1 Introduction

The last decade has seen massive growth in the field of Autonomous Driving, primarily due to a) proliferation of graphical processing unit(GPU), b) several projects like Google(Waymo) [1], Berkeley-DeepDrive [2], Apollo [3], have made their datasets open-source making it easier for people to work on these data and achieve better performance gains.

Training a deep neural network(DNN) forms the core of making a car autonomous. By using supervised learning, one can achieve reliable results as it gives greater control at each stage of training. The data-driven approach collects data in advance and labels it appropriately. It can then be fed to the DNN using supervised learning algorithms to train the best model possible.

Ever since the discovery of Alexnet in 2012 [4], the convolutional neural network(CNN) and deep learning(DL) are preferred choices to analyse images. However, it is well known that the camera sensors are susceptible even to a slight change in weather conditions. Sensors like radar [5], LIDAR [6], ultrasonic[7], depth camera give additional depth information for obstacle detection. These values then are fused with the camera images to make data fusion possible.

Even though there are some public data available, it is still not enough to reliably train a DNN. Then there is the cost of building an autonomous car. Fortunately, the last years have seen growth in reliable simulators which helped massively to collect data to help explore this field of research. To name a few simulators that are being actively used – LGSVL [8], Nvidia Drive [9], Carla [10], CarMaker [11]. In this thesis, the LGSVL simulator is used.

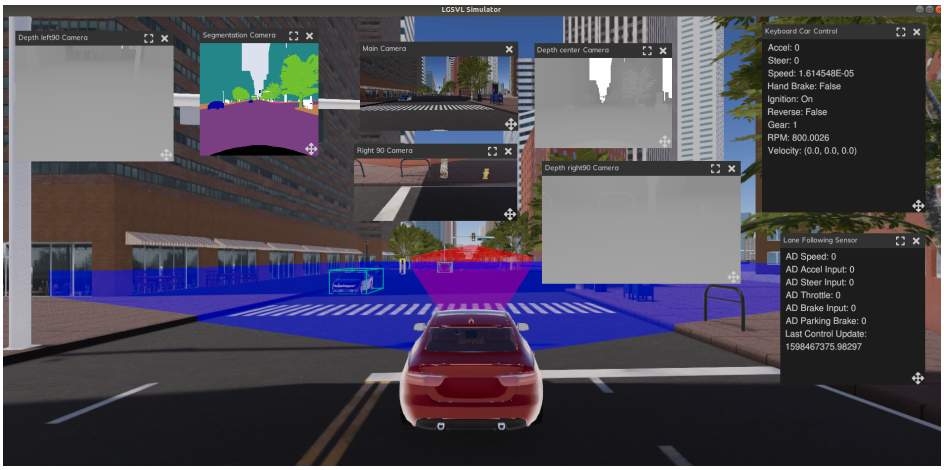

Figure 1.1: LGSVL[12] simulator active with all sensors

The LGSVL simulator allows the use of different sensors with minimal effort. The data from different sensors are published through websocket. So to capture these data, we need an interface/protocol which can understand the sent data's message types and enable the receiving node/programme to store them. However, the data from each sensor arrives at different rates. Hence it is necessary to collect and synchronise them in the order of their arrival before storing, to not lose their integrity and

thereby prevent corrupting the dataset. Robotic operating system(ROS) [13] and its functionalities fulfil this purpose. It allows seamless transfer of simulator's data by subscribing to it in the form of topics. Then the subscribing node synchronises it as necessary for storage.

So, now the data that resembles real-world is stored locally for later analysis and research.

1.1 Motivation

Though autonomous driving is one of the favourite research areas in mobility, a significant challenge is still the cost associated with integrating all the necessary sensors. Representing the environment around the vehicle(ego vehicle) requires information from all in-car sensors. The resources demanded to make an optimal decision are also a challenge. The motivation for this thesis is to use a simulator, do the required tests and determine whether using a simulator does indeed help in perceiving the environment and accomplish the goal of driving in the real-world.

As briefly mentioned, the high cost of associated sensors such as LIDAR [14], has put off many smaller research groups from implementing into their work. By using a simulator, again at a low cost, we can conduct adequate tests on how different constellation of sensors work, how different modalities interact with each other and what impact these factors have on the overall performance of the DNN.

