algorithms behind recommendation systems will always benefit from more information. A potential problem is the lack of data generated by a user of a streaming service. A user may frequently re-watch their favorite programs, and this type of behavior would not inform a recommendation system well at all. A clear solution would be access to more high-quality data to train a recommendation system to fit a particular user. Finding high-quality training data is a challenge due to the scarcity of labeled data. The internet contains a plethora of unlabeled data, however this unlabeled data would be useless to a recommendation system without a sentiment attached to it. Additionally, many sentiment analysis methodologies attach very basic sentiments to unlabeled data, such as "positive" or "negative". Ideally streaming services could have a sentiment analysis algorithm that attached meaningful numeric sentiment to unlabeled data, the goal of this project is to satisfy that ideal. The report and codebase is meant to be a mock proposal for the job posting scenario.

2. Problem Definition & Background Information

Problem Statement (Regression): How accurately can a regression model predict a numerical rating from a textual movie review on a scale from 1-10?

Problem Statement (Multi-Class Classification): How accurately can a model classify a numerical rating from a textual movie review on a scale from 1-10?

Why was it worthwhile to solve classification and regression problems?
It is not obvious that the difference between a rating of "1" and a rating of "2" is the same difference between a rating of "7" and a rating of "8". This ambiguity suggests that perhaps a classification solution would be more appropriate than a regression solution. However, the lack of ratings for "5" and "6" is cause for concern for a classification solution. Since the model will not be introduced to any mid-range values ("5" or "6") it will not be possible to classify mid-range values for a classification solution. One more component to consider is that simply because movie reviewers are unlikely to give a mid-range rating, that does not mean that a given textual review does not convey a mid-range rating. In other words, the bias against mid-range ratings is not necessarily accurate to the true sentiment of a review *See Figure 1*. These ideas were the impetus to study the numerical ratings from a classification and a regression perspective.

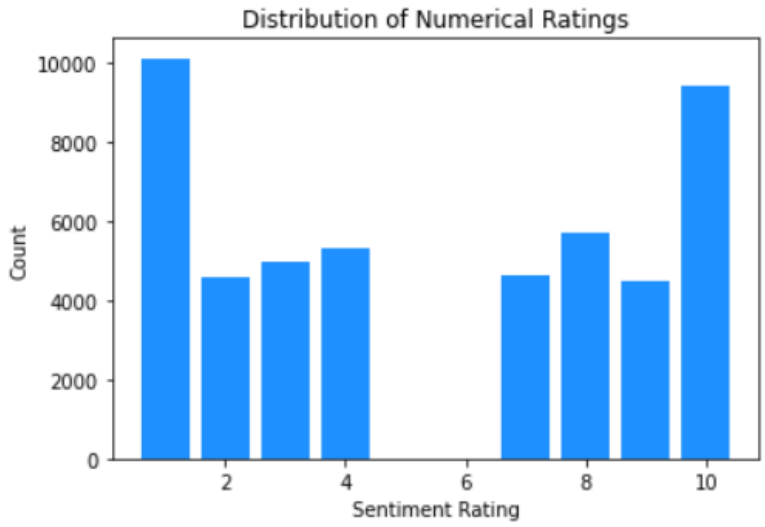

Figure 1: This histogram shows the distribution of numerical sentiment ratings in the dataset. The plot implies a slight bias to more extreme ratings (either positive or negative). We also can see that there are no numerical sentiment ratings that are five or six. The lack of borderline cases ("5" or "6") is reason enough to study the ratings from a regression and classification perspective.

3. Methodology: Description of the solution/baselines and the dataset

Dataset Description:

Data Source: The source of this data was the Stanford AI Lab. The data as well as the documentation can be found here. The title identification values correlated perfectly with the IMDb Movies dataset, located here. The joining of these two data sources allowed access to the meta-information for features like the genre of the film, or the start year, in correspondence with the textual review. The Stanford AI Lab dataset contains 25,000 training and validation movie reviews. Each set is perfectly balanced with 12,500 positive and negative reviews. A negative review is defined as having a rating from 1-5, and a positive review has a rating from 6-10.

	ReviewID	titleid	titletype	primarytitle	originaltitle	isadult	startyear	endyear	runtimeminutes	genres	Review	Score
0	0	tt0406816	movie	The Guardian	The Guardian	False	2006	NaN	139.0	["'Action'", "'Adventure'", "'Drama'"]	I went and saw this movie last night after bei...	10

Figure 2: This figure shows a sample row from the validation set with positive scores. This dataframe was the result of joining the two data sources mentioned in the Data Source description above.

