# Augmenting DAgger with Recurrence and Attention

**Ekeledirichukwu C. Nnorom**

**University of Toronto, St. George**

**MEng. Project Electrical and Computer Engineering**

**Supervised by Professor Florian Shkurti and Professor Enright Jerger**

**January 2020.**

# Abstract

Imitation learning techniques, where expert demonstrations of good behavior are used to learn a controller, have proven very useful in practice and have led to state-of-the art performance in a variety of applications. One such is to train a classifier or regressor to predict an expert's behavior given training data of the encountered observations (input) and actions (output) performed by the expert. However, since the learner's prediction affects future input observations/states during execution of the learned policy, this violates the crucial i.i.d. assumption made by most statistical learning approaches. This leads to poor performance in theory and often in practice. DAgger an iterative online no regret algorithm, which trains a stationary deterministic policy was used in this project. The supervised learning subroutines are a Feed Forward Convolutional, Neural Network, Recurrent CNN, CNN with Attention Gates and an End to End Recurrent Convolutional Network with Attention Gates. The system automatically learns internal representations of the necessary processing steps such as detecting useful road features with only the human steering angle as the training signal. A prominent difference is between the approaches here is most CNN are typically a feed-forward architecture – an incomplete representation of the human visual system which has ubiquitous recurrent connections. Despite static input, the activities of an RCNN unit evolve over time so that the activity of each unit is modulated by the activities of its neighboring units. This enhances the model's ability to integrate the context information, which is important for object recognition. The models trained with Attention Gates (AG) implicitly learn to suppress irrelevant regions in an input image while highlighting useful features. The result of these approaches is a more robust policy for Imitation Learning.

*Keywords*: Recurrence, Attention Gates, Dagger

# Acknowledgement

I would specifically like to thank Prof. Enright Jerger and Prof. Florian Shkurti for supervising my research on this project and providing direction for the experiments.
Lastly, I thank my family and friends for their treasured love and support.

# Contents

# Augmenting DAgger with Recurrence and Attention

# Introduction

Imitation learning [1, 2, 3, 4, 5, 6, 7] is a problem in machine learning that autonomous agents face when attempting to learn tasks from another, more-expert agent. The expert provides demonstrations of task execution, from which the imitator attempts to mimic the expert's behavior. In complex robotic systems where standard control methods fail, a controller is often learnt to make predictions. An approach is to train a classifier or regressor to predict an expert's behavior given training data of the encountered observations (input) and actions (output) performed by the expert. However, since the learner's prediction affects future input observations/states during execution of the learned policy, this violates the crucial i.i.d. assumption made by most statistical learning approaches. Ignoring this issue leads to poor performance both in theory and practice [7].An approach to addressing this [7]propose a non-stationary policy by training a different policy for each time step in sequence, starting from the first step. Unfortunately, this is impractical when T is large or ill-defined. In SMILE (Stochastic Mixing Iterative Learning) a stationary stochastic policy is trained by adding a new policy at each iteration of training creating a finite mixture of policies. This also yielded unsatisfying performance for practical applications as some policies in the mixtures were worse than others and the learned controller may be unstable.

As a result, Ragnell and Ross proposed a meta-algorithm for imitation learning which learns a stationary deterministic policy guaranteed to perform well under its induced distribution of states (number of mistakes/costs that grows linearly in T and classification cost E). A reduction-based approach [8] that enables reusing existing supervised learning algorithms was adopted. The approach was simple to implement, with no free parameters except the supervised learning algorithm sub-routine, and requires a number of iterations that scaled nearly linearly with the effective horizon of the problem. It naturally handled continuous as well as discrete predictions. Ross et al. showed that any no-regret learner can be used in a particular fashion to learn a policy that achieved similar guarantees. Recent approaches to this algorithm have involved supervised learning algorithm sub-routine including Convolutional Neural Networks.

Convolutional Neural Networks have been the de facto standard for object recognition due to the high representation power, fast inference and filter sharing properties. Partially inspired by neuroscience, Convolutional Neural Networks share many properties of the visual system of the brain. However, CNN are usually feed-forward architectures unlike the visual system which supports abundant recurrent connections. As such common CNNs are incomplete approximations of the biological neural network. Evidences have shown that the neocortex contains recurrent synapses which outnumber feed-forward and top-down (feedback) synapses [9]. The presence of recurrent synapses makes object recognition a dynamic process though the input is static. Though exact functions of these synapses remain unclear, but it is generally believed that recurrent synapses play an important role in context modulation. The processing of visual signals is strongly modulated by their context [10]. Normally we do not perceive this effect without attention, but the effect gets prominent in perceptual illusions. A feed-forward model only captures the context in higher layers where the RF is high, but context modulation at the lower layers responsible for recognizing smaller objects is not possible. Convolutional Deep Belief Networks (CDBN) use a top-down (or feedback) connection. In this project recurrent connections are used within the same layer of deep learning models with the expectation that the lateral connections may boost the model performance. Furthermore, attention gates are utilized in this project to suppress irrelevant regions of an input image

while emphasizing salient features useful for a specific task. Prior CNNs would require cascaded frameworks to extract region of interest. This leads to excessive and redundant use of computational resources and model parameters. CNN models with AGs can be trained from scratch in the standard way similar to the training of a FCN model [11]. It is expected that by combining these approaches, a more robust and sensitive no regret algorithm could be obtained for the task of Imitation learning for autonomous vehicles.

# Chapter 1

## Convolutional Neural Networks

# 1.1. Fully Connected Layers (FCL)

An FCL is a mathematical function that applies a linear transformation on an input vector of dimension *I* and outputs a vector of dimension ***O***. A bias parameter is usually added to reduce overfitting.

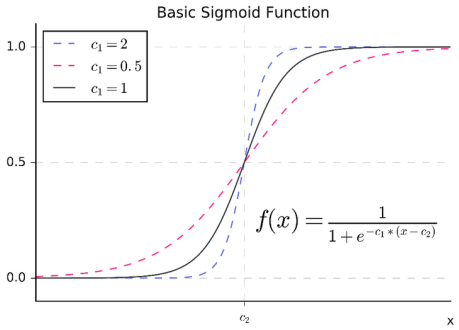
$$y = w.x + b$$

$$y_i = \sum_{j=1}^{I} (w_{i,j} x_j) + b_i$$

# 1.2. Non Linearity and Activation Functions

Non-linear activation functions enable the neural networks to approximate functions especially non-convex functions. An activation functions takes a vector and performs a fixed point-wise mathematical operation on it. Popular activation functions include:

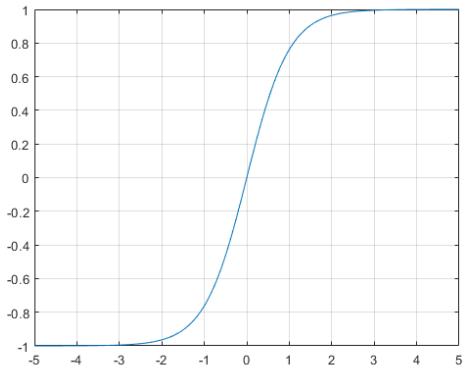
## 1.2.1. Sigmoid




$$y = \sigma(x) = {}^{1}/_{(1 + e^{-x})}$$

A sigmoid function is a monotonic, differentiable, bounded and often real function defined for real input values with a non-negative derivative at each point. A sigmoid function takes a real value and squashes it between 0 and 1. It has a characteristic 'S' shape. The logistic function is often used as a standard choice for the sigmoid function. The sigmoid function is convex for values less than 0 and concave for values greater than 0. Should the neuron's activation saturate at either tail of 0 or 1, the gradient at these regions is almost zero which could typically result in the Vanishing Gradient Problem.

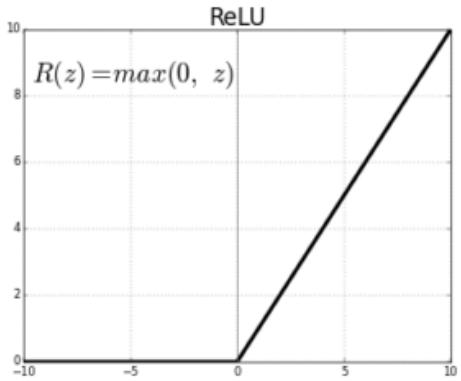
## 1.2.2. Hyperbolic Tangent




$$y = 2\sigma(2x) - 1$$

The hyperbolic tangent squashes the real value between -1 and 1 but suffers the same drawback as the Sigmoid.

### 1.2.3. Rectified Linear Unit




It is also known as a ramp function. It is an activation function defined on the positive part of its input. It's widely used for neural networks especially convolutional layers and linear layers. It was found to greatly accelerate the convergence of stochastic gradient descent compared to the hyperbolic tangent and the sigmoid. Fewer vanishing gradients problems occur and it also helps sparse activations. More so its computation is cheap compared to expensive exponentials. However it is non-differentiable at zero, unbounded, not zero-centered. Also, the ReLU removes all the negative information.

# 1.3. Spatial Convolution

Neural networks comprising solely of linear and activation layers do not scale well to images. For example, an image of size 3 x 128 x 128 (3 color channels - RGB, 128 pixels wide, 128 pixels high) would require the first linear layer having 3 * 128 * 128 = 49,152 parameters for a single neuron. Images of height and width of 224 and above are also not uncommon. As such, spatial convolution layers are adopted because images and feature maps exhibit many spatial relationships. Thus, a convolutional layer learns a set of $N_k$ filters with $F = f_1, \ldots, f_{Nk}$, which are convolved spatially with input image x, to produce a set of $N_k$ 2D feature maps z:

$$z_k = f_k * x$$

Where * represents the convolution operator. A good correlation of the filter and a region of the input image or feature map, results in a strong response in the corresponding feature map location. More so, in the convolutional layer, weights are shared over the entire image reducing the number of parameters per response and also learns equivariance (i.e. shifting an object in the input image would simply shift the corresponding responses in a similar way).