Finally, implement an end-to-end system which simulates real-world behaviour which can then be applied to future research and make it more robust.

1.2 Goal

The desired goals of this thesis are listed below:

- Building an autonomous driving framework -
  - ROS - use ROS2 to synchronise the data received from the simulator through a rosbrige, use functionalities such as slop and cache, to sort the data according to their received time in order not to scramble the information. During the evaluation, use the same functionalities to send command controls to the simulator.
  - Rosbridge - use a bridge transport protocol that connects the ros on the receiving end to the simulator on the sending end or vice versa.
  - Docker - set up a work environment that is independent of hardware or operating system which allows easy running of the commands for data collection and evaluation.
- Implement an end-to-end neural network architecture which learns to drive by training from image pixels to steering commands. Also, apply state of the art DL techniques to it.
- Implement a system that can efficiently collect and label data.
- Implement and analyse different constellation of sensors with different data fusion techniques.

1.3 Related Work

In 2012, Alexnet [4] used CNNs to do object classification which, then in Computer Vision became the dominated approach for classification. Both Chen *et al.* [15] and Bojarski *et al.* [16] extended [4]'s approach of using CNN and detailed that in addition to classification, CNN can extract features

from images. Then they went on to demonstrate through an end-to-end network(which self-optimises itself based on its inputs) steering angles can be predicted that keep the car in the lane of a road.

In a different field, but using CNN, Sergey Levine *et al.* [17] in 2016 corroborated that it was indeed possible to extract features with CNN and predict motor control actions in *object picking robots*.

Then, Xu *et al.* [18] in the same year with CNN-LSTM architecture showed that using the previous ego-motion events helped predict future ego-motion events. Using CNNs in an end-to-end architecture raised some questions on how it reached its decisions. So in 2017, both [19], [20] did visual analysis after the CNN layers to better understand the module's functionality. Vehicle control is more than just steering control. For smoother control, acceleration and braking are necessary besides steering. Both acceleration and deacceleration are dependent on the user's driving style, lane speed limit and traffic etc. Yand *et al.* [21] used CNN-LSTM architecture and provided the LSTM with feedback speed to determine the velocity of the ego vehicle.

Besides vehicle control, perceiving the environment is necessary for collision avoidance. The RGB colour camera sensors don't provide the depth information which is critical for collision avoidance. Hence, it is essential to fuse other sensors with diverse modalities with RGB to predict an optimal output. Liu *et al.* [22] provided rules in fusing data. They said that it was essential to pick out only vital information and discard other noisy data. They also described the techniques involved in data fusion – early/late fusion, hybrid fusion, model ensemble and joint training. Park *et al.* [23] gave us methods to enhance the features by using feature amplification or multiplicative fusion. Zhou *et al.* [24] detailed how fusing data into CNN affects the overall performance.

Even though the fused dataset gives a performance boost, it performs worse compared to individual modality. The combined fused model overfit more than its counterparts. The fundamental drawback of *gradient descent* in backpropagation causes the networks to overfit. This paper [25] introduced a technique called *gradient blending* to counteract this problem.

Xiao *et al.*[26] applied all the fusion techniques mentioned above with an imitation based end-to-end network[27]. RGB images with depth information(obtained through a different modality) could indeed result in better performing end-to-end network model was their conclusion.

1.4 Contribution

2 Fundamentals

2.1 Machine learning: What and why?

Machine learning is all about of learning from data; gaining knowledge from it. More the data, more the chances to learn. Machine learning was initially thought of as automating redundant human tasks and later developed into something that allowed solving complex mathematical problems. It was seen as an addition to humans than extension of them. Machine learning these days are required to perform tasks that are quite obvious and natural to humans; such as recognising faces in images or perceive the road, environment around the vehicle and make decisions instinctively.

All these attributes require to extend the field of machine learning to extract more knowledge from a given data. The figure 2.1 shows how artificial intelligence(AI) is divided to specific areas – Machine learning(ML) and deep learning(DL).

So, in this chapter, an overview is given on the concepts that are used in the later chapters.

