Autonomous Vehicle Motion Forecasting Challenge

Amogh Patankar, James Lu, Garrett Birch, Matin Ghaffari

Introduction

Team Introduction + Summary

Team Remote Kings
Members:

- Amogh Patankar (Data Science, Junior)
- James Lu (Data Science, Junior)
- Garrett Birch (Data Science, Junior)
- Matin Ghaffari (Data Science, Junior)

- We used a multi-layer perceptron to forecast the positions of autonomous vehicles in a short time span, and learned various different lessons along the way.

Background and Goal(s)

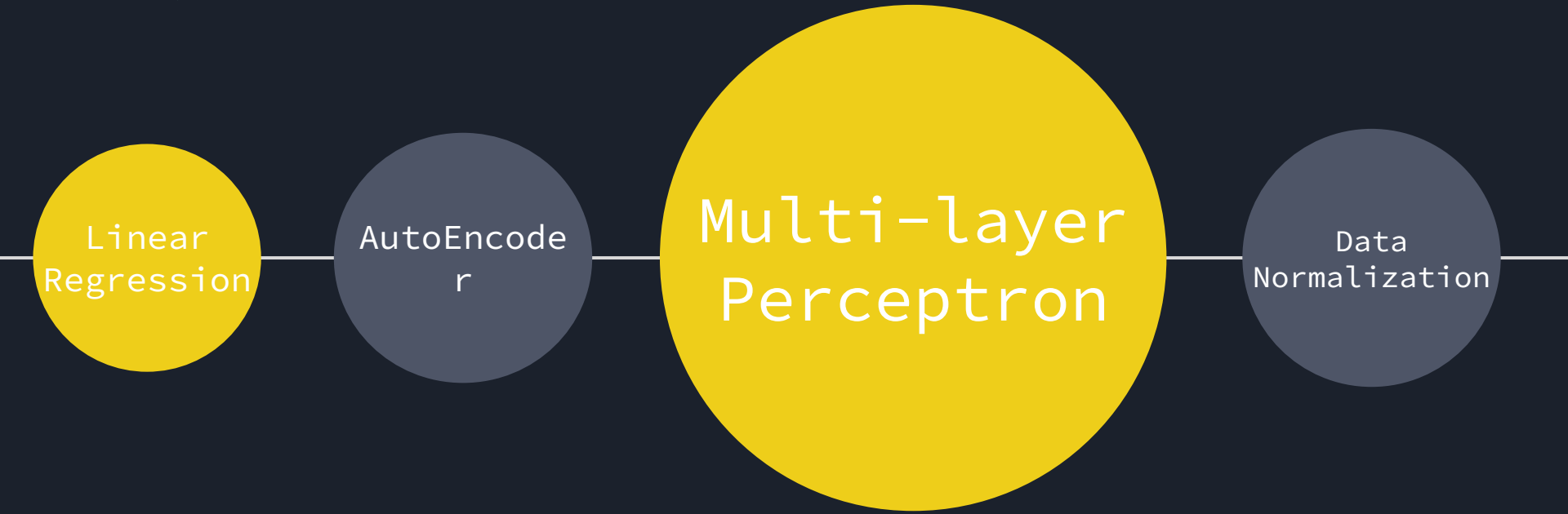
- This challenge is prevalent because tech and car companies are using deep learning to bring us closer to a fully self-driving future.

- Create a deep learning model to predict positions of autonomous vehicles; given a 5 seconds worth of positions, predict the next 6 seconds of positions.