# 1.4. Spatial Pooling

Pooling is typically present in Convolutional Neural Networks to provide invariance to slightly different input images and to reduce the dimension of the feature maps by typically by the height and width. Output feature maps are sensitive to the location of the features of the input, pooling is used to address this sensitivity by down sampling the feature maps. This makes the resulting feature maps more robust to changes in position of the feature in the image - local translation invariance.

$$p_R = P_{i \in R}(z_i)$$

Where P is a pooling function over the region of pixels R. Max pooling is preferred as it avoids cancelling negative elements and prevents blurring of the activations and gradients throughout the network as the gradient is placed in a single location during backpropagation. The spatial pooling layer is defined by its aggregation function(max pooling, average pooling and so on), the height and width dimensions of the area where it is applied to as well as the properties of the convolution (e.g. padding, stride).

# 1.5. Batch Normalization

This helps faster convergence by adding a normalization step - shifting inputs to zero-mean and unit variance to make inputs of trainable layers comparable across features. This promotes a high learning rate while keeping the network learning. It also allows activation functions escape being stuck at the saturation mode (e.g. gradient of 0).

# 1.6. Training

## 1.6.1 Initialization

The weights of a neural network must be initialized to small random numbers as the optimization algorithm used to train the model requires it. These optimization algorithms are nondeterministic algorithms that use elements of randomness to make decisions during the execution of the algorithm. This means different orders of steps will be followed when the same algorithm is rerun on the same data. These optimization

algorithms are incremental in nature of the search involves an optimization from an initial state or position to a final state or position. The weights of the network could be initialized as follows:

**Zero and Random initialization** biases are typically initialized to zero while weights are initialized to random numbers.

Other forms of initialization include:

**He Initialization**

In He Initialization we simply multiply the random initialization with:

$$\sqrt{\frac{2}{size^{[l-1]}}}$$

**Xavier Initialization**

This is slightly different from the He Initialization:

$$\sqrt{\frac{1}{size^{[l-1]}}}$$

## 1.6.2. Loss Functions

Loss functions are used to quantify the network's capacity to approximate the ground truth labels for all training inputs.

## Mean Squared Error (MSE)

It is a multi class loss formerly used to train neural networks. It is typically used for regression problems where the outputs are neither binary nor categorical.

$$Loss(x, y) = \frac{1}{n}\sum_{i}|x_i - y_i|^2$$

## Cross Entropy

$$Loss(x, y) = -\sum_{i} y_i * log\left(\frac{e^{x_i}}{\sum_j e^{x_j}}\right)$$

It is typically used for categorical and classification problems. Unlike the MSE which could slow down learning, the cross entropy doesn't.

## Loss Multi Label

This is an adaptation of the cross entropy loss for multi-label classification. It is a multi-label one-versus-all loss based on max-entropy.

$$Loss(x, y) = -\sum_i \left( y_i * log\left(\frac{e^{x_i}}{1 + e^{x_i}}\right) + (1 - y_j) * log\left(\frac{1}{1 + e^{x_i}}\right)\right)$$

## 1.6.3. Optimization Algorithms

Typically, the loss function of a CNN is highly non-convex. However, gradient based optimization algorithms can be applied as these loss functions are also differentiable. Still, because of the size of the parameter space, the first derivatives are used in practice. Second order derivatives are costly to compute. Optimization Algorithms include:

**Stochastic Gradient Descent (SGD):** It is one of the main optimization algorithms used. It consists in using a few examples to compute the gradient of the parameters with respect to the loss function. SGD reduces computational cost at each iteration. At each iteration of SGD, we uniformly sample an index $i \in \{1, \ldots, n\}$ for data instances at random, and compute the gradient $\nabla f_i(x)$ to update **x**:

$$x \leftarrow x - \gamma \nabla f_i(x)$$

Where n is the learning rate. This reduces the computing cost for each iteration from $O(n)$ to $O(1)$. SGD is also the unbiased estimate of the gradient$\nabla f_i(x)$. This algorithm reaches a good local minima in practice, even when the parameters are randomly initialized.

Batch Gradient Descent is a variation of SGD that calculates the error for each example in the training dataset, but updates the model only after all training examples have been evaluated. Variants of SGD include Mini-batch Stochastic Gradient Descent which splits the training dataset into small batches that are used to calculate model error and update model coefficients. It seeks to find the balance between the robustness of Stochastic Gradient Descent and the efficiency of Batch Gradient Descent.

## 1.6.4. Regularization

Regularization can be defined as any modification we make to a learning algorithm that is intended to reduce its generalization error but not its training error. This regularization is often done by putting some extra constraints on a machine learning model, such as adding restrictions on the parameter values or by adding extra terms in the objective function that can be thought of as corresponding to a soft constraint on the parameter values. If chosen correctly these can lead to a reduced testing error. An effective regularizer is

said to be the one that makes a profitable trade by reducing variance significantly while not overly increasing the bias.

**L2 Regularization and Weight Decay**

The first main approach to overcome overfitting is the classical weight decay, which adds a term to the cost function to penalize the parameters in each dimension, preventing the network from exactly modeling the training data and therefore help generalize to new examples:

$$Err(x, y) = Loss(x, y) + \sum_i {\theta_i}^2$$

**Dropout**

Dropout the idea is to randomly set a certain percentage of the activations in each layers to 0. During the training, neurons must learn better representations without co-adapting to each other being active. During the testing, all the neurons are used to compute the prediction and Dropout acts like a form of model averaging over all possible instantiations of the model.

**Data augmentation**

Data augmentation is a method of boosting the size of the training set so that the model cannot memorize all of it. This can take several forms depending of the dataset. For instance, if the objects are supposed to be invariant to rotation such as galaxies or planktons, it is well suited to apply different kind of rotations to the original images.

# Chapter 2

# CNNs for Self Driving Cars

Convolutional Neural Networks have been extensively used in pattern recognition [12]. Prior to their widespread adoption, most pattern recognition tasks required hand-crafted feature extractors followed by a classifier. The breakthrough of CNNs is that features are learned automatically from training examples. The CNN approach is especially powerful in image recognition tasks because the convolution operation captures the 2D nature of images. In [13] the researchers used a convolutional neural network to predict steering commands for an autonomous vehicle. The CNN Architecture learnt the entire processing pipeline needed to steer an automobile. The groundwork for their project was done over 10 years before in a Defense Advanced Research Projects Agency (DARPA) seedling project known as DARPA Autonomous Vehicle (DAVE) [14] in which a sub-scale radio control (RC) car drove through a junk-filled alley way. DAVE's performance was not satisfactory enough to be an alternative to the existing modular approaches to off-road driving with its mean distance between crashes of about 20 meters in complex environments. Mariusz Bojarski, Davide D. Testa et al expressed that the motivation for the work was to avoid the need to recognize specific human-designated features, such as lane markings, guard rails, or other cars, and to avoid having to create a collection of "if, then, else" rules, based on observation of these features.

## 2.1. The DAVE-2 System

The collection system for the training data for DAVE-2 consisted of three cameras mounted behind the windshield of the data-acquisition car. Time-stamped video from the cameras was captured simultaneously with the steering angle applied by the human driver. The steering command was obtained by tapping into the vehicle's Controller Area Network (CAN) bus. More so, the steering command was set to 1/r where r is the turning radius in meters, so as to make the system independent of the car geometry.

The network needed to learn how to recover from mistakes. Otherwise the car would slowly veer off the road. To make the system more robust, the training data was augmented with additional images which showed the car in different shifts from the center of the lane and rotations from the direction of the road. A block diagram of the training system is shown in Figure 2. Images were fed into a CNN which then computes a proposed steering command. The proposed command is compared against the actual command for that image and the weights of the CNN are adjusted by backpropagation.

# 2.2. Network Architecture

In the original paper the weights of the network we trained to minimize the mean squared error between the steering command output by the network and the command of either the human driver, or an adjusted steering command. The network consisted of 9 layers, including a normalization layer, 5 convolutional layers and 3 fully connected layers. The input image is split into YUV planes and passed to the network. The convolutional layers were designed to perform feature extraction and were chosen empirically through a series of experiments that varied layer configurations. Strided convolutions were adopted in the first three convolutional layers with a 2×2 stride and a 5×5 kernel and a non-strided convolution with a 3×3 kernel size in the last two convolutional layers.