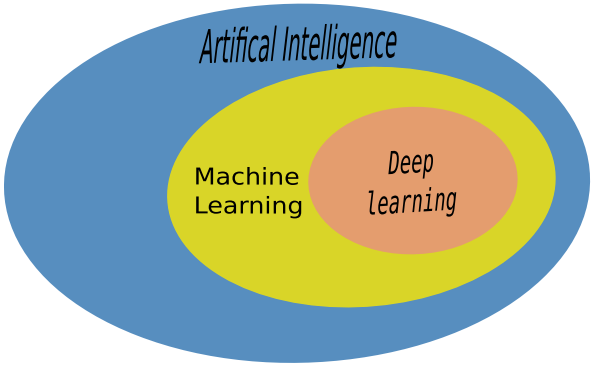

Figure 2.1: Schema of AI, ML and DL

2.1.1 Learning algorithms

Machine learning provides a means to tackle tasks that are complex to solve through fixed programmes and designed by human beings [28]. A learning algorithm is an algorithm which gains the ability to learn from data. A ML algorithm is one that gains the ability to learn from an experience E with respect to some class of tasks T and performance measure P [29].

Tasks, T

The two major tasks in ML are *classification* and *regression*.

In classification related tasks, the system is identify which of k categories an input belongs to. A function $f : \mathbb{R}^n \rightarrow \{1, \ldots, k\}$ is used by the learning algorithm to solve this task. When $y = f(x)$, the model assigns an input described by vector $\mathbf{x}$ to a category identified by numeric code y. There

are other variants of the classification task, for example, where f outputs a probability distribution over classes [30]. Alexnet [4] is one of the examples of classification tasks that performed object recognition.

Regression is similar to classification except that the output is a continuous value. A function f" : $\mathbb{R}^n \rightarrow \mathbb{R}$ predicts a numerical value for some input. Predicting the steering control value is a prime example for regression task.

There are ofcourse other tasks but only classification and regession are used in this thesis. Hence the narrow focus.

Performance measure, P

To evaluate the performance of a ML algorithm, it is must to design quantitative measure of its performance. Usually this performance measure P is specific to the task T. There are two distinct types of measurements – accuracy and error rate.

If the goal is to learn a mapping from inputs x to outputs y, where $y \in \{1, \ldots, C\}$, with C being the number of classes. If $C = 2$, this is called binary classification (in which case we often assume $y \in \{0, 1\}$); if $C > 2$, this is called multiclass classification. If the class labels are not mutually exclusive (e.g., somebody may be classified as tall and strong), we call it multi-label classification [31].

Accuracy is a proportion of how much the model produces the correct outputs. So in the case of binary classification, if the function f predicts a probability densities $\hat{y} \in \{0.3, 0.7\}$, for a ground truth y of value 1, then P is 70% accurate or the error rate is 30%.

It is must that the model is evaluated with a data that it has not seen before. This data *testing set*, gives a good judgement on the performance.

Experience, E

The ML algorithms can be classified into *supervised*, *unsupervised* and *reinforcement* learning based on the kind of experience they are allowed to have. A learning algorithm is allowed to gain experience by going through the *dataset*. A dataset is collection of all the examples for a given task. For example, to classify which category a shown image belongs to has collection of images as dataset [32]. Sometimes datasets are also called as *data points*.

The focus will be on supervised learning in our case. The CIFAR dataset [32] containing images as features has *targets* or *labels* associated with it. In supervised learning(SL), the target functionality is shown to the learning algorithm. A random vector $\mathbf{x}$ explicitly attempts to learn the probability distribution $p(\mathbf{x})$ and predicts $\mathbf{y}$ from $\mathbf{x}$, usually estimating $p(\mathbf{y} \mid \mathbf{x})$.

2.2 Deep Learning

Deep learning is a subset of machine learning. It takes all the algorithms, concepts from the machine learning and narrows the focus to enable a model learn from data to do tasks that involve less human interaction, large number of data and parameters.

2.2.1 Simple neural network

Linear regression is one of the common SL algorithms. It solves the regression problem. For example, if there is vector $\mathbf{x} \in \mathbb{R}^n$ as input and predict a scalar value $y \in \mathbb{R}$ as its output, then in linear

regression, output is a linear function of the input. We can define it as

$$\hat{y} = \mathbf{w}^T \mathbf{x} \tag{2.1}$$

where $\mathbf{w} \in \mathbb{R}^n$ is a vector of parameters.