Technical Standpoint

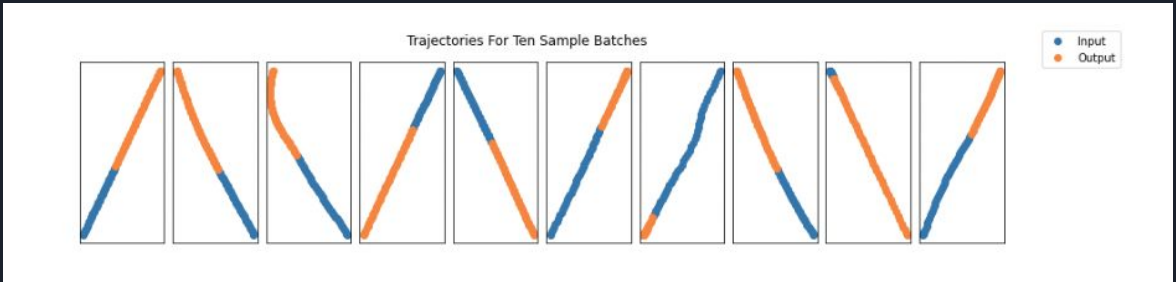
- Neural networks can model a function f.
- 10 Hz frequency, one second --→ ten time steps.
- Input: Positions at 50 time steps, defined as $x_1 \dots x_{50}$ and $y_1 \dots y_{50}$
- Output: Predictions of positions at 60 time steps, defined as $x_1 \dots x_{60}$ and $y_1 \dots y_{60}$

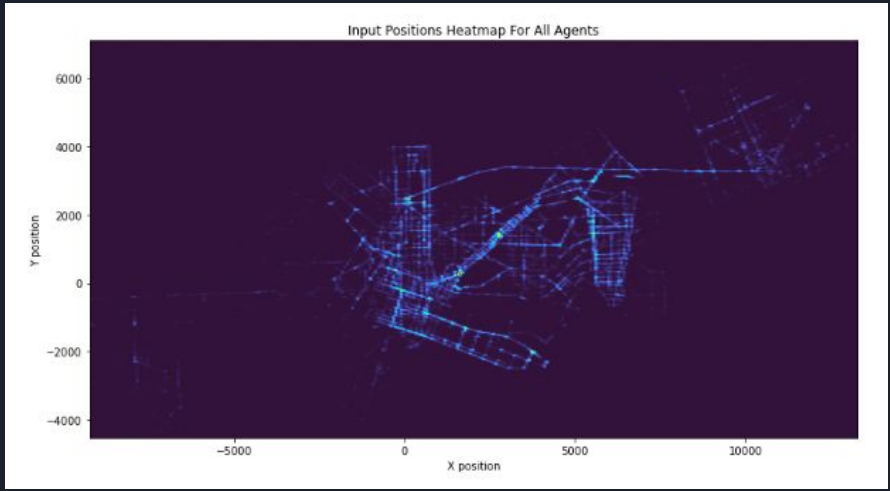
Key Words

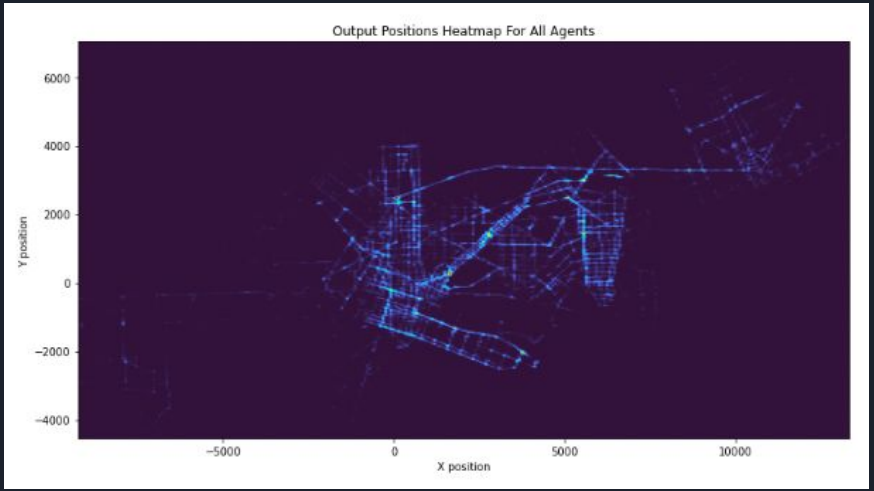

Linear Regression
AutoEncoder
Multi-layer Perceptron
Data Normalization

Methodology

Exploratory Data Analysis







Data Processing

Datasets

- Initially we used the given Argoverse dataset and created our dataloaders from that
- We ended up creating our own dataset class so we could generate a validation set using get_city_trajectories