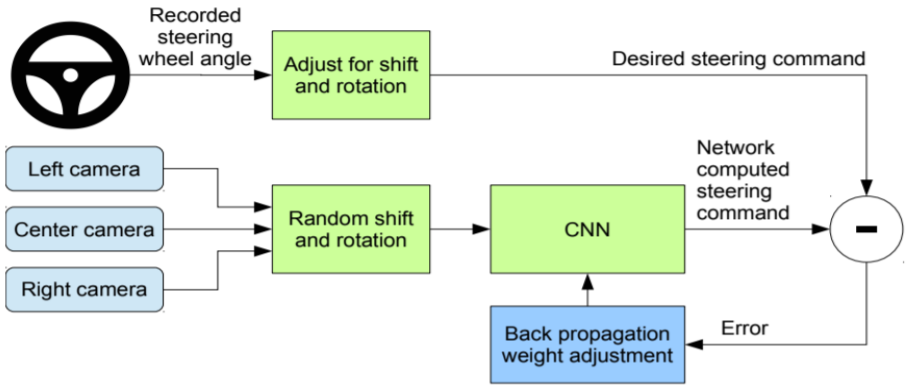



*Figure 1: The training of the network*

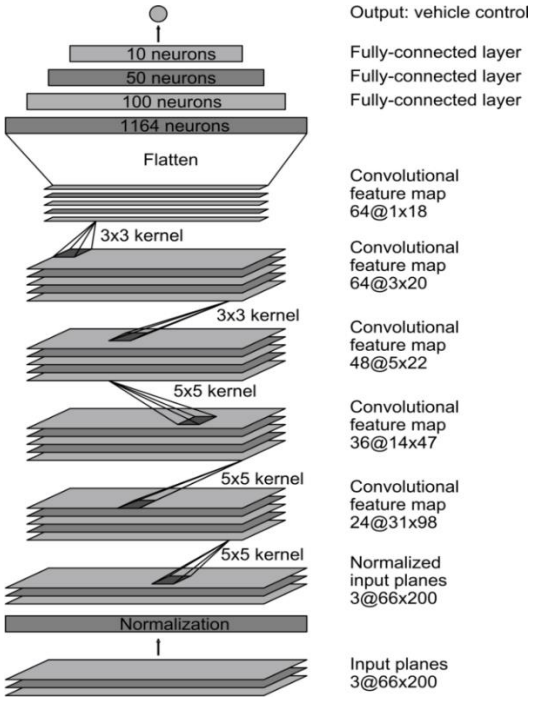



*Figure 2: CNN Architecture*

To remove a bias towards driving straight the training data included a higher proportion of frames that represented road curves. After selecting the final set of frames, the data was augmented by adding artificial shifts and rotations to teach the network recovery from a poor position or orientation.

# Chapter 3

# No-Regret Online Learning

According to Yisong Yue, Hoang M. Le, Imitation learning is a powerful and practical alternative to reinforcement learning for learning sequential decision-making policies. Also known as learning from demonstrations or apprenticeship learning, imitation learning has benefited from recent progress in core learning techniques, increased availability & fidelity of demonstration data, as well as the computational advancements brought on by deep learning.

Imitation learning techniques involve learning a controller from expert demonstrations of good behavior. These have been very useful in practice leading to state-of-the-art performance in many applications. A common approach is to train a classifier or regressor to predict an expert's behavior using training data with input corresponding to the encountered observations and the outputs being the action (e.g. steering command). This violates the i.i.d assumption of most statistical learning techniques because the learner's prediction affects future input observations/ states during learned policy execution.

This issue leads to poor performance both in theory and practice if ignored [7]. The researchers showed that if a classifier makes a mistake with probability $\in$ under the distribution of states/observations encountered by the expert can make as many as $T^2 \in$ mistakes in expectation over T-steps under the distribution of states the classifier itself induces [7].

Some approaches [7]could guarantee an expected number of mistakes linear (or nearly so) in the task horizon T and error by training over several iterations and allowing the learner to influence the input states where expert demonstration is provided (through execution of its own controls in the system). A non-stationary policy could be learned by training a different policy for each time step in sequence, starting from the first step. This approach would prove impractical for a large or ill-defined T.

Ross and Bagnell proposed an approach where a non-stationary policy was learned by training a different policy for each time step in sequence, starting from the first step. SMILe [7], similar to SEARN [48] and CPI [49], trained a stationary stochastic policy (a finite mixture of policies) by adding a new policy to the mixture at each iteration of training. This also had an unsatisfactory outcome for practical applications due to some policies performing worse than others within the mixture. The result would be an unstable learned controller.

DAgger proposes a meta-algorithm for imitation learning which learns a stationary deterministic policy guaranteed to perform well under its induced distribution of states (number of mistakes/costs that grows linearly in T and classification cost E). A reduction-based approach [8] that enables reusing existing supervised learning algorithms was adopted. The approach was simple to implement, with no free parameters except the supervised learning algorithm sub-routine, and required a number of iterations that scaled nearly linearly with the effective problem horizon. DAgger was able to handled continuous as well as discrete predictions. They showed that any no-regret learner can be used in a particular fashion to learn a policy that achieved similar guarantees.

**Definition of terms:**

$\Pi -$ class of policies

$\pi -$ a single policy

$d_\pi^t -$ Distribution of states at time t with learner executed policies from 1 to t-1

$d_\pi = \frac{1}{T}\sum_{t=1}^{T} d_\pi^t$ – Average distribution of states if we follow policy π for T steps

$C(s,a)$ – expected immediate cost of performing action a in state s

$C_\pi(s) = \mathbb{E}_{a\sim\pi(s)}[C(s,a)]$

$J(\pi) = \sum_{t=1}^{T} \mathbb{E}_{s\sim d_\pi^t}[C_\pi(s)] = T\mathbb{E}_{s\sim d_\pi}[C_\pi(s)]$ with C bounded in [0, 1]

Where $J(\pi)$ is the *cost-to-go*

Since the true costs C(s, a) are not directly measured, expert demonstrations are observed and J(π) is bounded for any cost function C based on how well π mimics the expert's policy $\pi *$.

$l$ – surrogate loss to be minimized instead of C

$$\hat{\pi} = arg\min_{\pi\in\Pi} \mathbb{E}_{s\sim d_\pi}[l(s,\pi)]$$

$\hat{\pi}$ - a policy which minimizes the observed surrogate loss under the induced distribution of states.

# 3.1. Supervised Approach to Imitation

$$\hat{\pi}_{sup} = arg\min_{\pi\in\Pi} \mathbb{E}_{s\sim d_{\pi^*}}[l(s,\pi)]$$

$$J(\hat{\pi}_{sup}) = (1 - \epsilon T)J(\pi^*) + T^2\epsilon$$

Traditional supervised learning approach ignores the change in distribution, training the policy that performs well under the distribution of states encountered by the expert. As a result traditional supervised learning has poor performance guarantee due to the quadratic growth in T.

## 3.1.1 Forward Training

The forward training [7] trained a non-stationary policy (one policy $\pi_t$ for each time step t) iteratively over T iterations. Based on the states induced by previously trained policies $\pi_1, \pi_2, \ldots, \pi_{t-1}$ models $\pi_t$ is trained to mimic the optimal policy $\pi^*$. Hence, $\pi_t$ is trained on the actual distribution of states it will encounter during execution of the learned policy. The expected loss under the distribution of states induced by the learned policy matches the average loss during training, and hence improves performance.

$$J(\pi) = J(\pi^*) + \sum_{t=0}^{T-1}[J(\pi_{1:T-t}) - J(\pi_{1:T-t-1})$$

$$= J(\pi^*) + \sum_{t=1}^{T} \mathbb{E}_{s\sim d_\pi^t}\left[Q_{T-t+1}^{\pi^*}(s,\pi) - Q_{T-t+1}^{\pi^*}(s,\pi^*)\right]$$

$$\leq J(\pi^*) + u\sum_{t=1}^{T} \mathbb{E}_{s\sim d_\pi^t}[l(s,\pi)]$$

$$= J(\pi^*) + uT\epsilon$$

In many cases u will be *O(1)* or sub-linear in T.

The forward algorithm becomes impractical when T is large (or undefined) as we must train T different policies sequentially and cannot stop the algorithm before we complete all T iterations. Hence it cannot be applied to most real-world applications.

## 3.1.2. Stochastic Mixing Iterative Learning (SMILe)

SMILe, alleviates this problem and can be applied in practice when T is large or undefined by adopting an approach where a stochastic stationary policy is trained over several iterations. Initially SMILe starts with a policy $\pi_0$ which always queries and executes the expert's action choice. At iteration n, a policy $\hat{\pi}_n$is trained to mimic the expert under the distribution of trajectories $\pi_{n-1}$ induces and then updates $\pi_n$.

$$\pi_n = \pi_{n-1} + \alpha(1-\alpha)^{n-1}(\hat{\pi}_n - \pi_0)$$

Ross and Bagnell (2010) showed that choosing α in $O\left(\frac{1}{T^2}\right)$ and N in $O(T^2 logT)$ guarantees near-linear regret in T and for some class of problems.

# 3.2. DATASET AGGREGATION – DAgger

The no-regret algorithm DAGGER (Dataset Aggregation) is an iterative algorithm that trains a deterministic policy that achieves good performance guarantees under its induced distribution of states.

Initialize $\mathcal{D} \leftarrow \emptyset$.
Initialize $\hat{\pi}_1$ to any policy in $\Pi$.
for $i = 1$ to $N$ do
Let $\pi_i = \beta_i \pi^* + (1 - \beta_i)\hat{\pi}_i$
Sample T-step trajectories using $\pi_i$.
Get dataset $\mathcal{D}_i = \{(s, \pi^*(s))\}$ of visited states by $\pi_i$ and actions given by expert.
Aggregate datasets: $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{D}_i$
Train classifier $\hat{\pi}_{i+1}$ on $\mathcal{D}$.
end for
Return best $\hat{\pi}_i$ on validation

Algorithm

In other words, DAGGER proceeds by collecting a dataset at each iteration under the current policy and trains the next policy under the aggregate of all collected datasets.

**Theorem 3.1.** For DAGGER, if N is $\tilde{O}(T)$ *there exists a policy* $\hat{\pi} \epsilon \hat{\pi}_{1:N}\ s.t.\ \mathbb{E}_{s\sim d_{\hat{\pi}}}[l(s,\hat{\pi})] \leq \epsilon_N + O(\frac{1}{T})$

**Theorem 3.2.** For DAGGER, if N is $\tilde{O}(uT)$ *there exists a policy* $\hat{\pi} \epsilon \hat{\pi}_{1:N}\ s.t. J(\hat{\pi}) \leq J(\pi^*) + uT\epsilon_N + O(1)$.