$\mathbf{w}$ is usually referred to as a set of weights that determine how each feature affects the prediction. A $\mathbf{w}_i$ is simply multiplied with a feature x_i to predict $\hat{y}$. By manipulating the $\mathbf{w}_i$ value, the corresponding feature has an effect on the prediction $\hat{y}$.

A learning algorithm, in this case linear regression, is implemented as a perceptron. It is a single-layer neural network. They generally consists of four main parts – input nodes x_i, weights w_i, bias b_0(if necessary), net sum Σ and an activation function σ. This is shown in the figure 2.2.

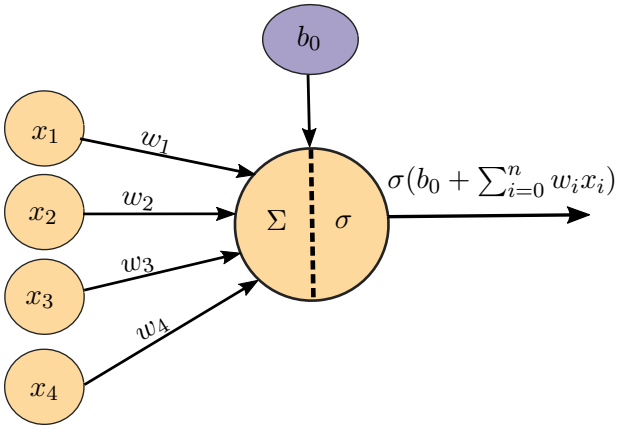

Figure 2.2: A simple neutral network

Activation function

The common activation functions used are Rectified Linear unit(ReLu), Sigmoid, tanh and softmax function. For each type of activation, σ then decides if the input received is relevant or not relevant. To convert linear inputs to non-linear, all that has to be done is to use a non-linear activation function. In the figure 2.3, shows the characteristics of some of the activation functions. For classification

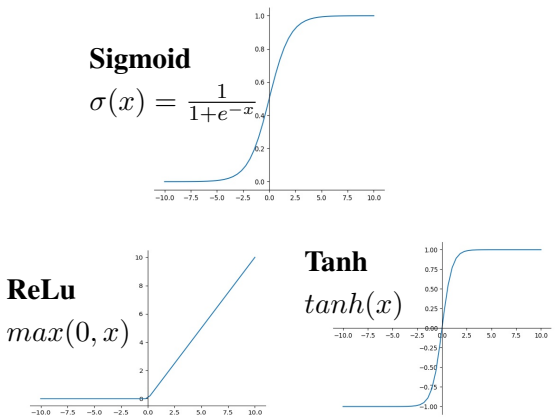

Figure 2.3: Activation functions

tasks, usually the last layer of the networks is equipped with softmax activation layer. This function normalises the output to a probability distribution over predicted output classes.

Multilayer feedforward networks

Deep feedforward networks or multilayer perceptrons are the quintessential deep learning models. Its goal is to approximate function f^*. In the below figure 2.4, information flows from inputs $\mathbf{x}$ to output y using a mapping function $\mathbf{y} = f(\mathbf{x}; \theta)$ where θ are the parameters values which the MLP learns for optimal approximation.

They are called feedforward as there are no feedback connections in which outputs of the model are fed back into itself. Feedforwards networks with feedbacks are called *recurrent neutral networks*.

Feedforwards networks form the core for many commercial applications. For example, the convolutional neural networks used for object detection are a special kind of feedforward networks.

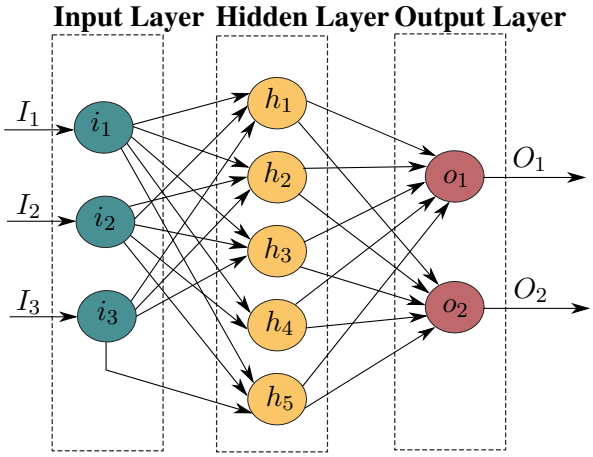

Figure 2.4: Multi layer perceptrons

More the hidden layers, more the depth of the feedforward networks. The width is given by the dimensionality of the hidden layer.

