CSE 151B Project Final Report

Alexander Friend
apfriend@ucsd.edu

1 Task Description and Background

1.1 Problem A [0.5 points]

Describe in your own words what the deep learning task is and why it is important. Provide some real-world examples where solving this task can have great impact on our daily life and the society.

The task for this Kaggle competition is to predict the positions of 60 individual vehicles 3 seconds into the future, given an initial 2 second observation of vehicle positions and velocities, as well as data on relative lane positions. This is an important task as autonomous vehicles (AV) are being increasingly rolled out to the public, and are expected to become the future standard of automobile transportation. In order for this future to be realized however, AV's must be able to predict future movement and positions of objects in their visinity with high accuracy in order for AV's to be safer than human drivers.

1.2 Problem B [0.5 points]

Use Google Scholar or other internet resources to research on this task. What type of methods have been examined before? Include some references and discuss their ideas in a few sentences. You can use Bibtex to manage your bibliography.

1.3 Problem C [1 points]

Define the input and output in mathematical language and formulate your prediction task. From the abstraction, do you think your model can potentially solve other tasks beyond this project? If so, list some examples and explain your rationale.

For this predition task, I will be taking in data on input and output positions, velocities, for 60 cars in a specific *scene*, with lane information for each respective *scene*. Each one of thes *scenes* is a data point to be used to predict the output position of the 60 cars.

My goal is to predict the positions of a specific car for 3 seconds into the future, in $\frac{1}{10}$ second intervals, using a prior 1.9 seconds of data (also in $\frac{1}{10}$ second intervals).

Since this prediction task relies on positional and velocity data, it can be generalized to any task that requires prediction of positions using prior positions, velocities, and lane information. In prediction models that do not use the lane information, this task can be further abstracted to predicting future positions, using prior positions and velocities.

This means that such a predictor may be able to be generalized to predict the future position of any moving object, using its prior positions and velocities.

2 Exploratory Data Analysis

Perform exploratory data analysis and report your findings with texts/tables/figures. If you include more exploratory analysis beyond the listed questions that provides insights into the data, you will receive bonus points.

2.1 Problem A [0.5 points]

Run the provided Jupyter notebook for loading the data. Describe the details of this dataset. Your description should answer the following questions:

- **what is the train/test data size?**
- **how many dimensions of inputs/outputs in the raw data?**
- **what are the meanings of these input/output dimensions?**
- **what does one data sample looks like?**

Our data is split into two sets, a training set and test set, title `new_train` and `new_val_in`, respectively. The training dataset contains the following fields:

- `p_in` - the (x,y) position input in the first two seconds (19 time steps)
- `v_in` - the (x,y) velocity input in the first two seconds (19 time steps)
- `p_out` - the (x,y) position output in the next three seconds (30 time steps)
- `v_out` - the (x,y) velocity output in the next three seconds (30 time steps)
- `track_id` - the track_id for each vehicle in the output sequence (30 time steps). This is a unique value for each object in the *scene*, not every object in the whole dataset.
- `scene_idx` - the id of a specific scene, i.e each one of the $205,944$ and $3,200$ training and validation datasets.
- `agent_id` - track id for an agent (car), of which the task is to predict its future position.
- `car_mask` - boolean index for the real car, which matches the first dimension of its corresponding `track_id`, `p_in`, `p_out`, `v_in`, and `v_out`.
- `lane` - (x,y,z) for centerline nodes in this scene (z position is always 0). There may be more than one lane in a scene, however this amount may vary from scene to scene.
- `lane_norm` - (x,y,z) the direction of each lane node (z position is always 0)

The validation dataset contains the same fields as the training set, except it is lacking the `p_out` and `v_out` fields, as these are the fields to be predicted.

Using these datasets, our all possible input data will be the the positions (`p_in`), velocities (`v_in`). Each input position and velocity data is of the shape $60 \times 19 \times 2$, and each output position and velocity data is of the shape $60 \times 30 \times 2$. The dimension represents the 60 agents (AV's) in the datapoints scene. The second dimension represents the number of times each agents' position and velocity is tracked. 19 times for 1.9 seconds for the input data, and 30 times for 3 seconds for the outputs. Finally the third dimension represents the respective x and y positiona and velocity vectors for each agent.

The ID's of each of the 60 vehicles (`track_id`) in validation set, which will be represented by a $60 \times 30 \times 1$ tensor. The first dimension is to represent each individual agent in the vehicle, and each of the 30 values in the second dimension is identical.