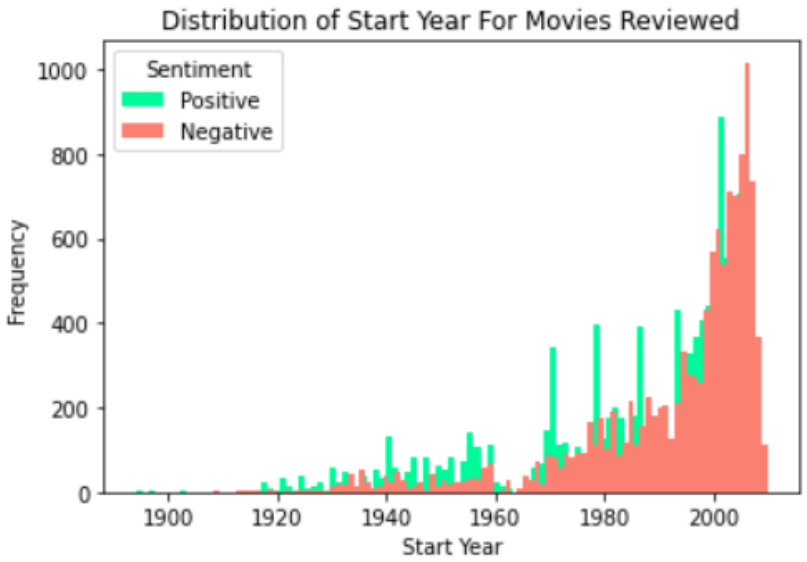

Figure 3: This histogram depicts the distribution of start years for positively and negatively rated movies. A negatively rated movie review has a rating score from 1-5. A positively rated movie review has a rating score from 6-10. There seems to be a slight bias for negative ratings to be for newer movies, however a two proportion Z-test would be required to prove this.

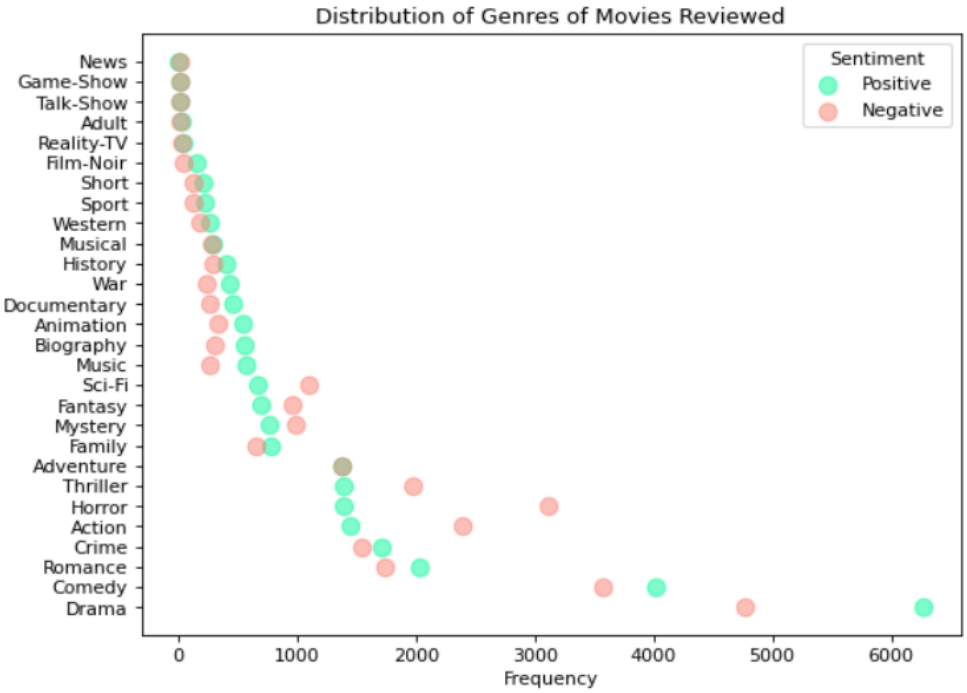

Figure 4: This plot shows the frequencies of genres in the reviews dataset for positive and negative reviews. There are some insights we may extract from this plot. It seems that Horror films are more likely to be negatively rated, and Drama movies are more likely to be positively rated.