Loss function

As mentioned before, a mapping function f noisily approximates the input x to output y. So, the noise or the deviation from the true value(ground truth) must be kept at minimum. The function that calculates the deviation is called *cost* or *loss* function. It is important to choose the right loss function for a model.

For multi-label classification tasks, *categorical cross-entropy* function is used. For each category, cross-entropy is calculated. The difference between the cross-entropy of training data and the model's predictions is the cost function.

For regression tasks, the models are subjected to loss functions such as *mean absolute error*(MAE), *mean squared error*(MSE) and *mean squared logarithmic error*(MSLE). In MAE, the mean of absolute differences among predictions and expected results are calculated.

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{2.2}$$

In MSE, the mean of squared differences among predictions and true outputs are calculated.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{2.3}$$

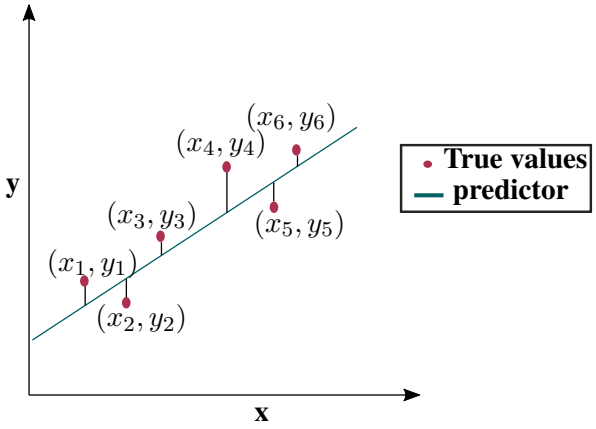

Figure 2.5: Mapping from x to y. The predictor is shown as linear line. The distance between the true values and predictor gives the loss. The sum of all the distances gives the loss function.

In MSLE, the mean of relative distances between predictions and true outputs are calculated.

$$MSLE = \frac{1}{n}\sum_{i=1}^{n}(log(y_i + 1) - log(\hat{y}_i + 1))^2 \tag{2.4}$$

Gradient descent

Gradient descent is another technique to minimise the cost function parameterised by a model parameter $\mathbf{w}$. The first derivative(or gradient) gives the slope of the cost function. Hence, to minimise it, direction opposite to the gradient is chosen.

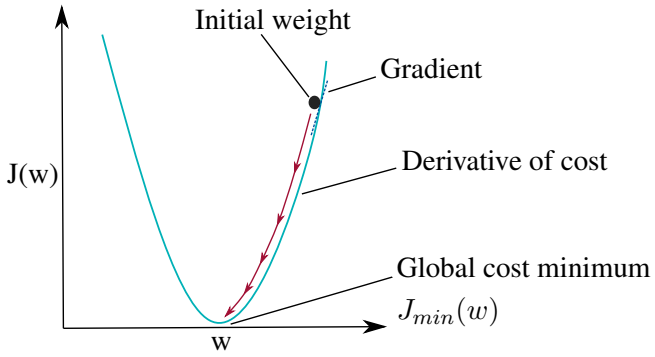

Figure 2.6: Finding the stochastic gradient descent

The rate at the which the gradient step reduces is given by the *learning rate*. If the learning rate is high, greater the step size of each gradient; possibly causing the step to miss the global minima. Lower the learning rate, more steps or training cycles needed to reach the global minima. So the rate must be carefully chosen for each model.

Backpropagation

Backpropagation are a class of algorithms which help in training feedforward neural networks for supervised learning. A model is said to fit when the gradient computation of the loss function is efficient w.r.t the weights of single input-output in the network. Backpropagation performs the effective gradient computation using the loss functions explained above.