**Finite Sample Results**

**Theorem 3.3**
For DAGGER if N is $O(T^2 \log(1/\delta))$ *and m is* $O(1)$ *then with probablity at least* $1 - \delta$ *there exists a policy* $\hat{\pi} \epsilon \hat{\pi}_{1:N}\ s.t.\ \mathbb{E}_{s\sim d_{\hat{\pi}}}[l(s,\hat{\pi})] \leq \epsilon_N + O(\frac{1}{T})$

**Theorem 3.4.**For DAGGER, if N is $O(u^2T^2 \log\left(\frac{1}{\delta}\right))$ and m is $O(1)$ *then with probability at least* $1 - \delta$ there exists a policy $\hat{\pi} \epsilon \hat{\pi}_{1:N}\ s.t. J(\hat{\pi}) \leq J(\pi^*) + uT_{\hat{\epsilon}_N} + O(1)$.

# Chapter 4 – DAGGER VARIANTS

## 4.1. Recurrent Neural Network

R**ecurrent neural network** (**RNN**) are artificial neural networks where connections between nodes form a directed graph along a temporal sequence. RNNs temporal dynamic behavior. Unlike feedforward neural networks, RNNs can use their internal state (memory) to process sequences of inputs. This makes them applicable to tasks such as unsegmented, connected handwriting recognition or speech recognition.

The term "recurrent neural network" is used indiscriminately to refer to two broad classes of networks with a similar general structure, where one is finite impulse and the other is infinite impulse. Both classes of networks exhibit temporal dynamic behavior. A finite impulse recurrent network is a directed acyclic graph that can be unrolled and replaced with a strictly feedforward neural network, while an infinite impulse recurrent network is a directed cyclic graph that cannot be unrolled.

Due to their effectiveness in broad practical applications, LSTM networks have received a wealth of coverage in scientific journals, technical blogs, and implementation guides. However, in most articles, the inference formulas for the LSTM network and its parent, RNN, are stated axiomatically, while the training formulas are omitted altogether. In addition, the technique of “unrolling” an RNN is routinely presented without justification throughout the literature [31].

However, the model is also related to the recursive neural network [32], in which a recursive layer is unfolded to a stack of layers with tied weights. Socher et al. [33] used a recursive neural network to perform scene parsing. Also, RNN for static visual signal processing have also been applied. Including multi-dimensional RNN (MDRNN) which was proposed for off-line handwriting recognition. This however, had a directed structure as it treated the image as 2D sequential data. More so, MDRNN had a single hidden layer which could not produce the feature hierarchy as CNN. Neural Abstraction Pyramid (NAP) was also proposed for image processing. NAP had both vertical and lateral recurrent connectivity, with which refining the image interpretation is done to resolve the visual ambiguities. However, though general framework of NAP has recurrent and feedback connections, for object recognition only the feed-forward version was tested. CDBN uses top-down connections for unsupervised feature learning with inference from the top layer propagating to the bottom layer through intermediate layers between them. However, unlike these architectures, RCNN adopts recurrent connections within the same layer, not between layers.

# 4.2. Recurrent Convolutional Neural Networks

With the adoption of CNNs and increasing deeper architectures Ming Liang and Xiaolin Hu explored the effects of adding recurrence to the convolutional layers of the neural network. They expressed CNN, as well as other hierarchical models including Neocognitron [15] and HMAX [16], is closely related to Hubel and Wiesel's findings about simple cells and complex cells in the primary visual cortex (V1) [17] [18].

CNN have been characterized by local connections, weight sharing and local and global pooling. The first two properties enable the model to discover local informative visual patterns with fewer adjustable parameters than MLP. The third property equips the network with some translation invariance. Over the past years, many techniques have been developed for improving the performance of CNN. The rectified linear function [19] is the most commonly used activation function due to its resistance to the gradient vanishing problem during Back Propagation. Dropout [20] is an effective technique to prevent neural networks from overfitting in training. Simonyan and Zisserman [21] used 3 × 3 convolutions to build very deep networks, considering that a stack of small filters has stronger nonlinearity than a large filter with the same amount of parameters. Szegedy et al. [22] proposed the multi-scale inception modules and built the GoogLeNet based on them. Small filters were also favored in this model.

However, these models were designed to have purely feed-forward architectures, which are incomplete approximations of the biological neural network in the brain. Anatomical evidences show that recurrent connections are prevalent in the neocortex, more so, recurrent synapses typically outnumber feed-forward and top-down (or feedback) synapses [23]. The presence of recurrent and top-down synapses make object recognition a dynamic process though the input is static. Recurrent synapses are believed to play an important role in context modulation. The processing of visual signals is strongly modulated by their context [24]. Context modulation become more apparent in perceptual illusions and can be observed in the responses of individual neurons in the visual system [25].

The context is important for object recognition (Figure 1). Feed-forward models are able to capture the context in higher layers where units have larger RFs, but this information cannot modulate the activities of units in lower layers responsible for recognizing smaller objects (e.g., the nose in Figure 1).

*Figure 3 without the context "face" it is hard to recognize the black curve in the middle area as nose*

To enable context modulation, one strategy is to use top-down (or feedback) connections to propagate it downwards [26], which is adopted in the convolutional deep belief networks (CDBN) [27]. Ming Liang et al. however adopted a different strategy; using recurrent connections within the same layer of deep learning models. Equipped with context modulation ability, these lateral connections are able to improve the performance of deep learning models.

The architecture is illustrated in Figure 2, where both feed-forward and recurrent connections have local connectivity and shared weights among different locations. This architecture is very similar to the recurrent multilayer perceptron (RMLP) which is often used for dynamic control [28] [29] (Figure 2, middle) except that that the full connections in RMLP are replaced by shared local connections, just as the difference between MLP [30] and CNN.

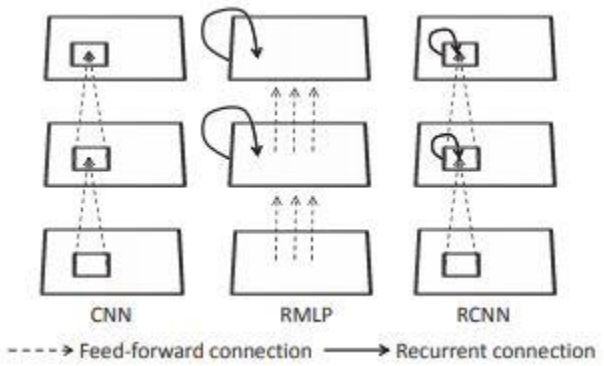



*Figure 4. Illustration of the architectures of CNN, RMLP and RCNN.*

## 4.2.1. Recurrent Convolutional Layer (RCL)

The key module of RCNN is the recurrent convolutional layer (RCL). The states of RCL units evolve over discrete time steps.

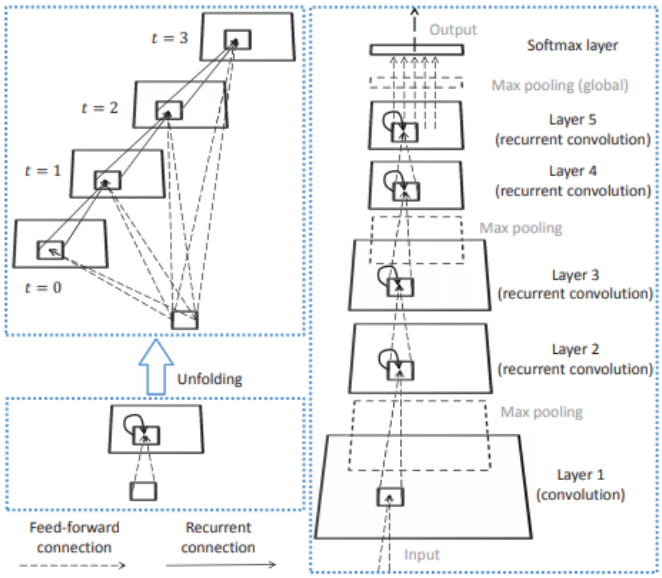
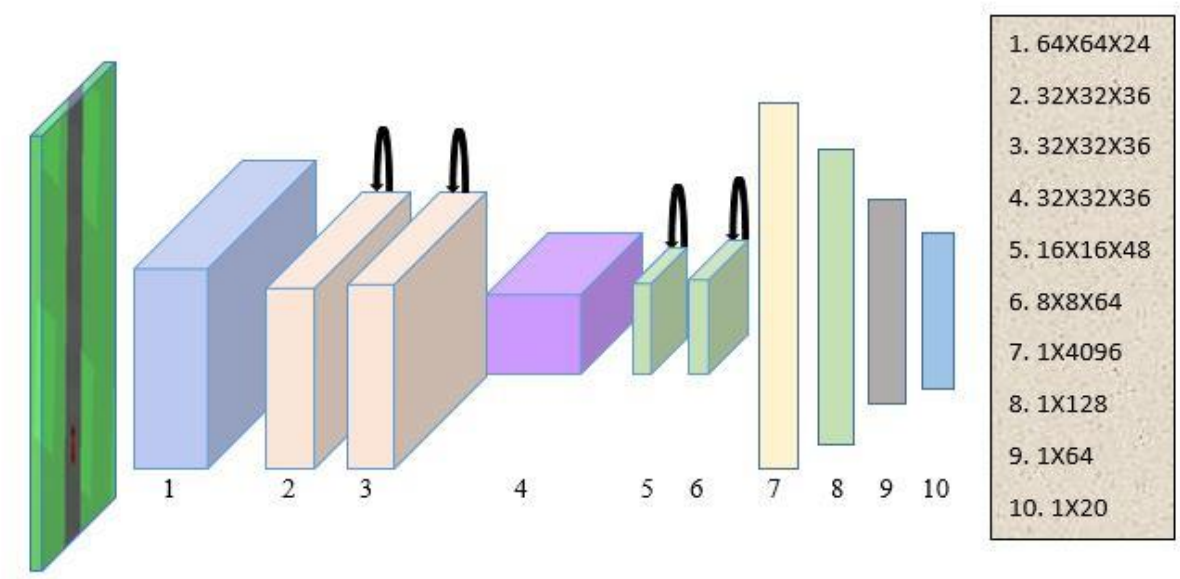



*Figure 5 overall architecture of RCNN. Left: An RCL is unfolded for T = 3 time steps, leading to a feed-forward subnetwork whose largest depth is and smallest depth is 1. Right: RCNN used by the authors.*

# 4.3 RecurrentDagger

## 4.3.1. Recurrent Subroutine




In this section, I introduce Recurrent DAgger, a variant of the iterative no-regret algorithm. Having seen that the DAgger approach would work with any supervised learning algorithm, I provide a method of improving the robustness and accuracy DAgger by adding recurrent blocks. With this approach, every unit of the subroutine model is able to incorporate contextual information in an arbitrarily large region of its current layer. As the time increases the state of every unit in the Recurrent Block is affected by other units in a larger and larger neighborhood in the current layer. Also, this deeper network does not drastically increase the number of parameters.