Furthermore lane information will include the position of the center of the lanes (`lane_id`), which is of dimensions $K \times 3$, where K is the number of lanes in a scene, and each one of the 3 values represents an x, y, z coordinate. The z coordinates are always 0, while the x, y data can be either negative as well as positive, where negative values indicate that the center of a respective lane is near the edge of the coordinate system. Additionally the direction of each lane (`lane_norm`) is of the dimension $K \times 3$, where K is the number of lanes in a scene, and each one of the 3 values represents an x, y, z coordinate. Similarly to the `lane_id` data, each `lane_norm` has values of 0 for each z

coordinate, however the x, y data can be either positive or negative, representing which direction relative to the lane center that the lane is pointing.

The training data set contains $205,944$ training scenes, and the validation set contains $3,000$ scenes.

Not all of this data may be extremely useful however, as incorporating the lane information will lead to an increase in complexity of any prediction model, which may not be necessary to produce an accurate prediction. The most important information to predict the future positions of an AV will the the input and output positions and velocities, as well as the `agent_id` which is necessary to identify the car wished to be tracked. The lane information, may still be useful in more complex models in order to account for changes in direction, which may be difficult to impossible to achieve using only prior positions and velocities.

2.2 Problem B [0.5 points]

Perform statistical analysis to understand the properties of the data. Your analysis should at least answer the following questions.

- **what is the distribution of input positions/velocity (magnitude) for all agents**
- **what is the distribution of output positions/velocity (magnitude) for all agents**
- **what is the distribution of positions/velocity (magnitude) for the target agent**

Statistic	x-Position	y-Position	x-Velocity	y-Velocity
count	2.280000e+06	2.280000e+06	2.280000e+06	2.280000e+06
mean	2.108387e+02	3.245734e+02	3.604328e-02	-1.889331e-02
std	7.032012e+02	8.485306e+02	1.749104e+00	2.185653e+00
min	0.000000e+00	0.000000e+00	-1.144689e+02	-1.688118e+02
25	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00
50	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00
75	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00
max	4.735943e+03	4.092156e+03	7.909621e+01	1.461265e+02

Table 1: Table of input positions and velocities of sample

Statistic	x-Position	y-Position	x-Velocity	y-Velocity
count	3.600000e+06	3.600000e+06	3.600000e+06	3.600000e+06
mean	2.109253e+02	3.245277e+02	3.481067e-02	-1.777745e-02
std	7.034579e+02	8.483126e+02	1.797118e+00	2.223628e+00
min	0.000000e+00	0.000000e+00	1.008539e+02	-1.215008e+02
25	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00
50	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00
75	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00
max	4.736574e+03	4.092117e+03	8.774132e+01	1.484342e+02

Table 2: Table of Output positions and velocities of sample

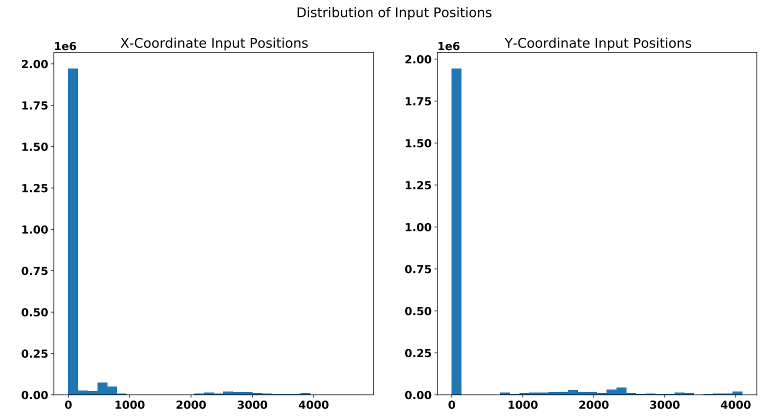

Figure 1: Histogram of the X and Y input positions from a subsample of 2000 training files