Training

- Initially our train loop followed this sequence:
  - For each city
    - Generate the necessary data loaders for train / validation set
    - Train the model on this city for n epochs
    - Write the predictions to submission.csv
  - Eventually we realized that our model was writing predictions before seeing all the cities

- We ended up changing the way we trained our model, but we will go over that more in detail later

Feature Selection and Engineering

Feature Selection

- The only features were x and y coordinates/positions, and we need both to predict future positions, hence no feature selection.

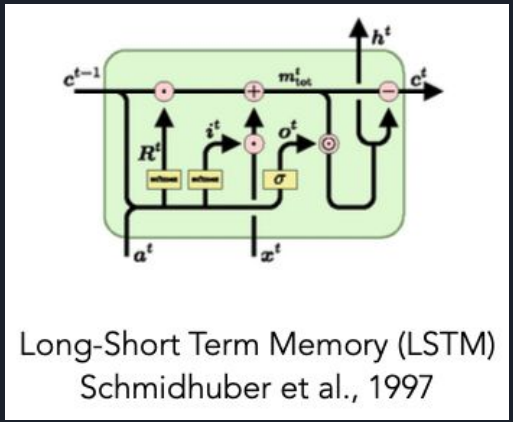
Feature Engineering

- Idea 1: Create a new feature velocity, but then realized that velocity is derivative (literally *and* mathematically) of position.

- Idea 2: Data augmentation- insert new data between time steps; could've helped the predictive capabilities of our model.

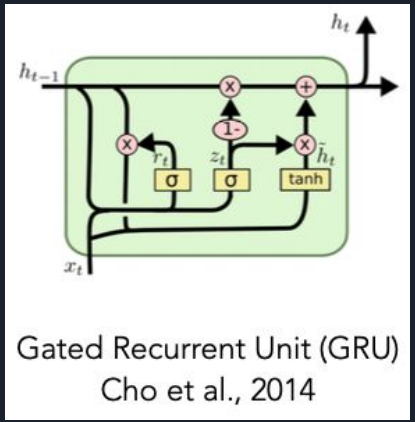
Architectures

- LSTM: LSTMs are useful for time-series and sequence data, and to some extent, we could leverage this built in capability for our task at hand.



Long-Short Term Memory (LSTM)
Schmidhuber et al., 1997

- GRU: The GRU architecture is a concept that is akin to the LSTM architecture, but contains fewer parameters, and lacks an output and forget gate.



Gated Recurrent Unit (GRU)
Cho et al., 2014

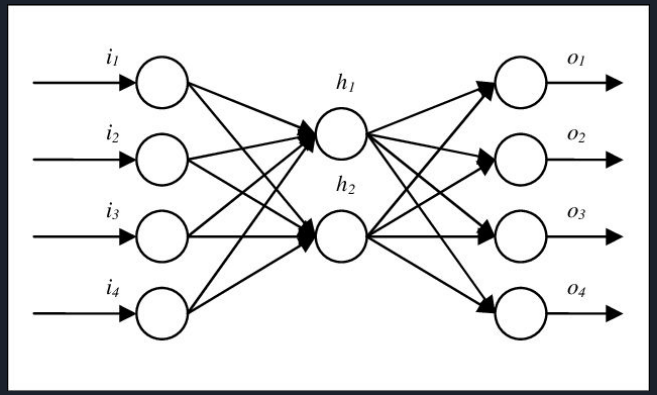
Architectures (cont'd)