Optimizer

The loss function was able to explain how far the predictions were compared to the true outputs in a mathematical way. During training process, certain parameters can be tweaked to help the loss function predict correct and optimised results. However, there are question such as how to change them, by how much and when?

This is exactly optmizer's function. As explained in **??**, gradient descent and learning rate form the core of optimizer's functionality. *Stocastic gradient descent*(SGD) is one of the oldest techniques in which gradients for all of your training examples on every pass of gradient descent are calculated. However, they are slow and require much computation power. Some of the other popular optimizers are Adam, Adaguard, RMSprop. In this work, Adam is used. Adam stands for adaptive moment estimation. It is a combination of all the advantages of two other extensions of SGD – Adaguard and RMSprop. Adam is computationally efficient, straight forward to implement, invariant to diagonal rescale of the gradients, less effort need to hyperparameters tuning.

Challenges in Machine learning algorithms

1. insufficient labelled data
2. poor quality data and irrelevant features
3. overfitting/underfitting the model

The first two issues can be solved if the user is careful during data collection and does preprocessing before feeding the data to the training model. However, if the training or the test data is too small, the model is subject to underfitting or overfitting. Though our aim is to reduce the error in the training set, we also need to reduce the error in the test set. The gap between training and testing error is also important parameter. Underfitting occurs when the model is not able to obtain sufficiently low error value for the training set. And if the gap between training and testing error is too large, overfitting happens. The sweet spot is to stop training the model when the gap between the two sets is at a minimum value. Left of the optimal point, the model underfits. Right of it, the model overfits. The below figure 2.7 shows it very well. Validation error is the error calculated for the test set.

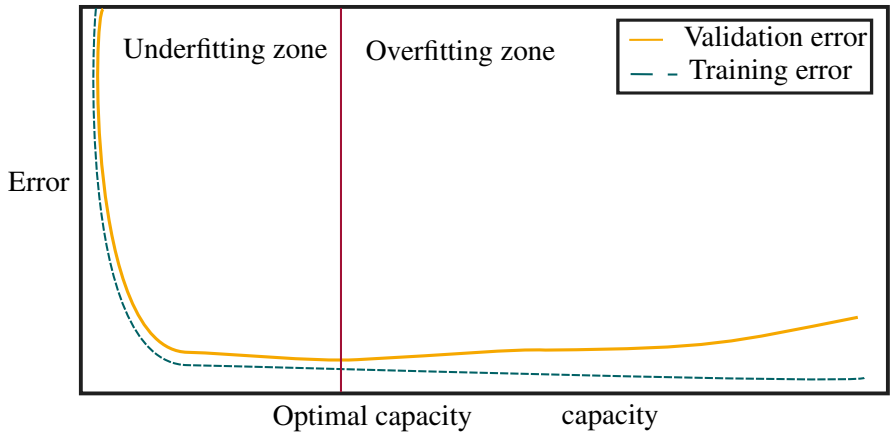

Figure 2.7: Relationship between capacity and error. Inspired from [28]

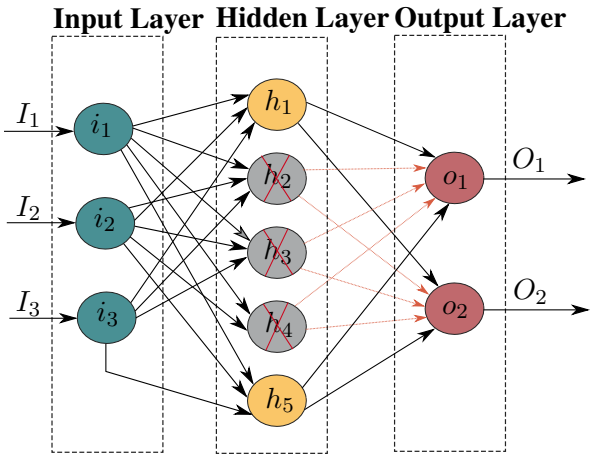

Figure 2.8: Illustrating dropout functionality

Dropout

DNNs contain multiple non-linear hidden layers and which makes them easily learn complex relationships between their inputs and outputs. With a small training set, this relationship adds sampling noise that won't exist in the real-world data even if drawn from the same distribution. This leads to overfitting and several methods have been developed to reduce its effect.