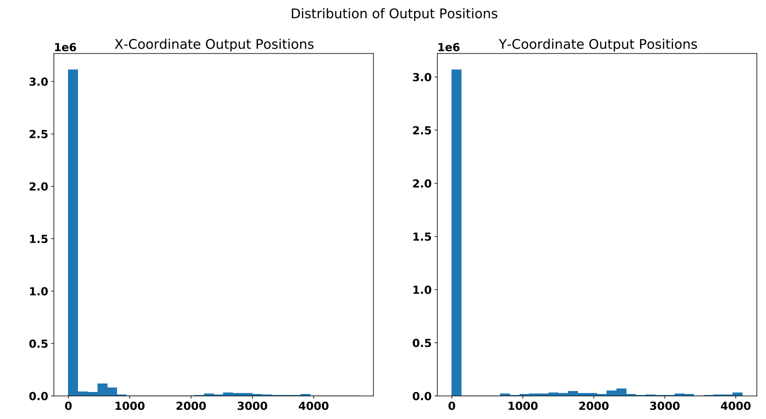

Figure 2: Histogram of the X and Y output positions from a subsample of 2000 training files

Analyzing these inputs we find that the vast majority of (x, y) input and output positions are at position 0, as seen in Figure 1, Figure 2, and Table 1 above. Interestingly, the x-positions seem to be more closely clustered around 0, while the y-positions have more variance, meaning that most cars are moving mostly in the y-dimension, with less overall travel in the x-dimension. This trend carries through the beginnning and end of the scene (the input and output data).

Looking at only the magnitudes of velocities shows the same trend, with the vast majority of velocities being 0, however there is also slightly more variance in the distribution of velocties in the y-dimension compared to the x-dimension.

2.3 Problem C [1 points]

Process the data to prepare for the prediction task. Describe the steps that you have taken to process the data. Your description should at least answer the following questions.

- **Did you use any feature engineering? If yes, how did you design your features? Explain your rationale.**
- **How did you normalize your data? Why did you choose this normalization scheme?**
- **Did you use the lane information provided in the dataset. If yes, how did you exploit this information.**

In order to prepare my data, I used batch normalization for each input (x, y) position and velocity vectors. This was in order to ensure that my models would predict the future positions based on relative positions and velocities in each respective scene, rather than the absolute position and velocity. This is based on the idea that actual people driving move in react based on relative positions and speeds of objects in their area, not based on absolute geospatial coordinates.

At this point I have not used any lane information, as my models are still on the simpler side due, however this is something I wish to imncorporate in future work. Using this information in a model would require normalization relative to the center of each lane as well. For more detail see section 6.

3 Deep Learning Model

3.1 Problem A [1 Points]

Describe the deep learning pipeline for your prediction task and answer the following questions.

- **What are the input/output that you end up using for prediction after preprocessing?**
- **What is your loss function? If you have multiple alternatives, discuss your ideas and observations.**
- **How did you decide on the deep learning model? Explain the rationale given the input/output.**

I ended up using the positions (`p_in`) and velocities (`v_in`) as a single tensor as input data for my models, with the models predicting the output positions (`p_out`) and velocities (`v_out`) as a single tensor. Only the output position data ended up being used for this specific prediction task however.

As a loss function I used both mean squared error (MSE), and root mean square error (RMSE). I ended up using RMSE as my final loss function however, as this is the same loss being used by the Kaggle competition leaderboard. Using RMSE as a loss function allowed me to better judge how my models were performing relative to others in the competition.

I used single layer and 2 multilayer regression neural networks, as well as a Long Short-Term Memory (LSTM) neural network as models for my prediction task. The multilayer regression model used 3 linear layers. I chose the single and multilayer regression models as they are simple models to start out with, making developing better models easier to do. They also are well suited for predicting future position as an equation of starting position and time. An LSTM approach was chosen since a RNN architecture allows for using previous predicted sequences to predict future sequences, which is useful for predicting multiple positions in a time series. I chose a LSTM specifically as it is able to remember previous hidden states, which alleviates the vanishing gradient problem often seen in RNN architectures.

3.2 Problem B [1 Points]

Describe all the models you have tried to make predictions. You should always start with simple models (such as Linear Regression) and gradually increase the complexity of your model.

- **Use an itemized list to briefly summarize each of the models, their architecture, and parameters, and provide the correct reference if possible.**
- **If you end up designing your own model architecture, include a picture/sketch of your model architecture. Explain why you choose such a model.**
- **Describe different regularization techniques that you have used such as dropout and max-pooling in your model.**

You can also use mathematical equations to explain your prediction logic.]

In addition to the single layer linear regression neural network, 2 multiple layer nerual network, and LSTM models, I used a non deep-learning model based in simple Newtonian physics, using average velocity and a starting position to predict future positions.

