CMP\_SC: 8770 Neural Networks

Project 2: Sequence/Temporal-Based Neural Networks

By: Mikey Joyce

Pawprint: MPJYKY

Due: April 24th, 2024

1. Technical Description:

General Introduction:

In this project, we were asked to implement a recurrent neural network (RNN) and long short term memory (LSTM) neural network on a dataset of our choosing. The goal is to compare and contrast the two different types of networks and observe their performance on the dataset of our choosing.

The dataset I am choosing is again going to be the Indianapolis Glaucoma Progression Study (IGPS), the same dataset that I utilized in project #1. When taking into account the results I produced with the RBF, you may be wondering why I would like to utilize the same dataset to attempt to solve the progression problem again, and I believe that is because due to the dataset’s temporal nature. Recurrent neural networks and long-short term memory networks could possibly be useful in classifying progression because progression itself is a temporal concept due to the nature of requiring multiple datapoints to identify its presence (i.e., if you have no baseline data from where the patient was before, how do you know their condition got worse?). This concept itself seems to be a struggle with the clinical datasets in general as many of these datasets will make massive conclusions from only two data points: an initial visit, and a follow up visit. When analyzing a disease such as glaucoma that progresses slowly overtime, it is a curious approach to only collect data twice over multiple years’ time. This is where IGPS comes in. We have data that was collected over many visits every six months for a couple of years. Utilizing this data, I am proposing the utilization of RNNs and LSTMs to attempt to digest this data and identify the different progression labels. I will focus my experiments on binary classification of the labels, so only one form of progression at a time. The main reason for this decision is because since the progression labels are derived differently due to their different natures, I will not be utilizing the same feature set for each type of progression. Last time I only focused on structural progression, and as it turns out that isn’t necessarily intuitive to model. The other type of progression is called functional progression. I also have two sets of labels for each type of progression, the first type of label is defined as an “indicator” by the MD’s that crafted the dataset. This basically means it is indicating that they are progressing at the moment, this indicator can turn on and off at any time based on the data. The second type of label is the official progression label, which only turns on if the indictor flashed “True” or “1” for two consecutive visits. If the official progression label turns on, it stays on for the entire sequence. The way I interpret this in the big picture for the RNN and LSTM is that the indicator label is representing a current status of the patient’s glaucoma progression at any given time step and the progression label is essentially saying at any point in time during this sequence did the patient experience progression. The type of progression label I was attempting to predict with the RBF was the structural progression indicator, not the official structural progression label. Since the official structural progression label turns on and cannot turn off, it is easier to predict, but it may not be as useful because it seems to be more of an observation after the fact rather than the current status of the patient’s glaucoma. Despite it being not as good as the indicator, I am still including in this project since it is easier to predict, so it may act as a good starting point for this difficult problem.

Recurrent Neural Networks:

First, we will start with the question, what is a recurrent neural network? A recurrent neural network, like many of the other networks we have talked about in this class, have a set of weights, biases, activation functions, etc. that acts as a universal function approximator. The main attribute of an RNN that other neural networks such as a multi-layer perceptron do not posses is the feedback loop. This feedback loop allows for the processing of sequential data, such as time series data. Here is a general diagram of an RNN that shows the feature that the RNN has introduced to the neural network computing paradigm.

A diagram of a network

Description automatically generated

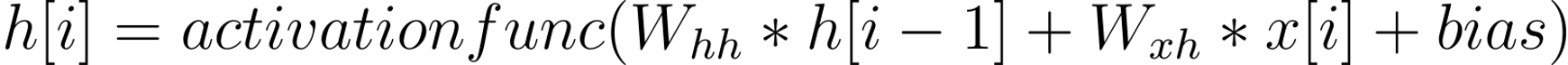
*Fig. 1.1: Very general RNN diagram*

The feedback loop was mentioned, but the process of what the feedback loop actually is was not yet discussed. This feedback loop is a method for the network to be able to take into account what happened in the past to make a future decision/prediction. What is happening is each time step utilizes a shared weight neural network (f[w]), where it takes in an input (x[i]) and computes an output (y[i]) utilizing the weights and biases. The network is integrating the output at each time step into the next time step through the “hidden state” (h[i] & h[i-1]; feedback loop). Here is a diagram of what was just discussed and the equations for the update of the hidden state and the output at each time step.