Architecture	MSE
Long Short-Term Memory (LSTM)	~ 81,000
Gated Recurrent Unit (GRU)	~ 80,000

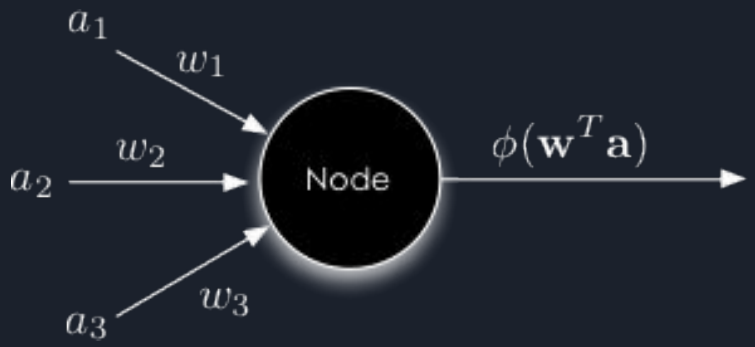
Architectures (cont'd)

Plan B: Use our intuitions about our data and try a simpler model.

- Multi-Layer Perceptron (AutoEncoder): Basic neural network, input layer, hidden layer ("compress"), and output layer.



- Linear Regression:
  - sklearn: it was a quick way to know if we were headed in the right direction.



Architectures (cont'd)

Architecture	MSE
Long Short-Term Memory (LSTM)	~ 81,000
Gated Recurrent Unit (GRU)	~ 80,000
Multi-Layer Perceptron (AutoEncoder)	~ 184
Linear Regression (sklearn)	~ 21

Architectures (cont'd)

Plan C: Modify the existing linear regression to attempt to improve it.

- Linear Regression:
  - PyTorch: We wanted to see if we can implement a version that is better suited for our dataset specifically.

Architectures (cont'd)

Architecture	MSE
Long Short-Term Memory (LSTM)	~ 81,000
Gated Recurrent Unit (GRU)	~ 80,000
Multi-Layer Perceptron (AutoEncoder)	~ 184
Linear Regression (sklearn)	~ 21
Linear Regression (PyTorch)	~ 102

Architectures (cont'd)

Plan D: Focus on one model using our knowledge of our data, and work on it endlessly.

- Multi-Layer Perceptron: We moved away from our autoencoder- decided a combination of MLP and linearity was probably the way to go.
- Linear layers in a multi-layer perceptron

Architectures (cont'd)

Better! Combining linearity and a multi-layer perceptron seemed like a good idea.

Architecture	MSE
Long Short-Term Memory (LSTM)	~ 81,000
Gated Recurrent Unit (GRU)	~ 80,000
Multi-Layer Perceptron (AutoEncoder)	~ 184
Linear Regression (sklearn)	~ 21
Linear Regression (PyTorch)	~ 102
Multi-Layer Perceptron	~ 35

Experiments and Hyperparameter Tuning

Data Processing/Training (revisited)

Improved training method

- Our initial training method was writing results before seeing all the cities
- We created a train loop such that one epoch was a single pass through all the cities
- For each epoch we shuffled the order in which the cities were trained, as well as used dataloader shuffling
    - made our model more robust
- After *n* epochs we would write the predictions ensuring that our model has learned from all the data