Each recurrent block increases the depth of the network but keeps the number of adjustable parameters constant by weight sharing. Hence Recurrent DAgger is consistent with the trend of modern architectures – going deeper with relatively small number of parameters.

More so, the time-unfolded version of each block is actually a CNN with multiple paths between the input layer and the output layer – facilitating learning. The existence of longer paths make it possible for the network to learn highly complex features; while the shorter paths may help gradient backpropagation during training. In this project the results of the Recurrent Dagger surpass the Vanilla version in training and accuracies as the advantages of recurrence are incorporated into this version of DAgger.

Recurrent blocks and convolutional blocks work seamlessly with no block interfering with the operations of the other resulting in a more robust DAgger subroutine. This approach combines the benefits of DAgger and the benefits of recurrence. This approach worked well for 10 Dagger iterations with consistently superior performances to Vanilla DAgger. The architecture is shown above.

# 4.4. AttentionDAgger

## 4.4.1 Attention Gates

The human perception process [34] provides evidence of the importance of attention mechanisms, which uses top information to guide bottom-up feedforward process. Deep Boltzmann Machine (DBM) [35] contains top-down attention by its reconstruction process in the training stage. Attention mechanisms have been applied to recurrent neural networks (RNN) and long short term memory (LSTM) as well for image captioning and other sequential decision tasks [36] [37] [38] [39]. In these approaches top information is gathered sequentially in order to decide where to attend for the next feature learning steps. More so residual learning [40] has been adopted to learn residual identity mapping. Residual learning increases the depth of feedforward neural networks. However, recent convolutional networks have become deeper but the key focus has been on training feedforward convolutional neural networks.

Attention Gates have been commonly used in natural image analysis, knowledge graphs, and natural language processing (NLP) for image captioning, machine translation and classification. Initial work has explored attention-maps by interpreting gradient of output class scores with respect to the input image [11]. Trainable attention, however, has been enforced by design and categorized as hard and soft attention.

Hard attention [41], is often non-differentiable and obtains parameter updates using reinforcement learning, which makes model training more difficult. Recursive hard-attention has been in image segmentation to [42] detect anomalies in chest X-ray scans. In contrast, soft attention is probabilistic and uses standard back-propagation with no need for Monte Carlo sampling. Channel-wise attention [43] has been used to highlight important feature dimensions, and was the top-performer in the ILSVRC 2017 image classification challenge. In order to remove the dependency on external gating, self-attention techniques [44] [45] have been proposed. In [44] [46] self-attention was used to perform class-specific pooling, resulting in higher accuracy and robust image classification performance.

In Attention U-Net the authors proposed a self-attention gating module that can be utilized in CNN based image analysis for dense label predictions which is adopted in this project. In the CNN, the convolutional layers progressively extract higher dimensional image representations ($x^l$) by processing local information layer by layer separating pixels in a high dimensional space according their semantics. As a result model predictions are conditioned on information collected from a large receptive field.

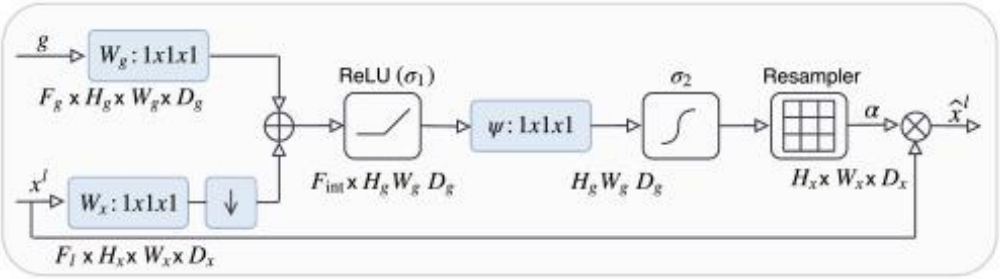



*Fig 6 Additive Attention Gate (AG). Input features ($x_l$) are scaled with attention coefficients ($\alpha$) computed in AG. Analyzing the activations and contextual information provided by the gating signal (g) (collected from a coarser scale) selects the spatial regions.*

## 4.4.2. Attention Gates for Image Analysis

The attention gates are used to reduce the false positive predictions for small objects that show large shape variability. This does not require the training of multiple models and a large number of extra model parameters. AGs progressively suppress feature responses in irrelevant background regions without the requirement to crop a ROI between networks. Attention coefficients, $\alpha_i \in [0,1]$, identify important regions in the image and prune feature responses to preserve only the relevant activations. The output of AGs is the element-wise multiplication of input feature-maps and attention coefficients: $\hat{x}_{i,c}^l = x_{i,c}^l \cdot \alpha_i^l$. The default setting is a single scalar attention value computed for each pixel vector $x_i^l \in \mathbb{R}^{F_l}$where $F_l$ corresponds to the number of feature-maps in layer l. Although this is computationally more expensive, it has experimentally shown to achieve higher accuracy than multiplicative attention [47]. Additive attention is formulated as follows [11]:

$$q_{att}^l = \psi^T\left(\sigma_1\left(W_x^T x_i^l + W_g^T g_i + b_g\right)\right) + b_\psi$$
$$\alpha_i^l = \sigma_2\left(q_{att}^l\left(x_i^l, g_i\,;\,\theta_{att}\right)\right)$$

where $\sigma_2(x_{i,c}) = {}^1/_{(1+\,e^{-x_{i,c}})}$ correspond to sigmoid activation function.
The parameters learnt by the gate include linear transformations: $W_x \in \mathbb{R}^{F_l \times F_{int}}, W_g \in \mathbb{R}^{F_g \times F_{int}}, \psi \in \mathbb{R}^{F_{int} \times 1}$ and bias terms $b_\psi \in \mathbb{R}, b_g \in \mathbb{R}^{F_{int}}$.

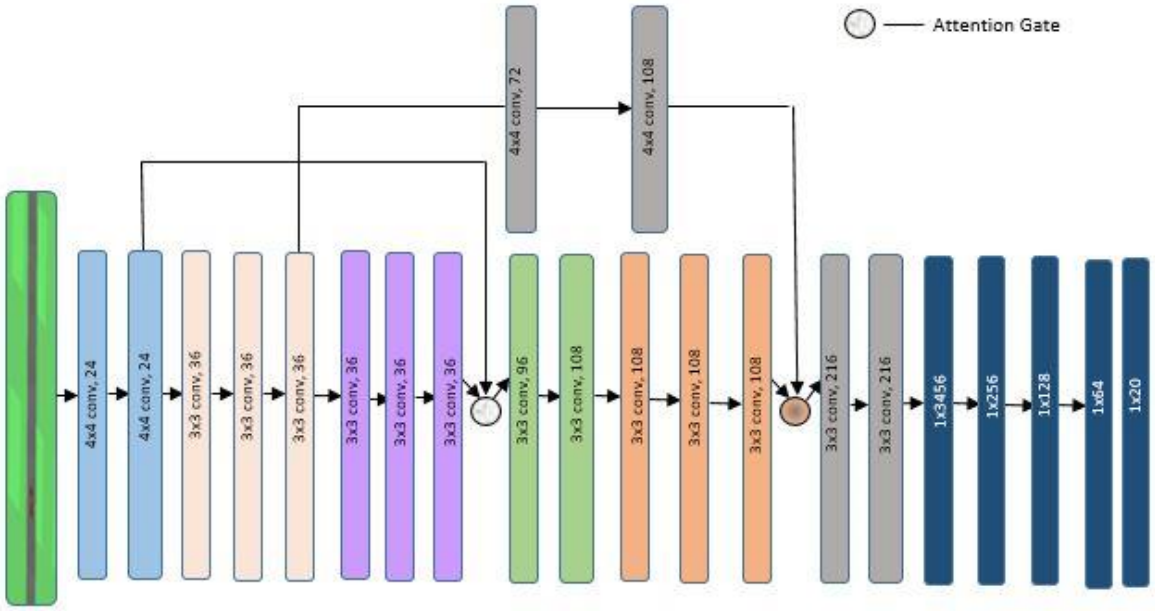
Channel-wise 1x1x1 convolutions are used to compute the linear transformations. This is referred to as *vector-concatenation-based-attention* [45].

During the forward and backward passes AGs filter the neuron activations. The gradients originating from the background regions are down weighted during the backward pass. As a result model parameters in shallowed layers can be updated based mostly on spatial regions that are relevant to the given task.