1. early stopping as soon as the validation error gets worse than the training error.

2. L1 and L2 regularisation which penalises the weights [33].

3. Randomly drop units(along with their connection) from the neutral network during training [34]. Figure 2.8 illustrates how to do the random dropping of units.

2.3 Deep deep learning

2.3.1 CNN

convolution - kernels, strides, padding

max pooling

batch normalisation

flatten

fully connected

2.3.2 RNN

2.3.2.1 LSTM

2.4 Sensors

2.4.1 visual sensors

RGB

depth

segmentation

2.4.2 measurement sensors

radar

control

2.5 data fusion

2.5.1 types of data fusion

2.6 keras

2.6.1 functional api

different layers

2.6.2 callbacks

model checkpoint

early stopping

tensorboard

2.7 ROS

2.7.1 ROS2

Sub,pub,msg filter, cache, slop, callbacks, spin, topic, bridge, message types

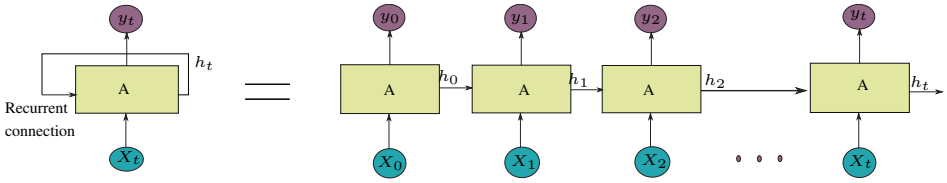

Figure 2.9: simple rnn

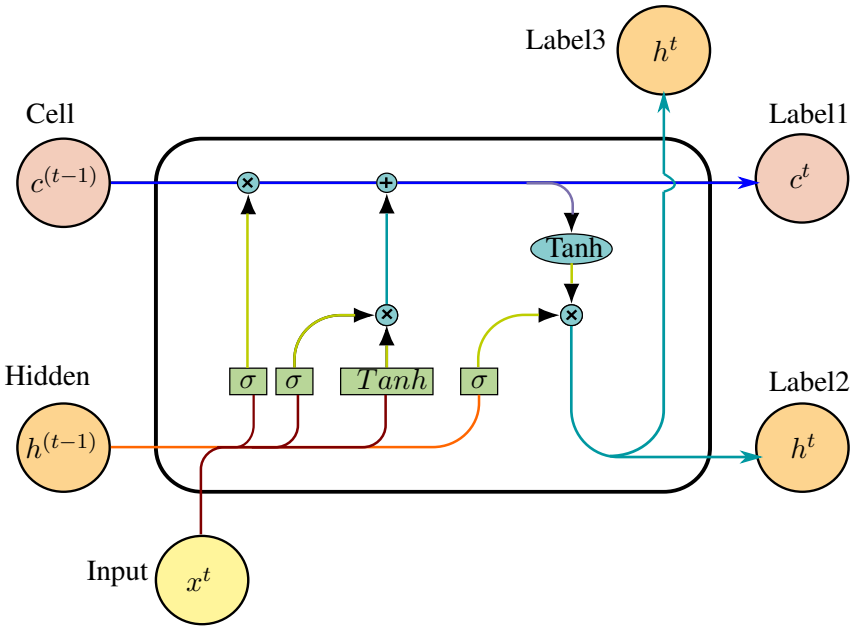

Figure 2.10: lstm

References

[1] "Waymo. [online]." https://waymo.com/

[2] "Berkeley-deepdrive. [online]." https://bdd-data.berkeley.edu/

[3] "Apollo. [online]." http://apolloscape.auto/

[4] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2012, pp. 1097–1105. http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

[5] "Radar introduction." https://www.explainthatstuff.com/radar.html

[6] "Lidar introduction," 2016. https://www.geospatialworld.net/blogs/what-is-lidar-technology-and-how-does-it-work/

[7] "Ultrasonic - what is it?. [online]." https://www.microcontrollertips.com/principle-applications-limitations-ultrasonic-sensors-faq/

[8] G. Rong, B. H. Shin, H. Tabatabaee, Q. Lu, S. Lemke, M. Možeiko, E. Boise, G. Uhm, M. Gerow, S. Mehta *et al.*, "Lgsvl simulator: A high fidelity simulator for autonomous driving," *arXiv preprint arXiv:2005.03778*, 2020. https://www.lgsvlsimulator.com/

[9] "Nvidia simulator. [online]." https://www.nvidia.com/en-us/self-driving-cars/drive-constellation/

[10] "Carla simulator. [online]." https://carla.org/

[11] "Carmaker simulator. [online]." https://ipg-automotive.com/products-services/simulation-software/carmaker/

[12] "Lgsvl simulator. [online]."