1. "Physics Model" (1)
   (a) This model uses kinematic equations from physics to predict future position. Specifically it computes the average input velocity of each agent in a scene and uses the final input position coordinate to predict the position of each future time step.
   (b) It can be summarized by the equations:

$$\vec{r}_{out}^{(t)} \approx \vec{r}_{in}^{(19)} + \left(\frac{1}{N} \sum_{i=1}^{N} \vec{v}^{(i)} \right) \cdot \Delta t$$

      i. Where $\vec{r}_{in}^{(19)} \in \binom{x}{y}$ is the position vector in $2d$ space at the 19th time step in the input data.
      ii. $\vec{r}_{out}^{(t)} \in \binom{x}{y}$ is the position vector in $2d$ space at the tth time step in the output data.
      iii. $\vec{v}^{(i)} \in \binom{v_x^{(i)}}{v_y^{(i)}}$ is the velocity vector in $2d$ space at input time step i
      iv. Δt represents each consecutive time step for the prediction task, which is up to 3 seconds predicted at a rate of $\frac{1}{10}$ per second. $\Delta t \in [1, 30]$
      v. N corresponds to each the number of time step in the training data, in this case $N = 19$ for each position and velocity vector measured for 1.9 seconds at a rate of $\frac{1}{10}$ per second
2. Single Layer Linear Regression Neural Network (2)
   (a) A single layer PyTorch linear regression model, which takes the input position and velocity tensors and predicts for 19 time steps and predicts the output position and velocity tensors.
   (b) The input layer is of shape $(1, 60 \times 19 \times 4)$, and the output layer is of shape $(1, 60 \times 30 \times 4)$
   (c) Input is a tensor of shape $(\text{batch size}, 60, 19, 4)$ reshaped to a tensor of dimensions $(\text{batch size}, 60 \times 19 \times 4)$ when fed to the model
      i. batch size is the number of scenes in a mini-batch duriung each training iteration in an epoch.
      ii. The second dimension of this tensor represents the 60 individual agents tracked in each respective scene in a mini-batch
      iii. The third dimension represents the 19 time steps measured in the 1.9 second period.
      iv. The first 2 elements of the $4th$ dimension of this tensor are the position coordinates in $2d$ space, and the last 2 elements of the tensor's $4th$ dimension are the velocity coordinates in $2d$ space for each respective agent in the mini-batch.
   (d) Output is a tensor of shape $(\text{batch size}, 60 \times 30 \times 4)$ to a tensor of shape $(\text{batch size}, 60, 30, 4)$.
      i. batch size is the number of scenes in a mini-batch duriung each training iteration in an epoch.
      ii. The second dimension of this tensor represents the 60 individual agents tracked in each respective scene in a mini-batch
      iii. The third dimension represents the 30 time steps measured in the 3 second period.
      iv. The first 2 elements of the $4th$ dimension of this tensor are the position coordinates in $2d$ space, and the last 2 elements of the tensor's $4th$ dimension are the velocity coordinates in $2d$ space for each respective agent in the mini-batch.
   (e) Position predictions are made by selecting the first two elements from the $4th$ dimension of each tensor, where the `agent_id` in the corresponding $2nd$ dimension of each tensor matches the desired `agent_id`'s from the validation dataset
   (f) See Figure 3 in Appendix A for a visualization of this model
3. Multiple Layer Linear Regression Neural Network (3)
   (a) A PyTorch neural network with linear input and output layers, and a hidden activation function layer, which takes the input position and velocity tensors and predicts for 19 time steps and predicts the output position and velocity tensors.
   (b) The output and input layers are the same shape as of model 2

   (c) The single hidden layer uses the *ReLU* activation function, which was chosen as this function is generally the best performing activation function for neural networks, and it allows for non positive neurons in the hidden layer are activated, since these are the only non-zero outputs from the *ReLU* activation function.[1]
   (d) The hidden layer is of dimension 1024.
   (e) Input and Output tensors are the same shape as model 1
   (f) Position Predictions are selected in the same manner as model 1
   (g) See Figure 4 in Appendix A for a visualization of this model