The update rule for convolution parameters in layer $l-1$ can be formulated as follows:

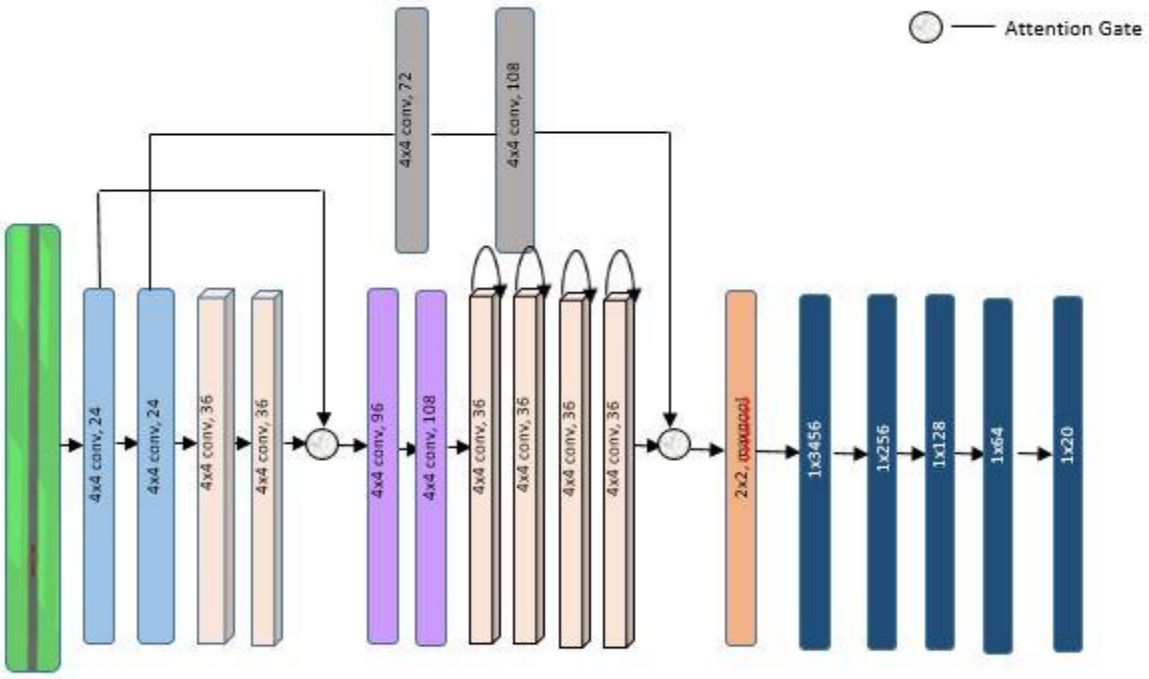
$$\frac{\partial(\hat{x}_i^l)}{\Phi^{l-1}} = \frac{\partial\left(\alpha_i^l\, f(x_i^{l-1};\,\Phi^{l-1})\right)}{\partial(\Phi^{l-1})} = \alpha_i^l \frac{\partial\left(f(x_i^{l-1};\,\Phi^{l-1})\right)}{\partial(\Phi^{l-1})} + \frac{\partial(\alpha_i^l)}{\partial(\Phi^{l-1})}\, x_i^l$$

## 4.4.3. Attention Subroutine




In this section I introduce Attention DAgger another variant of the iterative no-regret algorithm. I added attention gates to enable suppressing irrelevant regions in an input image while highlighting salient features useful for the task. With the application of attention gates to visual processing it was my hope to improve the robustness and accuracy of DAgger. Successful implementation of Attention Mechanisms to DAgger would lead to high accuracy and increase the robustness of DAgger. While preserving the online nature of DAgger, in this subroutine model, spatial regions are selected by analyzing both the activations and contextual information provided by the gating signal in the attention gate. The output of the AGs are element-wise multiplication of input feature-maps and attention coefficients. The network architecture is shown above.

## 4.4.4. Recurrent Subroutine with Attention Gates




In this section, I provide a subroutine for the no-regret online learning algorithm in the hope of yet improving the performance of the algorithm. The purpose of this subroutine was to combine the benefits of recurrent connections including context modulation, parameter reduction and feature reuse; with the benefits of attention gates including highlighting salient features and suppressing background features. During the experiments I discovered that the Attention subroutine did not perform satisfactorily with increasing DAgger iterations. I suspected that the presence of the sigmoid activation function in the Attention Gates resulted in vanishing gradients/ exploding gradients. However, since recurrent connections are a form of residual connections, this subroutine is able to attenuate this problem. In the experiments, I show the performance improvements from applying this procedure to DAgger. However, due to the attention mechanisms it is also more computational expensive than the purely Feed Forward Convolutional Neural Network Subroutine.

The network architecture is shown above.

# Chapter 5

# Experiments

In order to extend the functionality of the convolutional neural network I combined attention modules with recurrent connections. The proposed networks are. In this section I present the results of experiments on these subroutines on DAgger.

The recurrent convolutions are within the layers and provide context modulation for lower layers of the neural network while the Attention Gate improve the sensitivity and accuracy of the system by limiting the activations on background features and highlighting activations at salient regions of the image.

## 5.1. Dataset

I performed experiments using the CarRacing-V0 OpenAI gym environment. This environment was chosen due to its simplicity. Additionally, it was possible to generate expert trajectories since a traditional PID controller was used. The general setup of the experiment was first to train our model to generate a robust policy using the Convolutional Neural Network. The network is trained with Batch Normalization for regularization and then trained with dropout and L2 Regularization. The initialization method for the network was chosen empirically. Having observed the performances of Xavier and Kaiming Initialization techniques, Xavier Initialization was adopted. The experiments were run on pytorch with the Nvidia GeForce GTX 1070 GPU.

Following this the DAgger algorithm was used to generate its optimal policy based on this feedforward Convolutional Neural Network. The policy was trained on 10 DAgger iterations. Each DAgger iteration consisted of training the model for 50 epochs. After I had created a robust policy that was performing on the track, this process was repeated for the Recurrent Convolutional Neural Network, the Convolutional Neural Network with Attention Gates as well as the Recurrent Convolution Networks with Attention Gates.

# 5.2. CARLA Environment

Prior to adopting CarRacing-V0 environment I with the CARLA environment. I attempted to learn the controller of the agent using a convolutional neural network consisting of 6 layers; including 3 5x5 convolutional layers with 2x2 strides and two 3x3 convolutional layers with 1x1 strides. For the classifier, I used 3 fully connected layers. Batch normalization was performed after the convolution operation while the dropout rate was arbitrarily set to 50%. The learnt controller was able to predict the actual steering command using a Mean Squared Error as actual steering commands were required.
The environment was difficult to work with. More so, most of the time the vehicle was stationary at red traffic lights. This coursed the network to be biased resulting in an under fitting of the model. More so, the data collection was tedious, as a result I switched to the Car Racing environment as the objectives were similar.

# 5.3. CarRacing-V0 Environment

## 5.3.1. OriginalDrivingPolicy (Feed Forward CNN)

In the experiment I ran the original Feed Forward Convolutional Neural Network called OriginalDrivingPolicy; which was able to perform Behavior Cloning on the CarRacing-V0 environment [13]. The model consisted of 4 convolutional layers with; 4x4 convolutional kernels, 2x2 strides and 1x1 padding. Different from the experiment with the CARLA environment, the goal of this network was to minimize the cross entropy error. This was because the steering commands had been bucketed into ranges.

Without Regularization the network was effective in predicting the steering commands of the agent with loss of 0.49021 & validation accuracy 85.09898%. The agent was able to navigate across the maze with almost no difficulty. The agent was able to earn a reward of about 822.29999. In order to ensure the model was not overfitting, I experimented with the network by introducing Drop out and L2 Regularization.

I experimented with drop out values of 0.25, 0.2 and 0.1, then empirically chose 0.1 for a better model performance. With drop out the validation accuracy fell to 74.8005% with the loss rising to 0.90532. With L2 regularization at a decay rate of 10% the loss was 0.44022 and validation accuracy 85.29986%. Following these, I ran this Convolutional Neural Network as the subroutine of DAgger and observed loss and accuracy. In order to ensure the model was not overfitting I used L2 Regularization while running DAgger. I obtained the cumulative rewards of the agent. Following these I changed the color tracks of the environment to observe the behavior of the agent on the environment as a simulation of another environment.

### 5.3.2. RecurrentDagger

In the experiment I ran the Recurrent Convolutional Neural Network to perform Behavior Cloning on the CarRacing-V0 environment. The subroutine model consisted of 2 convolutional blocks and 4 recurrent blocks. Each convolutional block had 2 convolutional layers and 1 Batch Normalization layer. In order to improve model generalizability the Batch Normalization operation was performed between the convolutional layers. Each convolution layer had 4x4 convolutional kernels, with strides of 2x2 and padding of 1x1. I set the number of recurrent operations to be 4 as upon experimentation, I found 4 to give better results. Finally the classifier consisted of 4 Fully Connected layers.

Layer 1 is the standard feed-forward convolutional layer part of a convolutional block without recurrent connections. Two Recurrent Blocks proceed a Convolutional Block. The output of the fourth convolutional block is flattened.

Again the goal of this model was to minimize the cross entropy error as the steering commands had been bucketed into ranges. I followed the same procedure as with the Feed Forward Convolutional Neural Network discussed earlier including; creating models with and without regularization.

Without Regularization the network was effective in predicting the steering commands of the agent with a loss of 0.356654 & validation accuracy of 85.68999%. The agent was able to navigate across the track with almost no difficulty. The agent was able to earn a reward of 819.09999. Dropout and L2 Regularizations were performed to curb overfitting with a dropout and weight decay rate of 10%.

I proceeded to run this Recurrent Convolutional Neural Network as the subroutine of DAgger and observed loss and accuracy. In order to ensure the model was not overfitting I used L2 Regularization. I obtained the cumulative rewards of the agent. Following these I changed the tracks of the environment to observe the behavior of the agent on the environment.

