My capstone project for Machine Learning is a stock earnings predictor. I implemented this as an ensemble model containing a recurrent neural network and a linear classifier. It takes both time-series and scalar data as input. The RNN processes the time series data and outputs an embedding that is used along with the scalar data to predict the magnitude of the earnings beat or miss.

Publicly listed companies produce an earnings report each quarter that describes their performance, which investors care deeply about. Whether or not the earnings reported beats the expectations the market has for the earnings determines large fluctuations in the stock price in the days after it is released. Since the stock fluctuations are largely dependent on whether or not expectations the market had for the earnings were exceeded or not, this problem works well as a classifier with it classifying between beat or miss. This task is important because having a reliable predictor for determining earnings misses and beats allows one to make a lot of money because it would essentially allow you to predict stock price fluctuations. If you know the stock will go up or down, you can take a position in it that will profit when that happens. On a more altruistic note, the task is important because the model would be able to make sense of current data related to earnings and make a prediction. If this prediction was publicly disseminated, people would react to this prediction before earnings come out. They would push the price of the stock closer to the true value it would be at after the earnings were released. This would decrease volatility in the market which is detrimental to the financial system because it increases risk and makes it harder for companies to be sure of the capital they raise.

To make a classifier that is successful in predicting earnings, I needed to gather a dataset. I wanted this dataset to contain information that could convey a lot about a stock's future earnings, but also in complex enough ways that a deep learning model would be the right way to approach the problem, rather than a less complex pattern recognition algorithm. There were a multitude of types of information that fit this description for the problem. The most significant one I wanted to look at is analyst estimates for earnings. The market expectation for a stock's earnings is the average of the analyst predictions for the earnings. These analysts are employees of various financial companies and they are experts in their specific companies. When a person is determining the success or failure of a company's earnings, they look to this average of the analyst predictions to see whether they did better or worse than expected. The expected earnings is regarded as a fact, but really it is just these analyst's predictions, and humans are not perfect. Some analysts will be better at predicting earnings than others, so granular analyst prediction data could convey a lot of information. Analysts also have the ability to revise their forecasts leading up to the earnings based on new information. When this is done frequently before earnings or in a specific way it can be indicative of what is to come based on the earnings, because it indicates the analysts are not sure or just received new information. One of my hopes was that the model would be able to pick up on these patterns.

Some other dynamic data that I wanted to provide the model was volatility of the stock leading up to the earnings, and also volume of shares traded leading up to the earnings. I thought that abnormal numbers for these variables could indicate a lot about earnings performance.

People moving in and out of the stock a lot again indicates that they are unsure, or are receiving new information that updates their view of the stock.

While most of the focus in the earnings is on the change in a company's outlook, it is still important to keep in mind the general information of the company when forming an earning's prediction, to provide context. Background information I wanted to feed into the model included market capitalization, sector, and industry of the company.

No singular dataset had all of these variables that I was looking for. I knew I was going to have to construct it out of different sources. What I did not account for was the lack of publicly available historical financial data. Unfortunately, because of the high demand for financial data (it is what advanced financial firms use to develop trading strategies), companies have sprung up as vendors of the data, charging exorbitant fees. That means the places where I could find the data I was looking for, for free, were pretty sparse. I was able to find a good dataset on Kaggle though that covered most of my needs, and I pulled the other data I wanted from the yahoo finance api. I unfortunately was not able to get super granular analyst prediction data.

It took significant preprocessing to get the data into the form of examples that I wanted. The Kaggle dataset was a group of .csv files, one of which contained earnings estimates and true earnings results for thousands of companies with varying timeframes, and one of which contained stock price and volume data. The format I wanted for each example was a list of time series data spanning 7 consecutive earnings reports, and based on the date of the last earnings report, the previous 3 month's stock volatility. I also wanted the market capitalization, sector, and industry of the stock. In order to assemble this, I first gathered the companies in the dataset that had over 7 years worth of earnings data. I did not want to include companies in the dataset where I could not form many examples from the data, as I thought that might just confuse the model. I also only included the top 100 companies by market cap, as I thought smaller companies would be too volatile to draw meaningful patterns from their earnings. For these companies that had over 7 years of data, I pulled their market cap, sector, and industry data from yahoo finance and created dictionaries so that I could easily look this data up when creating the examples. I converted the sector and industry categories into one hot vectors to make it more interpretable for the model. I normalized the market cap data in order to not have super large inputs to the model.

To form the examples, I iterated through each of the 100 largest stocks. For each one, I pulled the earnings information for it, which consists of a date of earnings release, an earnings estimate, and the actual earnings result, along with which quarter the earnings occurred. Each time series was going to contain 7 of those instances, 6 of them forming the example and the last one being the label for the example. When creating the examples here, there was a decision I had to make. Originally I was not going to make the data overlap - if there were 28 earnings instances, I was only going to make 4 examples out of it. I thought that I ran the risk of muddling training and test data if I made the data overlap. However, I did some research and it is common practice to overlap the sequence data for input into RNNs. Because of this, I was able to get at least 21 different examples out of each of the companies, yielding me 2100 examples in my dataset. For each point in time of the time series, it contained 6 features - the earnings estimate, the actual earnings, and a one hot vector indicating which quarter the earnings was from. Earnings performance varies by quarter. For example, Q4 includes holiday spending which boosts a lot of companies' earnings. I thought this conveyed enough information to be worth including. Once I had this time series data for a specific example, I pulled the stock volatility data for the 3 months prior to the date of the earnings of the label. I constructed this data from the second .csv file in the dataset. I also performed a similar operation with the volume data, but I normalized the volume data so as to not include super large numbers as inputs to the model. I then pulled the market cap, sector, and industry data from the dictionaries I had created.