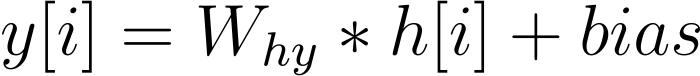
A diagram of a flowchart

Description automatically generated

*Fig 1.2: RNN flow diagram, shown in a way that loops through so no need for large, expanded diagram that shows each individual time step. In this diagram, h[i-1] is initialized as zero.*

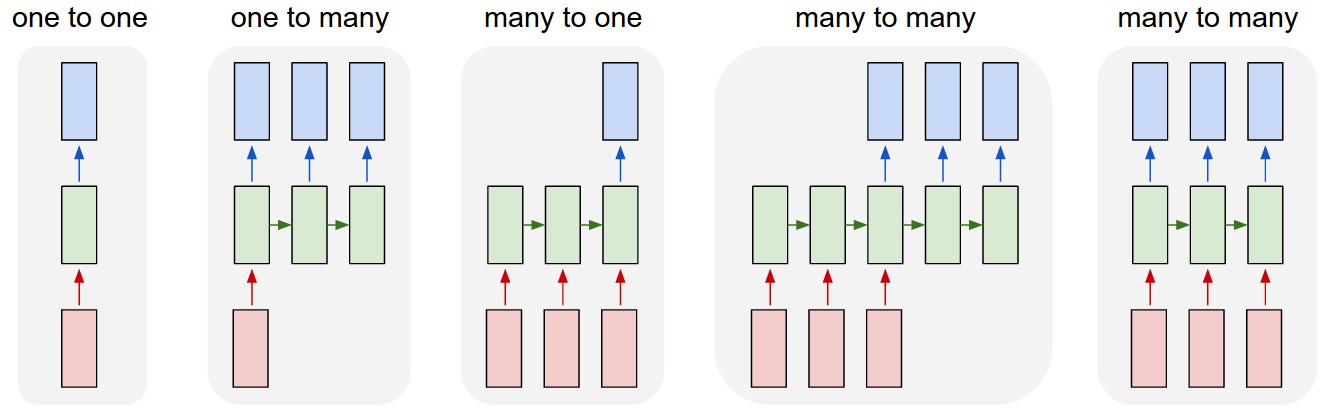


*Fig. 1.3: Hidden state update equation*



*Fig 1.4: RNN output equation*

With the diagram and equations, we can clearly see how the recurrent neural network takes into account the past at each time step. We can see through this though that the network at each time step just shoves/includes all of the information into the hidden state, no matter if this information is useful or not. This realization is a limitation/weak part of utilizing an RNN because it cannot get rid of the information that we do not want. Later in this section we will discuss a long-short term memory (LSTM) neural network that will provide a solution to the problem just discussed. But before we jump ahead let us talk about different ways to formulate a problem to an RNN (there are lots of ways). The first way is through a one to one relationship, where we take one input and just compute one output. However, doing this is kind of a waste of the RNN architecture in general because we are essentially doing the same thing that is happening in a multi-layer perceptron or a convolutional neural network, we haven’t done anything new since we have not propagated through time due to the one time step prediction. Another method is the one to many relationship. In this relationship we take in an input and try to compute N predictions through time. This method could be difficult to train since we are giving the network only one time step and asking it to model the sequence, even though it hasn’t seen much of the sequence at all. The next type of relationship is the many to one, where we take in a sequence and try to predict the next step in the future. The last type of relationship is the many to many relationship. There are a couple ways to formulate this specific relationship. The first is we take in a sequence, and after this sequence has propagated through time we try and predict a new sequence (i.e., we ingest x1, x2, x3, and after ingesting we predict y4, y5, y6). The second type of formulation for a many to many relationship is where we have a sequence and we are predicting a sequence, however instead of waiting for the initial sequence to propagate through before we start predicting, the sequence we are predicting is happening in tandem with the inputs (i.e., we ingest x1, predict y1, ingest x2, predict y2, ingest x3, predict y3, etc.). Here is a visualization of these paradigms, I understand it is not good form to take in charts off the web for this type of assignment, but it seemed a bit redundant to make this chart from scratch and I wanted to include a visualization to go along with my explanation.