### 5.3.3. AttentionDAgger

The AttentionDAgger followed the same procedure as the previous subroutines by experimenting with and without regularization techniques and implementing the model as the subroutine for DAgger. The subroutine model had a superior performance to the OriginalDrivingPolicy with a loss of 0.45178 and validation accuracy 86.4787%. As in the previous subroutines regularization techniques were adopted to curb overfitting. However, the agent could not navigate along the entire track earning a cumulative reward of 141.7070537.

More so, on running DAgger, the agent could navigate along the entire track with a higher loss and a lower accuracy. As such I experimented with a weight decay 0.05% which did not significantly improve the performance of the agent. I suspected that the presence of the sigmoid

function in the Attention Gate might have resulted in vanishing or exploding gradients, hence the reason for the performance. The model consisted of Convolutional Blocks and 2 Attention Gates.

The Attention Gate consisted of 3 convolutional layers. One of these convolutional layers had a sigmoid function in order to obtain the attention coefficients. The gating vector is used for each pixel to determine focus regions. The gating vector within the Attention Gate contains contextual information to prune lower-level feature responses. Besides the attention gates, the subroutine had 6 convolutional blocks; 3 of which had 3 convolutional layers called Conv_block_3 in the code. The other 3 convolutional blocks called Conv_block_2 in the code had 2 convolutional layers. There exists a convolutional residual block which is also a Conv_block_2. Following the feature extraction were 5 Fully Connected Layers. As was the case with the Recurrent Variant, Batch Normalization is performed between convolutional layers. Following these I changed the tracks of the environment to observe the behavior of the agent on the environment.

### 5.3.4. RecurrentDagger with Attention Gates

The RecurrentDAgger + Attention Gates followed the same procedure as the previous subroutines by experimenting with and without regularization techniques and implementing the model as the subroutine for DAgger. The subroutine had a superior performance to the OriginalDrivingPolicy with a loss of 0.24539492 and validation accuracy 87.2661%. As in the previous subroutines, regularization techniques were adopted to curb overfitting.

The agent was however, not able to navigate the track in one iteration, receiving a reward of 136.59664. I assumed this was due to the same problem the Attention subroutine had. However, on running it as the subroutine for DAgger the agent was able to navigate through the track earning a full reward of 819.2999.

I propose that the recurrent connections were able to attenuate the vanishing gradient problem with more DAgger iterations. Borrowing from the previous designs, this subroutine contained 3 convolutional blocks, 4 recurrent convolutional blocks, a max pool layer and 2 Attention blocks.

# Chapter 6

# Conclusion and Further work

In this research project I have examined existing feedforward, recurrent and attention networks for the purpose of imitation learning. These architectures were incorporated as the subroutine to the DAgger algorithm. DAgger approach provides a learning reduction with strong performance guarantees in imitation learning. More so, from the experiments, I saw the robustness of these subroutines as well as the shortcomings especially of the Attention Network subroutine which was combated using recurrent connections.

In future, I will consider other approaches to the implementing Attention including hard attention and projection. More so, in further work I intend on implementing recurrence and convolutions in other imitation learning architectures like EnsembleDagger with necessary changes individually and not combined.

# APPENDIX

# TABLES

| ***Subroutine Model*** | ***Loss No Regularization*** | ***Accuracy No Regularization*** | ***Loss With Weight Decay*** | ***Accuracy With Weight Decay*** | ***Loss With Dropout*** | ***Loss Without Dropout*** |
|---|---|---|---|---|---|---|
| *Feedforward CNN* | 0.49021 | 85.09898 | 0.4402293 | 85.29986 | 0.90532 | 74.8005 |
| *Recurrent CNN* | 0.45178 | 86.4787 | 0.4554855 | 82.912315 | 0.73844 | 77.0163 |
| *Attention CNN* | 0.356654 | 85.6899 | 0.334244 | 85.89058 | 0.681275 | 78.1254 |
| *Recurrent CNN + Attention Gates* | 0.245395 | 87.26614 | 0.3561768 | 84.36359 | 0.642239 | 78.2827 |

*Table comparing losses and accuracies with and without regularization*

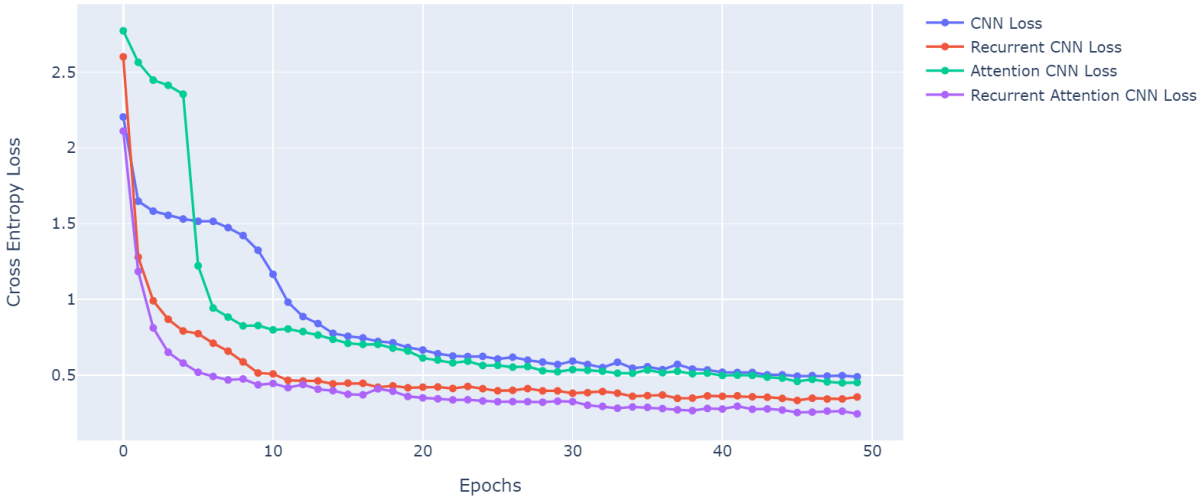
| ***Subroutine Model*** | ***Losses*** | ***Validation Accuracy*** |
|---|---|---|
| *Feedforward CNN* | 0.36157 | 87.571 |
| *Recurrent CNN* | 0.25705 | 85.8069 |
| *Attention CNN* | 1.66521 | 83.7345 |
| *Recurrent CNN + Attention Gates* | 0.25433 | 83.30785 |

*Table comparing losses and accuracies with L2 Regularization over DAgger iterations*

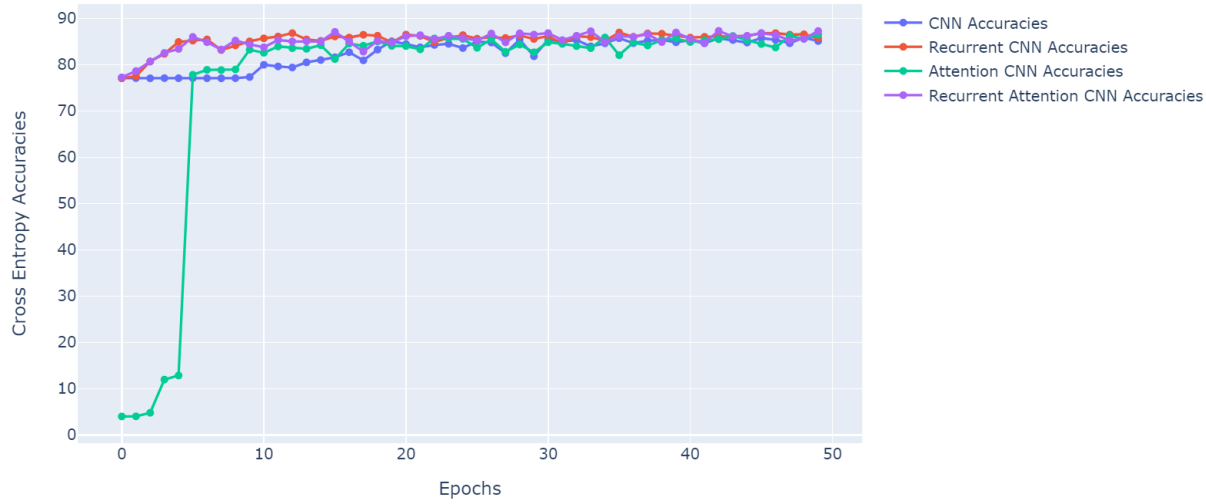
| ***Subroutine Model*** | ***Normal Tracks*** | ***Green Tracks*** | ***Purple Tracks*** | ***Blue Tracks*** |
|---|---|---|---|---|
| *FEEDFORWARD CNN* | 819.4999 | 819.9999 | 81.38255 | 820.2999 |
| *RECURRENT CNN* | 819.7999 | 818.5999 | 818.6999 | 819.6999 |
| *ATTENTION CNN* | 819.6999 | 818.8999 | 818.7999 | 819.1999 |
| *RECURRENT CNN + ATTENTION GATES* | 819.2999 | 817.9999 | 818.6999 | 819.5999 |

# GRAPHS

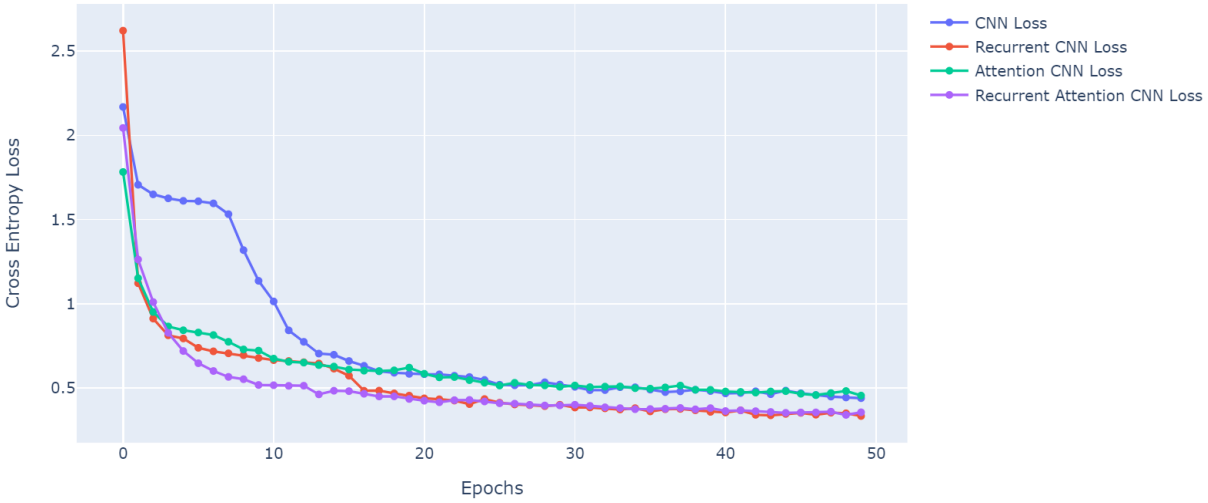
The Jupyter notebook has an interactive plot.