4. Multiple Layer Linear Regression Neural Network (4)
   (a) A PyTorch neural network with linear input and output layers, and 3 hidden activation function layers, as well as 3 hidden linear layers, which takes the input position and velocity tensors and predicts for 19 time steps and predicts the output position and velocity tensors.
   (b) The output and input layers are the same shape as of model 2
   (c) The hidden layers consist of an *ReLU* activation layer of dimension 2048, fed to a linear layer which takes in
   (d) The activation function between the first 3 layers is *ReLU*, the motivation behind this choice is the same as explained for model 3. The final hidden
   (e) The hidden layer is of dimension 1024.
   (f) Input and Output tensors are the same shape as model 1
   (g) Position Predictions are selected in the same manner as model 1
   (h) See Figure 5 in Appendix A for a visualization of this model

5. Long Short-Term Memory (LSTM) and CNN Model (5)
   (a) PyTorch LSTM layers with a final convolutional layer to map to the output tensor.
   (b) The LSTM layers consist of an input layer of shape $(60 \times 4, 1)$ with 2 hidden layers of shape $(2048 \times 4, 1)$
   (c) The Convolutional layer has 240 channels and takes in the inputs of the hidden layer, of shape $(2048 \times 4, 1)$, and returns them to an output layer of shape $(60 \times 4, 1)$
   (d) Position Predictions are selected in the same manner as model 1

4 Experimental Design

4.1 Problem A [1 points]

Describe how you set up the training and testing design for deep learning. Answer the following questions:

- **What computational platform/GPU did you use for training/testing?**
- **How did you split your training and validation set?**
- **What is your optimizer? How did you tune your learning rate, learning rate decay, momentum and other parameters?**
- **How did you make multistep (30 step) prediction for each target agent?**
- **How many epoch did you use? What is your batch-size? How long does it take to train your model for one epoch (going through the entire training data set once)?**

Explain why you made these design choices. Was it motivated by your past experience? Or was it due to the limitation from your computational platform? You are welcome to use screenshots or provide code snippets to explain your design.

The platform I am currently working in is my local machine running Ubuntu 20.04 with a 4-core 4-thread Intel i7-7600k CPU running at 4.2 GHz, a GTX 1070 GPU with a max clock speed of 1721 MHz and 8 GB of GDDR5 memory, and 32 GB of 3000 MHz DDR4 memory. Since the start of this project, I have upgraded my local machine from 16 GB to 32 GB of RAM.

I used all $202,944$ training data sets, with the remaining $3,000$ training data sets set aside as a test set. This test set was randomly sampled to be the same size as the batch size for each model I implemented, with the exception of the Since this model was not a deep learning model and did not have any requirements for batch size, the entire test set of $3,000$ scenes was used to test the loss of this model.

I used Adam with a learning rate of 0.3 as my optimizer since it implements an updating momentum, based on the gradient, meaning that it will converge more quickly to an estimate of the gradient. Additionally, Adam utilizes a mini-batch approach to approximating the gradient, similar to Stochastic Gradient Descent (SGD), which allows for better parallelization, and generalization, as well as the added benefit of using less memory, potentially increasing computational performance.

For my sing average velocity, I predicted each time step by multiplying the average velocity from the $N = 19$ time steps in the input data by the amount of time elapsed since the last tracked input, and then added this to the last tracked position. For example the first and last time steps at $t = 0.1$ and $t = 3$ seconds after the last input data, respectively:

$$\vec{r}_{out}^{(1)} \approx \vec{r}_{in}^{(19)} + \left(\frac{1}{N} \sum_{i=1}^{N} \vec{v}_i \right) \cdot 0.1$$

$$\vec{r}_{out}^{(30)} \approx \vec{r}_{in}^{(19)} + \left(\frac{1}{N} \sum_{i=1}^{N} \vec{v}_i \right) \cdot 3$$

The deep learning models would each produce outputs predicting the positions and velocities for each of the 30 times steps into the future. Extracting the predictions for these was as simple as just extracting the position elements from the outputted tensor.

Both models outputted tensors predicting the positions for all agents in a scene. So the element corresponding to the desired agents in the validation set were selected, as described further in Section 3.2.2.e

Model 2 and model 3 were trained for 1, 5, 10 and 20 epochs, with batch sizes of 128, 64, 32, and 16. Model 3 was also trained with batch sizes of 128, 64, 32, and 16, however it was only trained for 1, 5, and 10 epochs as the loss of that model did not decrease after any more training. Model 5 was only trained for 5 epochs with a batch size of 128 due to a lack of time (see Section 6 for more details on future plans).

5 Experiment Results

5.1 Problem A [1 points]

Select a few representative models of yours and report the following results by comparing different models.