*Fig. 1.5: RNN formulation paradigms, taken from* [*http://karpathy.github.io/2015/05/21/rnn-effectiveness/*](http://karpathy.github.io/2015/05/21/rnn-effectiveness/)

The last item that needs to be answered to develop your own RNN is how to actually train this network. A traditional neural network like a multi-layer perceptron is trained utilizing the backpropagation algorithm. However, since the network is essentially unraveling an entire network on each time step, the traditional backpropagation algorithm will not work. The algorithm that needs to be utilized to train the RNN is a variant of the backpropagation algorithm known as the backpropagation through time (BPTT) algorithm. The key difference is on each time step the network is doing a backpropagation and since it is going through time it does this for each time step and then sums up the gradients. Since often times this unfolding can get very large from having a large sequence, training RNNs can be difficult.

Long-Short Term Memory Neural Networks:

Now that we have covered what an RNN is, we must improve on this infrastructure with the LSTM. But what is an LSTM network? An LSTM network is an improved version of the RNN where it has all of the same functionality with an extra feature called the cell state. This cell state is essentially a memory function for the network, where it allows the network to “decide” to include or not include information from the past. This is the general functionality that it introduces, although when shown in a diagram it appears a lot more complex with all of the wires that are required.

A diagram of a computer

Description automatically generated

*Fig. 1.6: A diagram of a single LSTM cell*

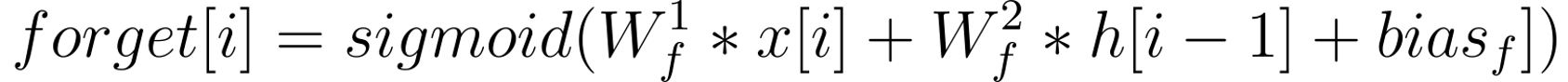
So, with the diagram that I created, we can see that this cell state adds lots of complexities that the RNN does not have. To add a sense of memory, the LSTM is introducing three logic gates that determine what to do with the input and the past. The forget gate and the input gate are used simultaneously to tell the LSTM what to send to the output gate, here is a simple table showing what these gates do for the cell.

|  |  |  |
| --- | --- | --- |
| Input Gate | Forget Gate | Output Gate |
| 0 | 1 | Remember previous state |
| 1 | 1 | Add to previous state |
| 0 | 0 | Erase previous state |
| 1 | 0 | Overwrite previous state |

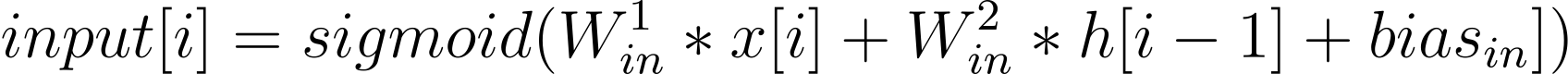
*Table 1.1: Showing actions an LSTM cell can take*

Each LSTM cell can essentially do four different things into account shown in the table above. The hidden state, like the RNN, is the output of the network. But the cell state is essentially the memory context the network has and is dynamic as it can determine when to remember things, when not to remember, and when to reset.

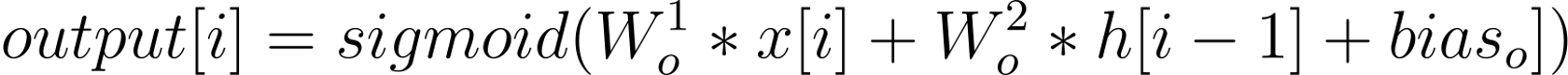
Lastly, we must cover how the LSTM updates the three different logic gates that it introduces and how it incorporates these gates into the hidden state and the cell state to allow for the sense of “memory.” The equations that show the updates for the parameters are below:



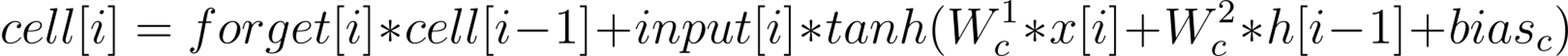
*Fig 1.7: Forget gate update equation*



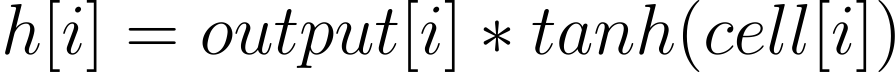
*Fig 1.8: Input gate update equation*



*Fig. 1.9: Output gate update equation*



*Fig. 1.10: Cell state update equation*



*Fig. 1.11: Hidden state update equation*

As we can see from the equations there is a clear structure to allow the updating of the logic gates that were introduced and the added cell state.

Since fig. 1.6 is just a single cell, an LSTM layer consists of a certain number of LSTM cells that the user defines. The user that creates an LSTM can also stack these layers and essentially have multiple LSTMs feeding into each other to create a more complex network. Now that we have discussed what makes an LSTM different and how it improved on the RNN, it is time to compare and contrast these networks in practice by applying it to a hard problem. “Theory will only take you so far”- Oppenheimer (from the movie I don’t know if he actually said that).

1. Experiments and Results:

In this section we will run some experiments for the RNN and the LSTM and compare and contrast their results on the IGPS dataset. I will break up the experiments in two sections, the functional progression section, and the structural progression section. In each section I will run 8 different experiments, since there are two different networks, with two different labels. I want to run two experiments per label per network one will try and be a simple/general solution and the second will be a grid search to see if we can find the optimal network. 2 algorithms x 2 labels x 2 experiments = 8 experiments per section with a total of 16 experiments. For the indicator label, since it is dynamic, I will opt for a MIMO formulation. For the official progression label, since it is more akin to analyzing whether or not progression was experienced by the patient at any point in the sequence, I will opt for a MISO formulation.

Functional Progression:

A table with numbers and letters

Description automatically generated

*Fig 2.1: Feature vector for functional progression experiments*

During project #1, we saw it was difficult to predict progression with the formulation that I originally had. I think part of this is because the features did not relate to the definition of progression in glaucoma. Because of this I have opted to change the feature vectors to be variables that have a relationship (or at least are supposed to have a relationship) with the given progression labels.

*Official Functional Progression Label Experiments:*

The first experiments that we will conduct will be with regards to the official functional progression labels. I decided to choose this one first because this label is the easier label to “predict.” We essentially will be formulating these experiments as a MISO experiment since we want to ingest the sequence and give the user an analysis, or claim, that the given patient experienced functional progression at some point during the sequence. Our data has 69 sequences (patients) available to us, so 54 of the patients will be for training, and 9 patients will be for testing (I know that is only 63 of the patients, 6 patients are not analyzed). I am considering my “test” results more of a validation set, since it is contained within the same dataset as the training data. Ideally, the test data would be a completely different dataset as that it is most valuable to have a model that can transfer across datasets in clinical analysis. Let’s compare the results of the general LSTMs and RNNs and the grid search (optimal) LSTMs and RNNs. When performing the grid search of the hyperparameters I decided in case of a tie, the simpler model is the one that wins, as we want the cheapest computations. It is worth it to note that during all experiments in this project I utilized BCE loss, Adam optimizer, and a learning rate of 0.001. I wanted to change as little as possible to keep a controlled experiment.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Model | Hyperparameters | Accuracy | Precision | Recall | F1 |
| RNN Base | Hidden Size: 24  Num Layers: 1 | 0.778 | 1.000 | 0.500 | 0.667 |
| RNN Grid Search | Hidden Size: 2  Num Layers: 1 | 1.000 | 1.000 | 1.000 | 1.000 |
| LSTM Base | Hidden Size: 24  Num Layers: 1 | 0.889 | 1.000 | 0.750 | 0.857 |
| LSTM Grid Search | Hidden Size: 1  Num Layers: 1 | 1.000 | 1.000 | 1.000 | 1.000 |

*Table 2.1: MISO IGPS functional progression analysis over the official progression label*

As we can see from these experiments, we were able to find a model that could 100% classify our 9 validation patients with respect to functional glaucoma progression. While this is good to see, this is more of a sanity check. To actually prove whether or not this model is strong we would need to perform an experiment on a test set that is not IGPS and one that has more than 9 patients for testing. What I thought was interesting in this experiment was that the smaller models had the best performance. When watching the grid search perform, the larger the model got on this experiment, the worse it performed. Also, it is worth it to note that the “base” LSTM model outperformed the “base” RNN model, which was something Dr. A mentioned might happen.