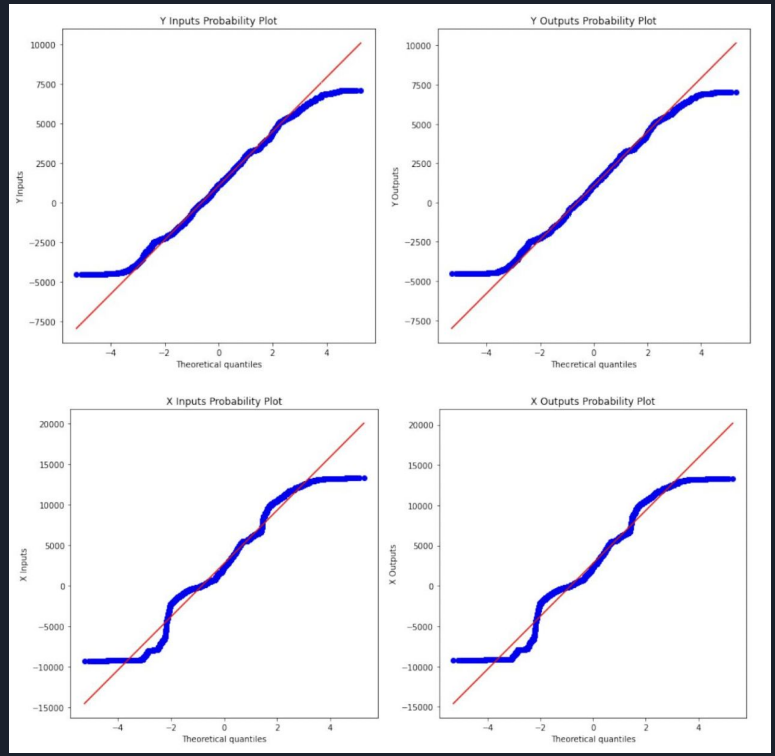
Scoring

Architecture	MSE
Long Short-Term Memory (LSTM)	~ 81,000
Gated Recurrent Unit (GRU)	~ 80,000
Multi-Layer Perceptron (AutoEncoder)	~ 184
Linear Regression (sklearn)	~ 21
Linear Regression (PyTorch)	~ 102
Multi-Layer Perceptron (MLP)	~ 35
MLP (with changed training)	~ 30

Standardization/Normalization

- We initially tried standardizing the data. However, upon further data exploration via qq-plots we learned that the data does not follow a Gaussian distribution, which explains why it produced poor results with our model. This prompted us to experiment with normalization and scaling.

- We then experimented with min-max scaling but it produced worse results than our model without normalization. After investigating why this might be, we were reminded that normalization is highly affected by outliers. Since we knew our data has a healthy amount of outliers based off our EDA, we ultimately decided to not normalize.



Regularization

- We experimented with the following regularization techniques: L1, L2/weight decay, early stopping, and dropout.

- We initially experimented with these techniques when implementing more complex models such as LSTM and GRU since our losses were fluctuating.

- Ultimately, when we switched to MLP we found our losses were decreasing smoothly and our model didn't have indications of overfitting. Because of this, these regularization techniques barely improved, if not worsened our model, which is why we didn't use them in our final model.

Data Transformation

- In an effort to improve learning speed and learning effectiveness we transformed our data to shrink the feature space
  - To do this we subtracted the first coordinate from all of the other coordinates for each agent in the dataset
  - This gave us a much smaller range of values, and a measurement of distance from the origin
  - To reverse this transformation we simply added the first point of the sequence back into all other coordinates
- This transformation greatly increased our model accuracy and training speed

Optimizers

SGD

- We began our experiments using stochastic gradient descent
- We quickly realized that we needed a dynamic learning rate for this problem
    - Our models were training extremely slow and were often getting stuck in local minima

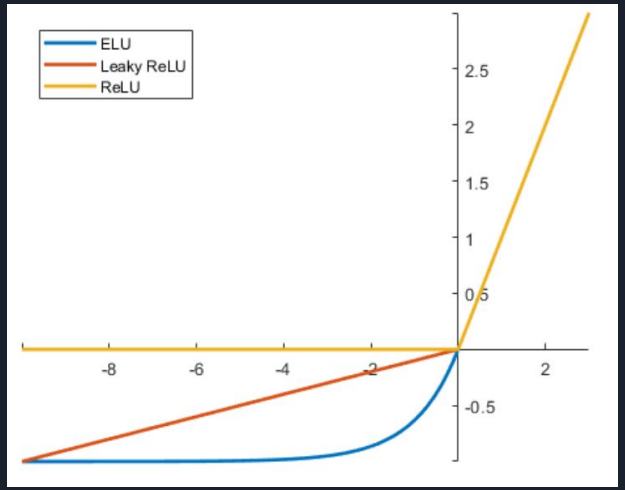
Adam

- After doing some research we found that Adam was the most commonly used optimizer for non convex problems
- It resulted in faster convergence and smaller training and validation losses