[13] "Robotic operating system. [online]." https://index.ros.org/doc/ros2/

[14] "Verge report on lidar." https://www.theverge.com/2020/1/7/21055011/lidar-sensor-self-driving-mainstream-mass-market-velodyne-ces-2020

[15] Z. Chen and X. Huang, "End-to-end learning for lane keeping of self-driving cars," pp. 1856–1860, 06 2017.

[16] M. Bojarski, D. D. Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, X. Zhang, J. Zhao, and K. Zieba, "End to end learning for self-driving cars," 2016.

[17] S. Levine, P. Pastor, A. Krizhevsky, and D. Quillen, "Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection," *The International Journal of Robotics Research*, 03 2016.

[18] H. Xu, Y. Gao, F. Yu, and T. Darrell, "End-to-end learning of driving models from large-scale video datasets," *CoRR*, vol. abs/1612.01079, 2016. http://arxiv.org/abs/1612.01079

[19] J. Kim and J. F. Canny, "Interpretable learning for self-driving cars by visualizing causal attention," *CoRR*, vol. abs/1703.10631, 2017. http://arxiv.org/abs/1703.10631

[20] M. Bojarski, P. Yeres, A. Choromanska, K. Choromanski, B. Firner, L. D. Jackel, and U. Muller, "Explaining how a deep neural network trained with end-to-end learning steers a car," *CoRR*, vol. abs/1704.07911, 2017. http://arxiv.org/abs/1704.07911

[21] Z. Yang, Y. Zhang, J. Yu, J. Cai, and J. Luo, "End-to-end multi-modal multi-task vehicle control for self-driving cars with visual perception," 01 2018.

[22] K. Liu, Y. Li, N. Xu, and P. Natarajan, "Learn to combine modalities in multimodal deep learning," 2018.

[23] E. Park, X. Han, T. L. Berg, and A. C. Berg, "Combining multiple sources of knowledge in deep cnns for action recognition." in *WACV*. IEEE Computer Society, 2016, pp. 1–8. http://dblp.uni-trier.de/db/conf/wacv/wacv2016.html#ParkHBB16

[24] Y. Zhou and K. Hauser, "Incorporating side-channel information into convolutional neural networks for robotic tasks," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, 2017, pp. 2177–2183.

[25] W. Wang, D. Tran, and M. Feiszli, "What makes training multi-modal classification networks hard?" in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 12 695–12 705.

[26] Y. Xiao, F. Codevilla, A. Gurram, O. Urfalioglu, and A. M. López, "Multimodal end-to-end autonomous driving," *CoRR*, vol. abs/1906.03199, 2019. http://arxiv.org/abs/1906.03199

[27] F. Codevilla, M. Mller, A. Lopez, V. Koltun, and A. Dosovitskiy, "End-to-end driving via conditional imitation learning," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, 2018, pp. 4693–4700.

[28] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[29] T. Mitchell, "M a c h ine l e a rnin g," *McCraw Hill*, 1996.

[30] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[31] K. P. Murphy, *Machine learning : a probabilistic perspective*. Cambridge, Mass. [u.a.]: MIT Press, 2013. https://www.amazon.com/Machine-Learning-Probabilistic-Perspective-Computation/dp/0262018020/ref=sr_1_2?ie=UTF8&qid=1336857747&sr=8-2

[32] "Cifar 10 image dataset." https://www.cs.toronto.edu/~kriz/cifar.html

[33] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural Networks*, vol. 61, p. 85117, Jan 2015. http://dx.doi.org/10.1016/j.neunet.2014.09.003

[34] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, 06 2014.