*Functional Progression Indicator Label Experiments:*

Now, let us try the same experiment on the indicator progression label, which is much harder to predict. Since this label requires us to formulate a MIMO experiment, we have a little bit more data to test on (because 9 patients x 9 sequence length = 8).

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Model | Hyperparameters | Accuracy | Precision | Recall | F1 |
| RNN Base | Hidden Size: 24  Num Layers: 1 | 0.790 | 0.679 | 0.704 | 0.691 |
| RNN Grid Search | Hidden Size: 3  Num Layers: 3 | 0.877 | 0.758 | 0.926 | 0.833 |
| LSTM Base | Hidden Size: 24  Num Layers: 1 | 0.741 | 0.688 | 0.407 | 0.512 |
| LSTM Grid Search | Hidden Size: 33  Num Layers: 2 | 0.901 | 0.828 | 0.889 | 0.857 |

*Table 2.2: MIMO IGPS functional progression analysis over the progression indication label*

Here we can see that the base models produced a worse result than in the last experiment. Something that is also noteworthy is that the RNN base was superior compared to the LSTM base model. Since there was a bit more validation (testing) data and the nature of the label being a little bit more dynamic, we expect the model to not achieve a perfect result. However, the LSTM model that was found through the grid search of the two hyperparameters seemed to get a pretty decent result, with a 90% accuracy over the validation data. It was also good to see that this model was able to get a pretty decent recall score, since that is more preferable over the precision score, as the consequences for a patient getting misclassified when we say they have progression is less than the consequences if we say they do not have progression, when in reality they do. Because of this point, you could make the claim that the RNN model found through grid search is a superior model due to its higher recall score, despite having a worse accuracy than the LSTM grid search model. Not only this, but the RNN model is a smaller, more elegant model. While it has more layers than the LSTM grid search, it has significantly less computational nodes. It was also the simplest model in this experiment. It is worth it to note again, that despite these results looking somewhat favorable, since we don’t have a non-IGPS test set, this just ensures that the models are doing what I expect (to some degree). Further testing would have to be done to ensure the model is not overfitting or doing something else that is not desirable.

Structural Progression:

A screenshot of a computer

Description automatically generated

*Fig. 2.2: Structural progression feature vector*

Now that we had pretty decent results in the functional progression experiments, it is time to take it up a notch by taking a look at structural progression. Again, we will start with the official structural progression labels which will be a MISO data formulation again. Then we will move to the progression indication labels (this was the label I was trying to predict with the RBF). These labels are a bit more complex than the functional progression, which means they are harder to predict, so I am expecting the results to be worse than the experiments that were just conducted. Not only this, but I don’t have access to the baseline data, where in the functional progression I had access to the baseline data, which makes it easier to determine progression with the baseline.

*Official Structural Progression Label Experiments:*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Model | Hyperparameters | Accuracy | Precision | Recall | F1 |
| RNN Base | Hidden Size: 24  Num Layers: 1 | 0.444 | 0.571 | 0.667 | 0.615 |
| RNN Grid Search | Hidden Size: 59  Num Layers: 2 | 0.778 | 0.833 | 0.833 | 0.833 |
| LSTM Base | Hidden Size: 24  Num Layers: 1 | 0.444 | 0.571 | 0.667 | 0.615 |
| LSTM Grid Search | Hidden Size: 11  Num Layers: 2 | 0.778 | 0.75 | 1.0 | 0.857 |

*Table 2.3: MISO IGPS structural progression analysis over the official progression label*