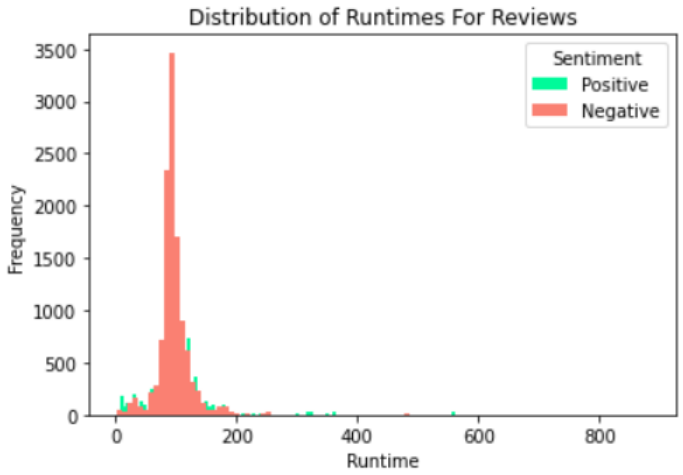

Figure 5: This figure shows the distribution of movie runtimes for positively and negatively rated movies. The distributions look almost identical for positive and negative ratings.

Baseline / Solution Description (Regression):

Optimizer: The optimizer for all of the solutions outlined for the regression task is the Adam optimizer. The Adam optimizer can act as a replacement for the Stochastic Gradient Descent optimizer. Optimization techniques are used to update network weights based on the training data. Some of the benefits of Adam optimization are that it is straightforward to implement, computationally efficient, low memory requirements, appropriate for tasks with large amounts of data or many parameters.

Loss Function: The loss function for all regression tasks is the MSE (Mean Squared Error) Loss. This loss function is defined below.

$$\text{MSE}_{\text{train}} = \frac{1}{N}\sum_{i=1}^{N}(\hat{y}^{(i)} - y^{(i)})^2$$

N: The number of observations

$\hat{y}^{(i)}$**:** The predicted rating score

$y^{(i)}$: The actual rating score

Multi-layer Perceptrons (MLP): MLP models are the simplest form of neural networks. They are used to theoretically approximate any function. The MLP for this solution uses several training strategies in order to predict a review's score. The training strategies include batch sampling,

batch normalization, and an Adam optimizer. The choice of Leaky-ReLU as the activation function was made in order to avoid exploding gradients. The difference between Leaky-ReLU and ReLU is that Leaky-ReLU has a negative slope as opposed to no slope for inputs less than or equal to zero. Activation functions are used to de-linearize their inputs, this is essential for all neural networks. The literature suggests that batch normalization layers should occur after an activation function.

MODEL A

DENSE (2470, 2048) { Leaky-ReLU }
DENSE (2048, 1024) { Leaky-ReLU }
BATCH-NORM-1D(1024)
DENSE (1024, 1024) { Leaky-ReLU }
BATCH-NORM-1D(1024)
DENSE (1024, 512) { Leaky-ReLU }
BATCH-NORM-1D(512)
DENSE (512, 64) { Leaky-ReLU }
BATCH-NORM-1D(64)
DENSE (64, 12) { Leaky-ReLU }
BATCH-NORM-1D(12)
DENSE (12, 12) { Leaky-ReLU }
BATCH-NORM-1D(12)
DENSE (12, 1) { Leaky-ReLU }
BATCH-NORM-1D(1)

Recurrent Neural Network (RNN): Recurrent Neural Networks (RNN) are neural networks that are used for sequential data input. The RNN used in this task is closely modeled after Elman Net,

which is able to learn grammatical structures from a finite number of examples. The application pattern for all RNNs in this task is a many to many problem. There are many tokenized words for input and the output is either an integer represented as a factor for classification, or a floating point in the case of regression.

MODEL B

RNN (2470, 1024)
RNN (1024, 512)
RNN (512, 256)
DENSE (256, 12) { Leaky-ReLU }
DROPOUT (0.01)
DENSE (12, 1) { Leaky-ReLU }
DROPOUT (0.01) { Leaky-ReLU }

Long Short-Term Memory (LSTM): The difference between an LSTM and an RNN is that an LSTM has a cell state in addition to a hidden state. It also has an extra logical gate that decides what information can afford to be forgotten. It also contains an input gate which combines inputs. There is finally an output gate that selects the next hidden state.

One-Layer LSTM:

MODEL C

LSTM (2470, 512)
DENSE (512, 64) { Leaky-ReLU }
DENSE (64, 12) { Leaky-ReLU }
DENSE (12, 1) { Leaky-ReLU }

Two-Layer LSTM

MODEL D

LSTM (2470, 128)
LSTM (128, 128)
DENSE (128, 64) { Leaky-ReLU }
DENSE (64, 12) { Leaky-ReLU }
DENSE (12, 1) { Leaky-ReLU }

CNN-LSTM: LSTM models can be combined with convolutional layers for enhanced feature extraction. Convolutional Neural Networks are typically used for computer vision tasks, however they can be used for Natural Language Processing (NLP).