I originally wanted to create a model that would solely classify between earnings beats and misses, for example outputting 1 if beat and 0 if miss. However, when it actually came to implementing I found this did not make sense for a few reasons. First, there would be instances where earnings estimates were directly on the actual value, yielding neither a beat nor miss. I could not represent this in the classifier without creating another class where I would not have as many examples to train on. Second, the point of the model is to determine earnings beats or misses so that you can capitalize on stock price fluctuations, but the amount the stock price will fluctuate is based on the magnitude of the earnings beat or miss. This means that a basic classifier would not be of much use. I decided to make the model output a continuous value between -1 and 1.

In order to create the label for each example, I took the seventh earnings instance from the time series and subtracted the earnings estimate from the actual earnings. That meant that an earnings miss would be negative, being correct on earnings would be 0, and an earnings beat would be positive. In order to make the labels something that the model could reproduce reasonably, I had to normalize the labels. I divided all of the differences by the absolute value of the max difference between earnings and earnings estimate for each stock. This resulted in labels ranging from -1 and 1, with -1 meaning missed earnings by a large margin, and 1 meaning beat earnings by a large margin. This completed the production of the example. I saved all of these examples as pytorch tensors. From previous experience, this makes it much faster to load the examples from memory at train time.

I chose to use pytorch to create this model as I have used it before and was comfortable implementing my custom dataset in it. As mentioned before, I created an ensemble model with an RNN to handle the time series data and linear layers to output the prediction. The purpose of the RNN was to create a representation of the time series data which then could be used as an input along with the other scalar data into the classification model. I originally tried using a vanilla RNN for the encoding task, but I ended up using an LSTM to protect against exploding gradients and for the additional encoding capability. I also created a model using a convolutional neural network for the encoding task, but I found that the RNN based model had better performance, although marginally.

In terms of the actual architecture I used for the model, I chose to use the Adam optimizer with a learning rate of 1e-3. I have found that the Adam optimizer is better for more complicated models like RNNs. For the learning rate - I arrived at it through experimentation. I also arrived at a batch size of 64 through experimentation. For the activation I used for the model, I did not have much of a choice because, as previously explained, I wanted the output to be between -1 and 1. A tanh activation function gave me this output, so I used it.

When training the model, I ran into a significant error of exploding gradients. The model would get through a couple batches and then the loss would go to infinity and then be nan. I debugged this for a long time. I thought it was an issue with how I had set up the RNN, or that my activation function was wrong, or that the model was too complicated, or that my gradient was being updated too strongly. I changed all of these options and searched for solutions online, but to no avail. This is what spurred me to create the model with a CNN instead of an RNN, to see if the problem really was the RNN. However, the CNN also had an exploding gradient problem, which led me to the fact that there must be something wrong with my data. There was - in my processing I had accidentally created a few labels that had values of infinity, this was causing my loss to blow up when backpropagation was run. I resolved this by reprocessing my data to amend the error in the label calculation.

I found the model trained for 5 epochs before it started overfitting the training data. This is a relatively low number of epochs, I think this is due to there being a large number of training examples for a model that is not that large. Since the model is outputting a continuous value between -1 and 1, there is no accuracy data for the model, but there is mean squared error to look at. I was able to achieve mean squared error of 0.122 on the training dataset and 0.089 on the test dataset. These numbers are a little harder to interpret than accuracy numbers, but the way I think about it is if you take the square root of them, that is the average amount the model is off by. So for the test dataset, the model was on average off by 0.298. The range of the data is -1 to 1, so on average the model was wrong by 14.9%. Although this performance is not great, in the context of earnings beats or misses, if there is strong conviction of an earnings beat or miss, only being wrong by 14.9% would not usually swing the model the other direction, so it is still useful. I am satisfied with the performance based on the relatively small amount of data I had to work with.

If I were to continue working on this project, the main way I would seek to improve it is through making my training examples have more information. As I mentioned before, finance data is expensive, so it is hard to source data to experiment with. However, if I was able to get funding or appeal to some of the vendors because this is an academic project, I may be able to gain access to more data. The main thing I have interest in including is more granular earnings estimate data. I think there is a lot of predictive power in this data, and I would be interested to see how it improves the performance of my model.

The other way I would seek to improve my model is through better architecture. This goes hand and hand with accumulating more data in each training example. Since there was not that much data in each example, I did not have much ability to experiment with more complicated models for the classifier part. If I had more data being input to the classifier I could have experimented with some convolution layers which may have had better performance. Also, in terms of the encoder I used for the time series data, I feel like training that myself may not have been the best option. With the advent of transformers and large language models, there has to have been advancements in representing time series in lower dimensional spaces. However, in all the research I did for encoders for time series, the only thing that came up was encoders for text sequences, as that is what the LLM advancements are based off of. Perhaps I was not looking in the right place, but a better representation of the time series data would make it easier for the model to classify the data.