Intriguing enough, there was no difference at all between the base RNN and LSTM. While there was no difference, these clearly are terrible models. So, since they are bad, it is clear a grid search needs to happen to see if we can find another setup for the infrastructure that can approximate the function that determines whether or not a patient experienced structural progression at any point in our 9 visit sequence. From the results, we can see the LSTM clearly beats the RNN due to its perfect recall and finding a structure that has much less computational nodes. However, they both had the same accuracy. If we recall the MISO experiment that I conducted on the official functional progression labels, we were able to obtain a “perfect model” on our validation/test data. It is a bit underwhelming that we weren’t able to achieve the same thing on this experiment, despite having the understanding that structural progression is more difficult to determine than the functional progression.

*Structural Progression Indicator Label Experiments:*

Now let’s move on to the hardest label in the project to predict. If we can recall, I attempted to predict this label utilizing an RBF, and there was basically no success in those experiments. But I am trying to take another crack at it again, hoping that the sequence processing of the RNN and LSTM can fit this data better than the RBF. The best RBF model I found was able to achieve a 61% accuracy, 60% precision, 77% recall, and 67% F1 score, so if the RNN or LSTM can beat that I would be happy. It is time to run the experiment now.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Model | Hyperparameters | Accuracy | Precision | Recall | F1 |
| RNN Base | Hidden Size: 24  Num Layers: 1 | 0.506 | 0.433 | 0.813 | 0.565 |
| RNN Grid Search | Hidden Size: 1  Num Layers: 1 | 0.654 | 0.548 | 0.719 | 0.622 |
| LSTM Base | Hidden Size: 24  Num Layers: 1 | 0.395 | 0.351 | 0.625 | 0.449 |
| LSTM Grid Search | Hidden Size: 5  Num Layers: 2 | 0.630 | 0.522 | 0.75 | 0.615 |

*Table 2.4: MIMO IGPS structural progression analysis over the progression indicator label*

Well, the experiment didn’t necessarily go the way I had hoped, but that is okay. It was expected that the base models would have poor performance, but the models found through grid search still leave something to be desired. One thing I do respect about the grid search models is that smaller models were shown to be “better.” However, the RBF model achieved a better recall score, and as we have discussed, that score is probably the most important for this problem. Because of this, it is hard to make the claim that the RNN and LSTM are better for classifying structural progression (so far all of the methods are not good for classifying structural progression though). To me, since the RBF, RNN, and LSTM all seems to max out at similar scores, it is possible there is some inconsistencies in the data relating to the definition structural progression which could be inducing a large amount of noise which make it impossible to actually get a strong model.

1. Reflection:

In this project, our main goal was to explore the RNN and LSTM neural networks on a dataset of our choosing. We were not only to just implement the RNN and LSTM, we were also tasked with truly understanding how these networks work and were to provide a clear explanation for this. I believe I have met all of the goals that were outlined for this project. I implemented both an RNN and an LSTM utilizing the PyTorch library. I then used these networks to conduct various different computational heavy experiments (grid searches took about 1 hour per) over different feature vectors and labels contained within the IGPS dataset. Some of the experiments showed a decent level of success, while others still left something to be desired. If I had more time, I would have done a couple of things differently. First, I would have implemented the RNN and LSTM from “scratch” in PyTorch instead of using the modules that the amazing people at Meta created. The reason I would have done this is to see if they have it implemented in the same way, and to see if doing it from “scratch” would give the user an advantage or a disadvantage. During my experiments I was producing plots such as the error rate over time and the confusion matrices, however these plots seem to take up a lot of space. Because of this I decided to omit them in the report, I just wanted to discuss what worked and what didn’t and wanted to avoid a massive dump of plots and data. What I have found is that it could possibly be useful to utilize sequential neural networks to classify functional progression. However, for structural progression, if I want to model this, I am going to have to get a lot more creative because the RBF, RNN, and LSTM all capped out at a similar success rate. I still can’t determine if my methods for this are not optimal, or if there is something wrong with the data itself. All in all, I enjoyed learning about the RNN and LSTM and how they could possibly apply to my research. They are kind of tricky to wrap your head around at first, but this is also what I found intriguing about them. Now onto the transformer for my final presentation!