MODEL E

CONVOLUTION-1D
CONVOLUTION-1D
DENSE (3984, 2470) { Leaky-ReLU }
LSTM (2470, 128)
LSTM (128, 128)
DENSE (128, 64) { Leaky-ReLU }
DENSE (64, 12) { Leaky-ReLU }
DENSE (12, 1) { Leaky-ReLU }

Gated Recurrent Unit (GRU): The difference between an LSTM and a GRU is the number of logical gates. There also only exists a hidden state and there is no cell state. GRU cells are also 100% compatible with basic RNN cells because there is no cell state passed on, this is not the case for LSTM models. The GRU does also have fewer parameters than an LSTM which can be advantageous for some tasks.

MODEL F

GRU (2470, 256)
GRU (256, 256)
DENSE (256, 200) { Leaky-ReLU }
BATCH-NORM-1D(200)
DENSE (200, 64) { Leaky-ReLU }
BATCH-NORM-1D(64)
DENSE (64, 12) { Leaky-ReLU }
BATCH-NORM-1D(12)
DENSE (12, 12) { Leaky-ReLU }
DENSE (12, 10) { Leaky-ReLU }
BATCH-NORM-1D(10)
DENSE (10, 5) { Leaky-ReLU }
BATCH-NORM-1D(5)
DENSE (5, 5) { Leaky-ReLU }
BATCH-NORM-1D(5)
DENSE (5, 1) { Leaky-ReLU }

4. Evaluation: Results and Discussion (Regression)

MLP

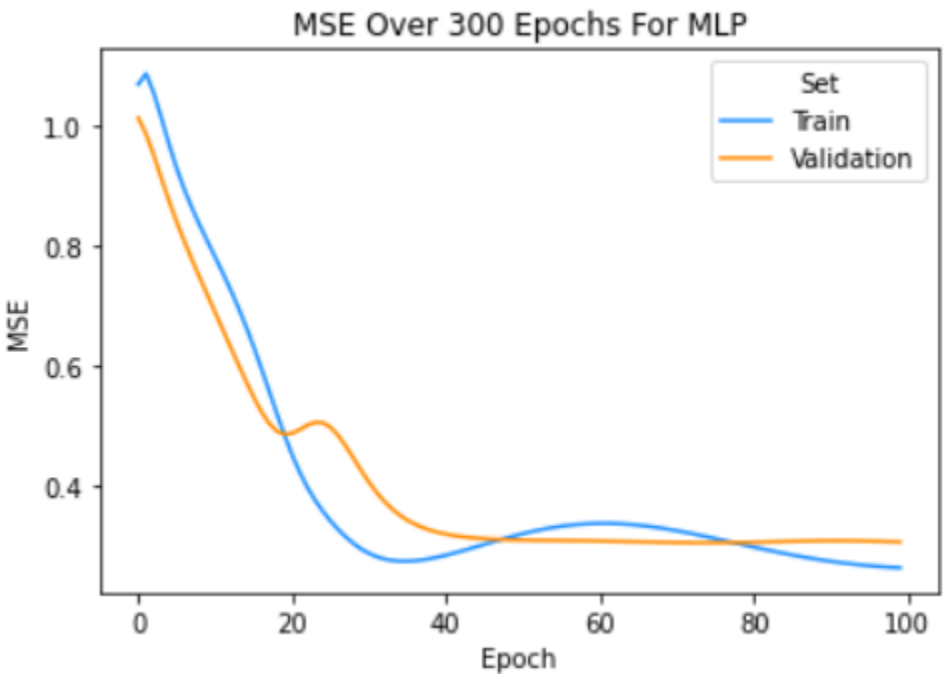

Figure 6: This figure shows the MSE loss for the Multilayer Perceptron (MLP) in *Figure X*. The training loss gets as low as 0.26, and the validation loss gets as low as 0.30. The proper way to interpret these results is that a true score will differ from its corresponding predicted score by 0.30, on average, for the validation set.

RNN:

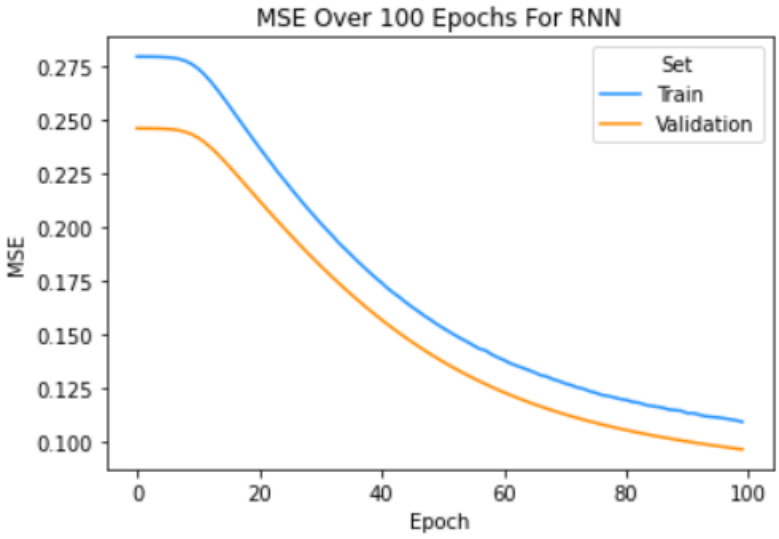

Figure 7: This figure shows the training and validation loss over 100 epochs for *Model B*. This model achieves a validation loss as low as 0.10, and a training loss as low as 0.11.

LSTM-1:

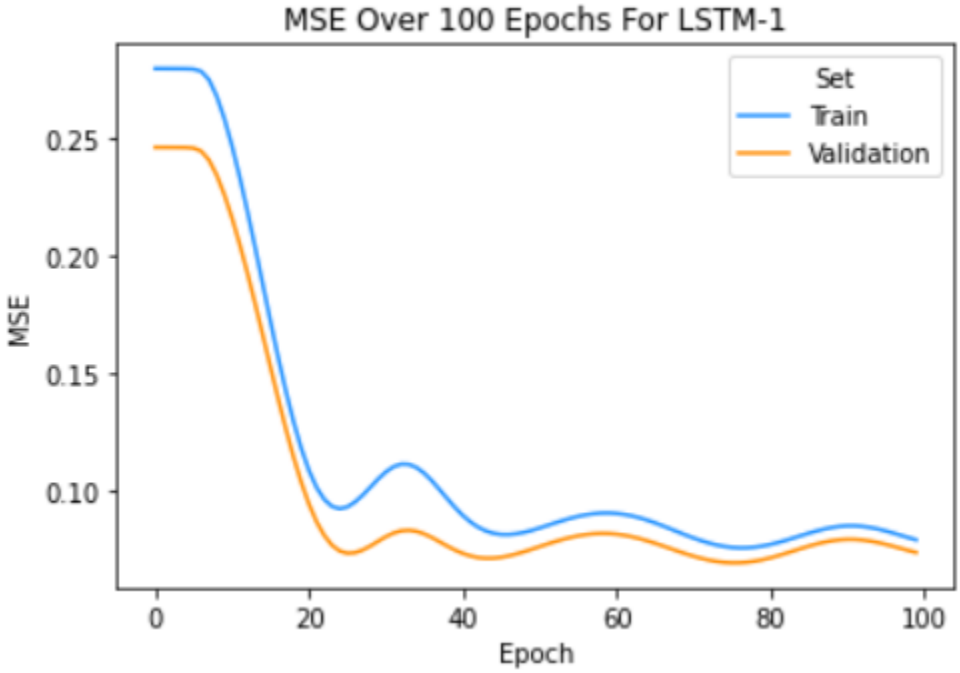

Figure 8: This figure shows the training and validation loss for the *MODEL C*. The validation loss gets as low as 0.06, and the training loss gets as low as 0.07.

LSTM-2:

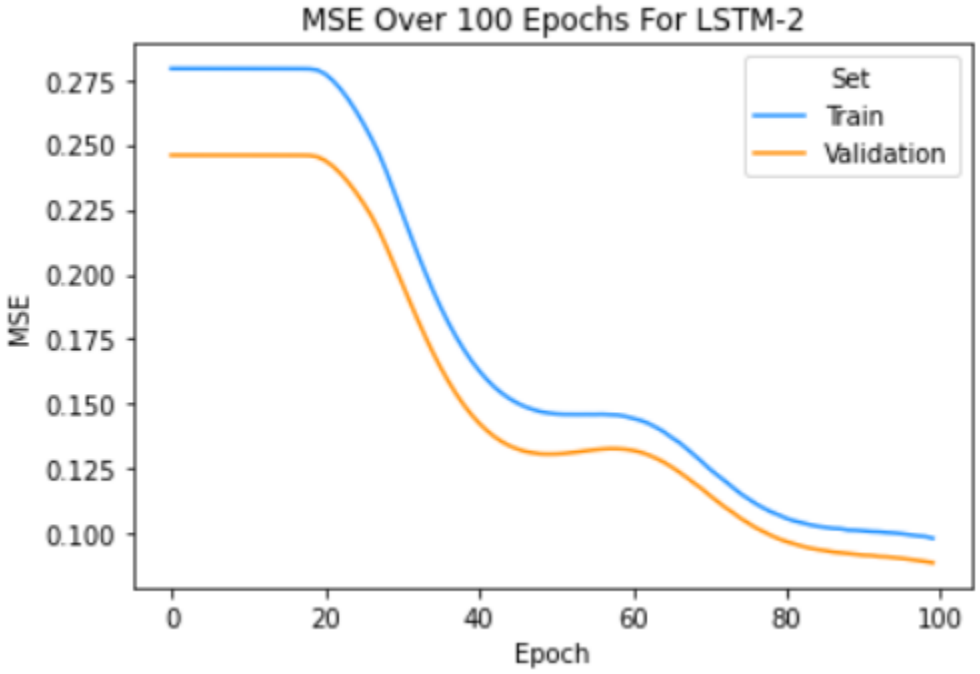

Figure 9: This plot shows the training and validation loss over epochs for *MODEL D*. The lowest validation loss was 0.08, the lowest training loss was 0.09.

CNN-LSTM:

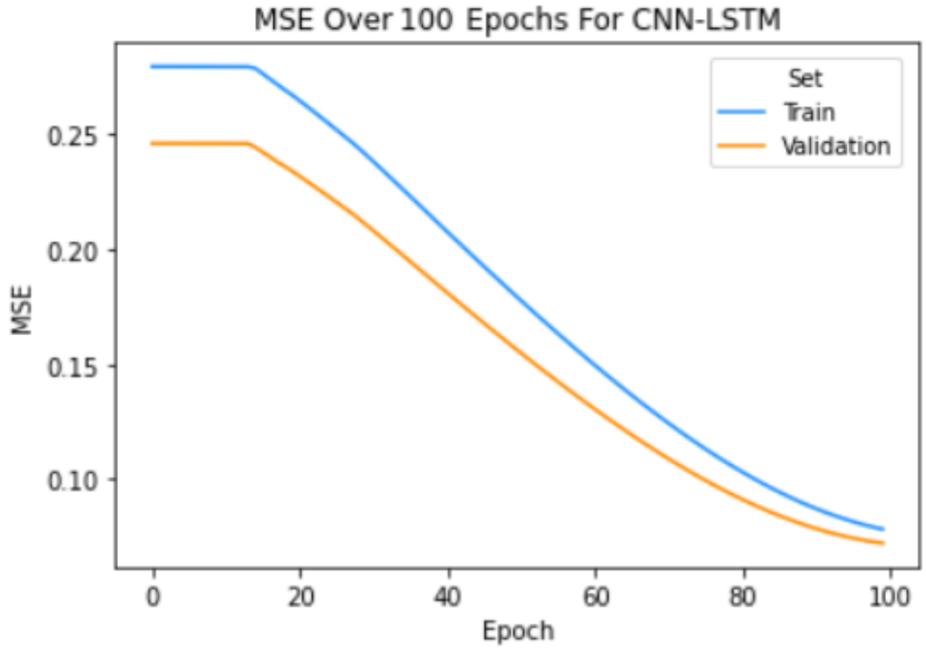

Figure 10: This figure shows the training and validation loss over 100 epochs for the CNN-LSTM model. The validation loss gets as low as 0.06, and the training loss gets as low as 0.07

GRU-2:

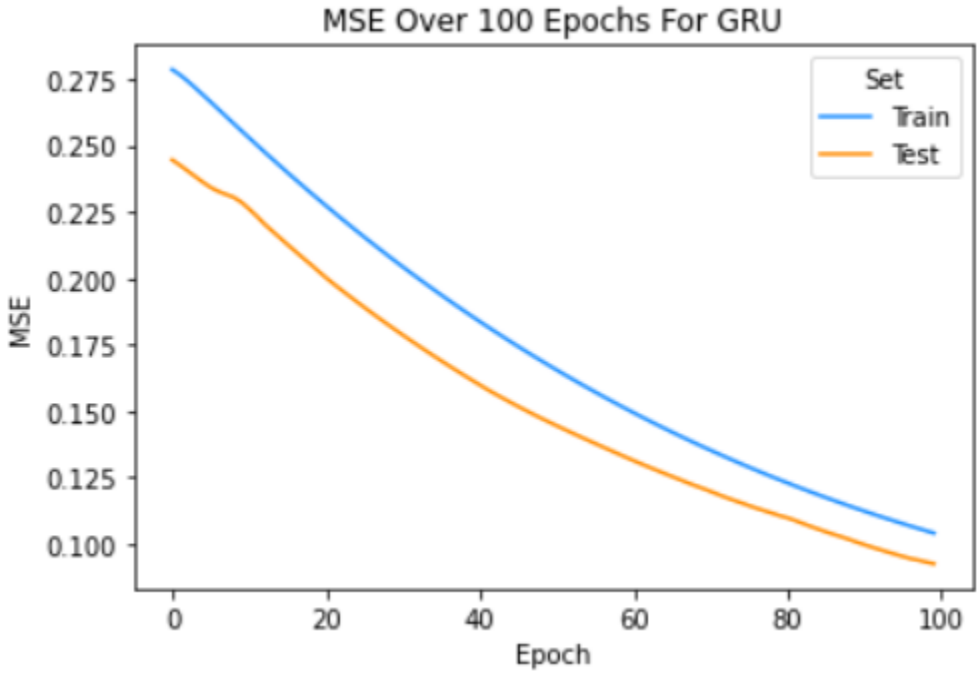

Figure 11: This plot shows the training and validation loss against the number of epochs for the GRU-2 model. The validation loss gets as low as 0.09, and the training loss gets as low as 0.10.

Final Results

Model	Model Architecture	Best Training MSE	Best Validation MSE
Model-A	MLP	0.26	0.30
Model-B	RNN	0.11	0.10
Model-C	LSTM-1	0.06	0.07
Model-D	LSTM-2	0.09	0.08
Model-E	CNN-LSTM-2	0.07	0.06*
Model-F	GRU-2	0.10	0.09

Figure 12: This table shows the final results for all six models that were trained and evaluated. The most successful observation in the table is the validation MSE loss for Model-E.

Discussion Question & Answers:

Q: What was the impact of adding convolutional layers to the LSTM model?
A: If you look at the architectures of *Model-D* (LSTM-2) and *Model-E* (CNN-LSTM-2), you'll see that the only difference is that *Model-E* has two convolutional and one linear layer at its start. Despite very similar parameters, the learning curves in *Figure 9* and *Figure 10* differ significantly. The learning curve for *Model-D* has more peaks (more hills and valleys so to speak. The *Model-D* has no decreasing training loss towards the end of the hundred-epoch training session. A flat training curve at the end of a training session is an indication of a good fit learning curve, because the training loss becomes stable. For *Model-E*, the learning curve is much more smooth, meaning there are no peaks in it. The training loss continues to decrease until the end of the training session, which is an indication of an under-fit model that would benefit from either more training data or more epochs. Since all attributes of *Model-D* and *Model-E* are held constant, except for the addition of the prepended convolutional layers, we may reasonably conclude that the addition of convolution made the LSTM model more under-fit. Logically, this is what would be expected because the purpose of convolution is feature extraction. The convolution creates more feature space through the process of filtering and pooling. It would make sense that this expanded feature space would take more epochs to produce a stable training loss curve.

Q: Why is the validation loss less than the training loss in many cases?
A: The likely root cause of this phenomenon is a dropout rate that is too high. In PyTorch, dropout layers are only activated during the training phase of the model. Regularization techniques in general tend to reduce over-fit models, so the Batch Normalization layers could also be a factor in the high training loss. The other potential cause is that the validation set has more reviews with obvious scores. Given the size and equal balance of the testing and validation sets, the latter cause is unreasonable to presume.

Q: Which models could benefit from more training?
A: *Model-F*, *Model-E*, and *Model-B* could all benefit from more training epochs. *Model-E* and *Model-F* are more clear candidates for more training epochs, because it is more clear that their training losses are still decreasing at the end of the training session. When training losses continue to decrease at the conclusion of the training session, then a model is likely under-fit, and could benefit from more training epochs.

Q: Which models could benefit from a lower learning rate?
A: When a learning rate is too high, a learning curve will be flat over a stretch of epochs. For this reason *Model-B*, *Model-C*, *Model-D*, and *Model-E* could all benefit from a lower learning rate. Lower learning rates will help a model converge to a solution faster, however if the learning rate is too low, then the learning curve will have more hills and valleys (more local maxima).

Q: Which model had the best learning curve?
A: A good learning curve is neither under-fit nor over-fit. In good-fit learning curves, the training and validation losses decrease until they eventually stabilize before the conclusion of the training session (before the epochs run out). The training curve should be very close to the validation curve for the final epochs. Given these conditions for good learning curves, it is clear that *Model A* has the best learning curve, despite having the highest validation loss.