Learning Rate Decay

- To further hone in on a good learning rate throughout all the epochs, we used learning rate decay to enforce making good steps towards the end of training.
- Adam already has an adaptive learning rate, however tuning the learning rate decay resulted in better performance.
- Our final parameters for the learning rate decay were gamma (γ) = 0.9, and step_size = 25 epochs.

Activation Functions



- We experimented with ReLU, Tanh, Sigmoid, Leaky ReLU, and ELU
  - Sigmoid and Tanh activation functions performed very poorly, which we realized is attributed to vanishing gradient problem.
  - We then realized it would be better to use ReLU to avoid the vanishing gradient problem and because the weights aren't nearly large enough to cause a exploding gradient problem.
  - Next we experimented with Leaky ReLU and ELU to see if they may perform better than ReLU.
  - We found they indeed outperformed ReLU, as it mitigated sparse gradients. We tuned the Leaky ReLU parameter (0.01, 0.05, 0.1, 0.15, and 0.2) and found that 0.05 was best, however ultimately we found ELU performed the best overall.

Batch Size [Tuning] and Number of Epochs

- We tuned these in parallel, due to time constraints.
- Obviously, using more and more epochs makes the model more accurate, but we settled on an optimal number for our model (200).
    - 1) Overfitting as we increase epochs, converge slowly!!
    - 2) Computationally impractical
- Batch size was a little trickier, as it's more impactful towards our data. We tried 32, 64, 128 and 256, and settled on 128.

Scoring

Architecture	MSE
Long Short-Term Memory (LSTM)	~ 81,000
Gated Recurrent Unit (GRU)	~ 80,000
Multi-Layer Perceptron (AutoEncoder)	~ 184
Linear Regression (sklearn)	~ 21
Linear Regression (PyTorch)	~ 102
Multi-Layer Perceptron	~ 35
MLP (with changed training)	~ 30
MLP (post-hyperparameter tuning)	~ 18

Discussion

Lessons Learned

- Deep learning theory vs application is very different.
    - eg. Debugging in a software different than a deep learning; lot more time consuming, as DL models take a lot more time due to complexity.
- Tune hyperparameters *with intent* not blindly changing parameters randomly.
- “Story of your data” can lead you in the right direction, and using complex models doesn’t guarantee they’re the right solution.

Future Work

- Our dataset only contained two features; given a dataset with more features, we can do feature selection, and also perform feature engineering.

- Going back to the LSTM and GRU, and attempting to fix and optimize those models.

References

- Dolhansky, B. (2014, October 28). *Artificial Neural Networks: Linear regression (part 1)*. BRIAN DOLHANSKY. Retrieved June 5, 2022, from http://www.briandolhansky.com/blog/artificial-neural-networks-linear-regression-part-1
- *Figure 1.8: Illustration of output of Elu vs relu vs Leaky Relu...* (n.d.). Retrieved June 6, 2022, from https://www.researchgate.net/figure/Illustration-of-output-of-ELU-vs-ReLU-vs-Leaky-ReLU-function-with-varying-input-values_fig8_334389306
- Jamil, D. (2021, April 2). *Lets discuss encoders and style transfer*. Medium. Retrieved June 5, 2022, from https://medium.com/analytics-vidhya/lets-discuss-encoders-and-style-transfer-c0494aca6090
- Kain, N. K. (2019, December 2). *Understanding of multilayer perceptron (MLP)*. Medium. Retrieved June 5, 2022, from https://medium.com/@AI_with_Kain/understanding-of-multilayer-perceptron-mlp-8f179c4a135f