- **Use a table to compare the performances of different model designs. What conclusions can you draw from this table.**
- **Provide an estimate of training time (in flops or minutes) for different models. What did you do to improve the training speed?**
- **Count and report the number of parameters in different models.**

5.2 Problem B [1 points]

Play with different designs of your model and experiments and report the following for your best-performing design:

- **Visualize the training/validation loss (RMSE) value over training steps (You should expect to see an exponential decay).**
- **Randomly sample a few training samples after the training has finished. Visualize the ground truth and your predictions.**
- **Your current ranking on the leaderboard and your final test RMSE.**

6 Discussion and Future Work

6.1 Problem A [1 points]

Analyze the results and identify the lessons/issues that you have learned so far. Briefly answering the following questions.

- **What do you think is the most effective feature engineering strategy?**
- **What techniques (data visualization/model design/hyper-parameter tuning) did you find most helpful in improving your score?**
- **What was your biggest bottlenect in this project?**
- **How would you advise a deep learning beginner in terms of designing deep learn- ing models for similar prediction tasks.**
- **If you had more resources, what other ideas would you like to explore?**

A Appendix

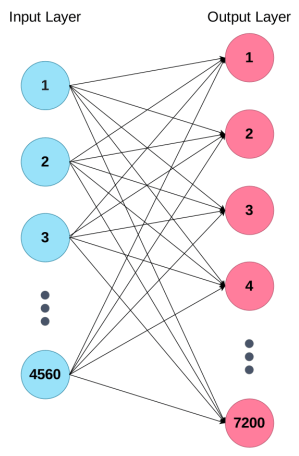

Figure 3: Diagram of single layer model 2

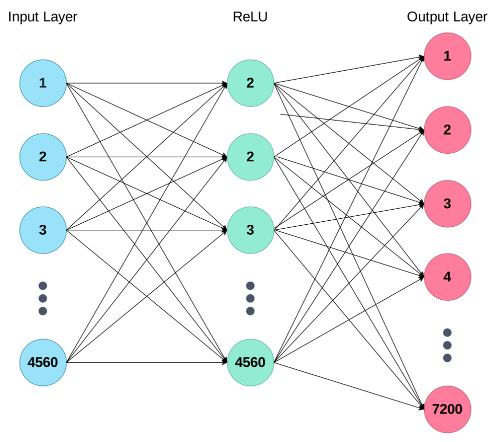

Figure 4: Diagram of single layer model 3

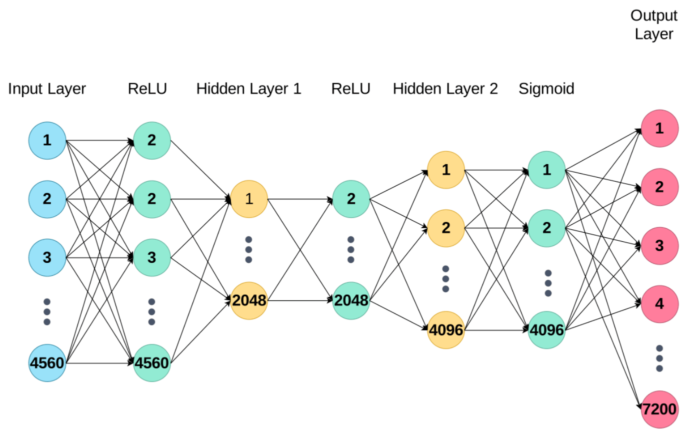

Figure 5: Diagram of single layer model 4

`https://github.com/apfriend/cse151b-kaggle.git`

References

[1] Siddharth Sharma &Simone Sharma &Anidhya Athaiya(2020) ACTIVATION FUNCTIONS IN NEURAL NETWORKS. In G. Tesauro, D.S. Touretzky and T.K. Leen (eds.), *International Journal of Engineering Applied Sciences and Technology*, Vol. 4, Issue 12, pp. 310–316. [Online]. Available: `https://www.ijeast.com/papers/310-316,Tesma412,IJEAST.pdf`

[2] Bower, J.M. & Beeman, D. (1995) *The Book of GENESIS: Exploring Realistic Neural Models with the GEneral NEural SImulation System.* New York: TELOS/Springer–Verlag.

[3] Hasselmo, M.E., Schnell, E. & Barkai, E. (1995) Dynamics of learning and recall at excitatory recurrent synapses and cholinergic modulation in rat hippocampal region CA3. *Journal of Neuroscience* **15**(7):5249-5262.