Q: Are some models better at predicting ratings for positive reviews, or vice versa?
A:

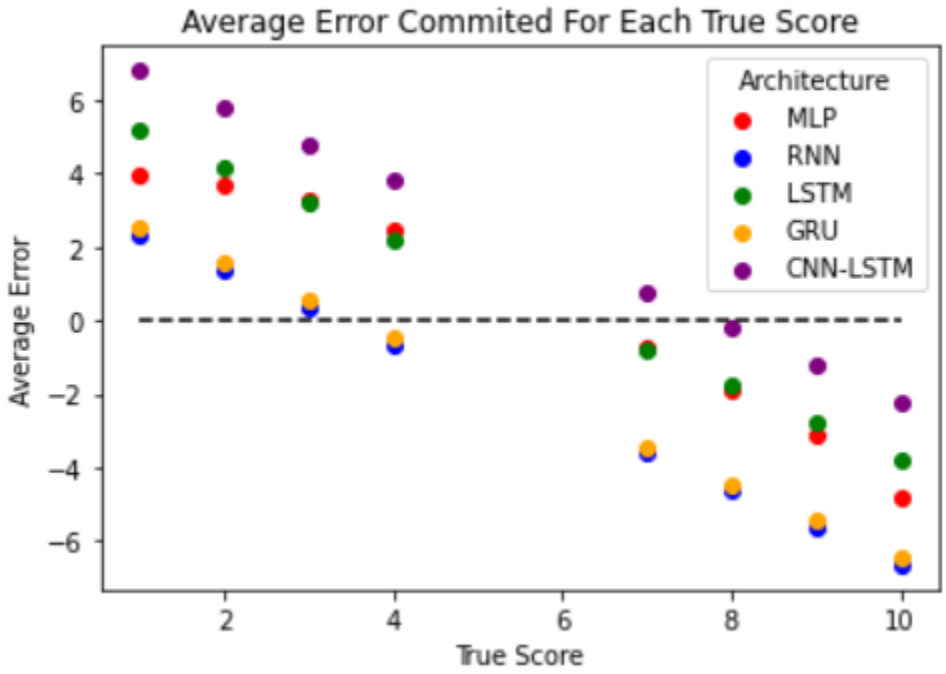

Figure 13: This figure shows the average error committed for each architecture across all true scores. You will notice that there are no entries for true scores of "5" or "6". This is because there are no such examples in the validation set. For example if we are looking at the true score of "2", and we wanted to know which model makes the best predictions when the true score is actually "2" we would be able to deduce that the best model is the RNN because the blue dot is closest to the black line. The black line represents a perfect prediction because the average error is zero. You can see that in some scenarios the average error approaches the black line. One such example is for the CNN-LSTM model when making predictions from reviews where the true rating is "8". Another such example is RNN models for true scores of "2".

5. Conclusions & Key Takeaways

In conclusion, it is clear that the RNN architectures were able to predict the rating score more accurately than the MLP architecture. The RNN architectures refer to all LSTM, GRU, and RNN models that were studied. In terms of validation loss, the results for all RNN-based models were likely not statistically different from each other. There is a random component when building models in PyTorch, and so it is important to know that the slight differences in validation loss between LSTM models and GRU models are likely meaningless.

Another takeaway from this project should be that a Multilayer-Perceptron can be really useful in some cases. Although the MLP studied (*Model-A*) had the highest validation loss, it still had the most successful learning curve. Something advantageous about the MLP architecture is

that there can be more experimentation with hyper-parameters because there are fewer levels of abstraction. For example, it is much easier to see what the direct result of a batch normalization layer might be in an MLP than a CNN-LSTM, because the CNN-LSTM is a more complex system.

One final take away from the findings here is that convolutional layers can have big impacts when prepended onto other architectures like an LSTM. The power behind this phenomenon is feature extraction. When more information can be extracted from input features, it will take longer for a given model to train because there is more information to learn, this can be beneficial when predicting unseen examples.

6. Personal Reflection: Challenges Encountered, Lessons Learned

Challenges Encountered:

1. Building Pytorch models from scratch can involve a lot of experimentation with tensor dimensions.
2. LSTM models with multiple layers require some non-intuitive programming in the forward method of the class definition when working with PyTorch
3. It can be time consuming to run models without a pipeline.
4. Hyper-parameter tuning is tedious without a pipeline that automates the process.

Lessons Learned:

1. If a model's training loss is still decreasing at the end of all of the epochs, it is okay to add more epochs because the model is still acquiring new information. There is no standard number of epochs that should be used for all tasks.
2. Sometimes a fully connected layer, where the input dimension is equal to the output dimension, can really improve the performance of a model
3. More layers can lead to overfitting, even when regularization methods are used.
4. Some problems can be taken as classification and regression tasks.
5. There is a tradeoff between a low learning rate and noisy learning curves (curves with many local maxima).
6. The batch size is a very important parameter, because it affects how much information the model has to learn.
7. Batch normalization layers can decrease the number of local maxima in learning curves, and thus allow for a lower learning rate. Lower learning rates are synonymous with faster learning rates because the gradient steps are larger.
8. The number of linear layers following an RNN cell should be three or less.
9. PyTorch has a random component and so setting the random seed can help you compare results more effectively.