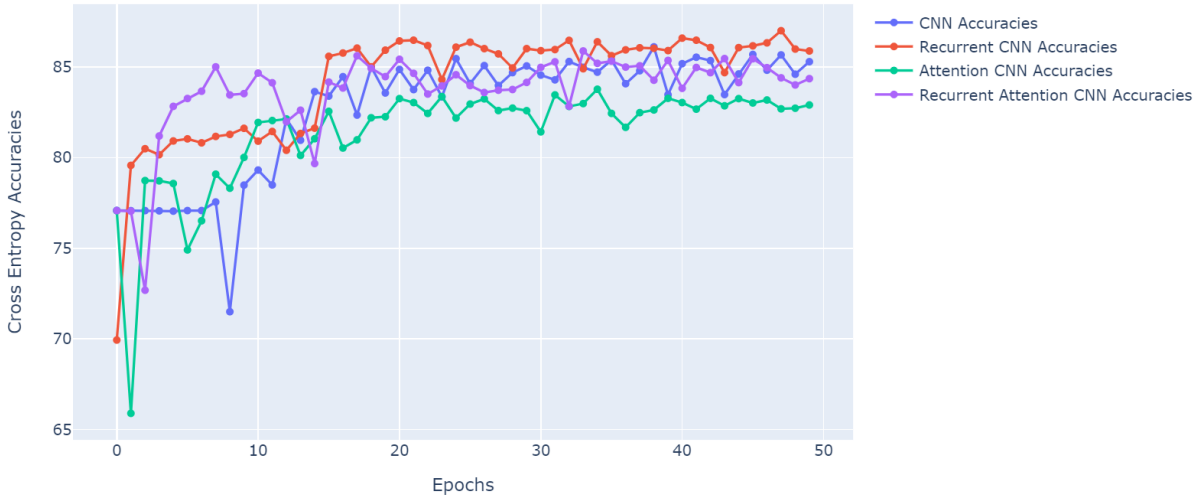

Loss without regularization after 50 epochs



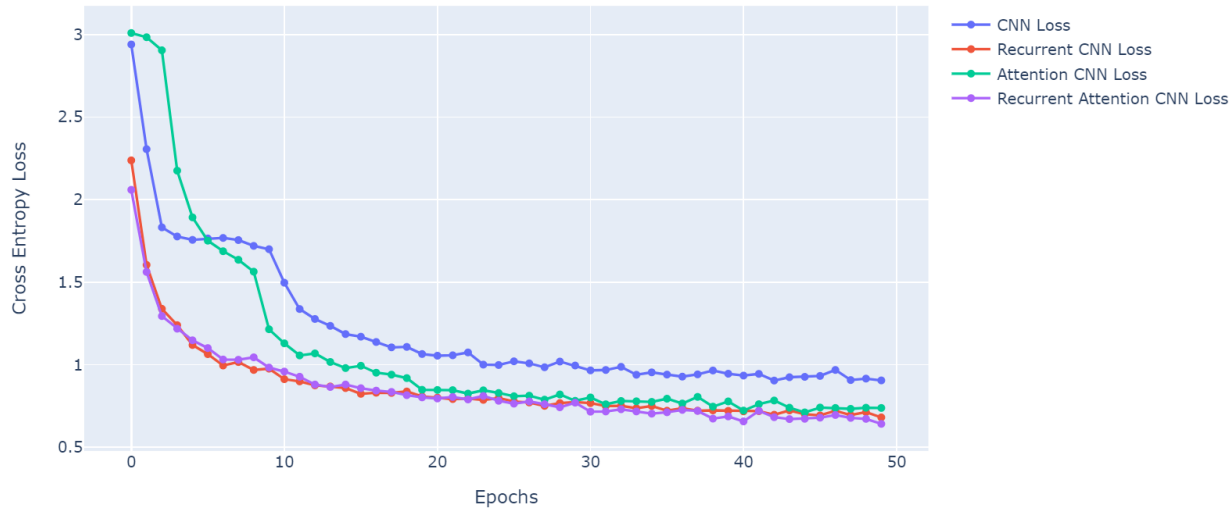

Validation Accuracies without regularization after 50 epochs



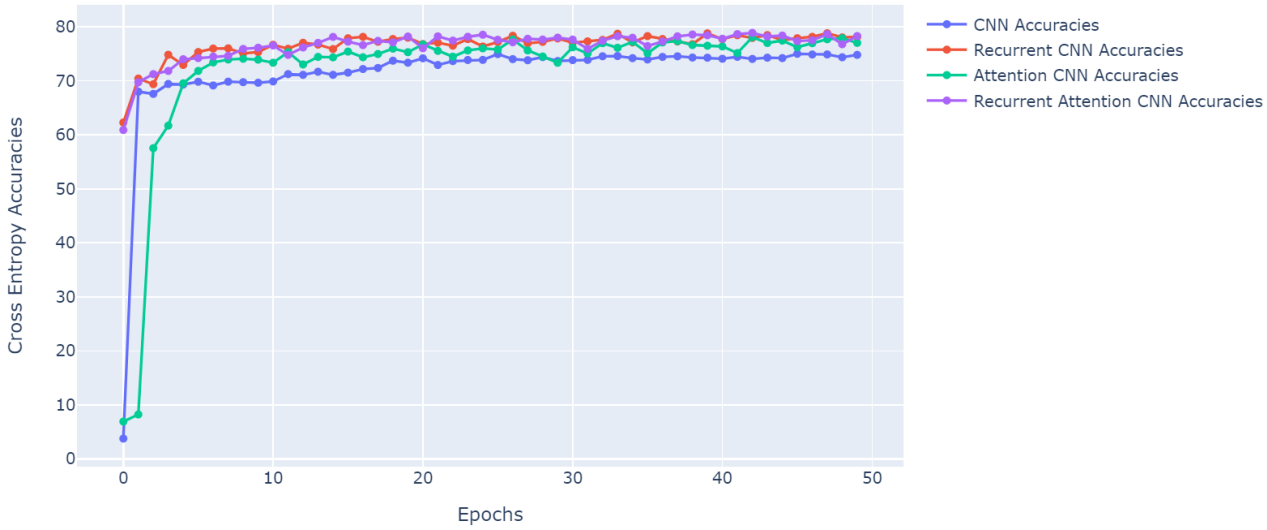

Loss with L2 regularization after 50 epochs



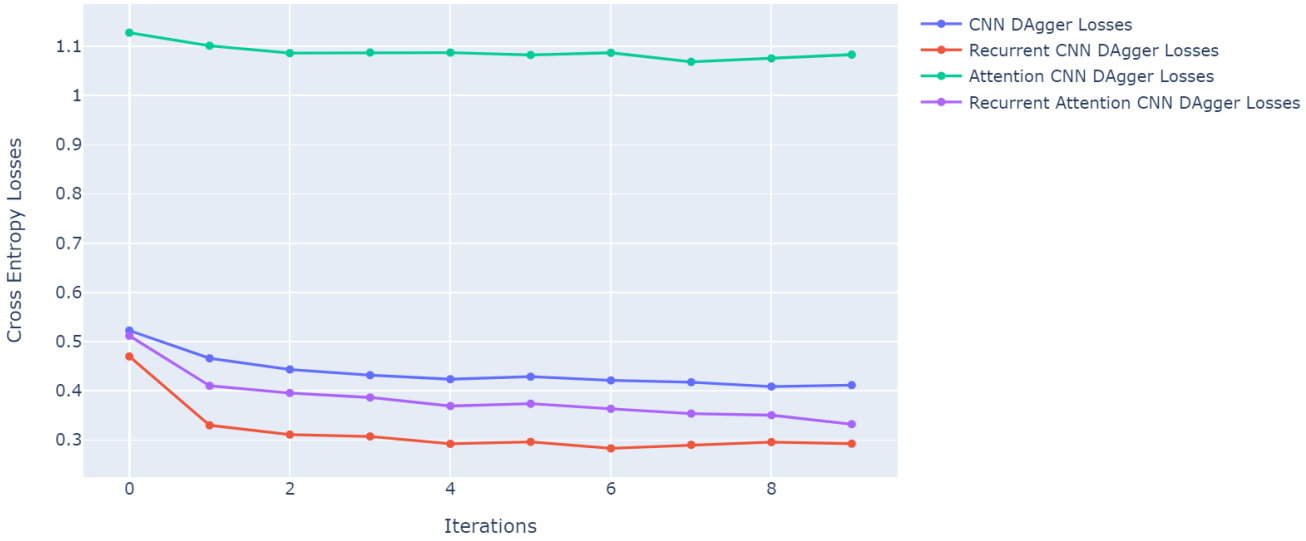

Validation Accuracies with L2 regularization after 50 epochs



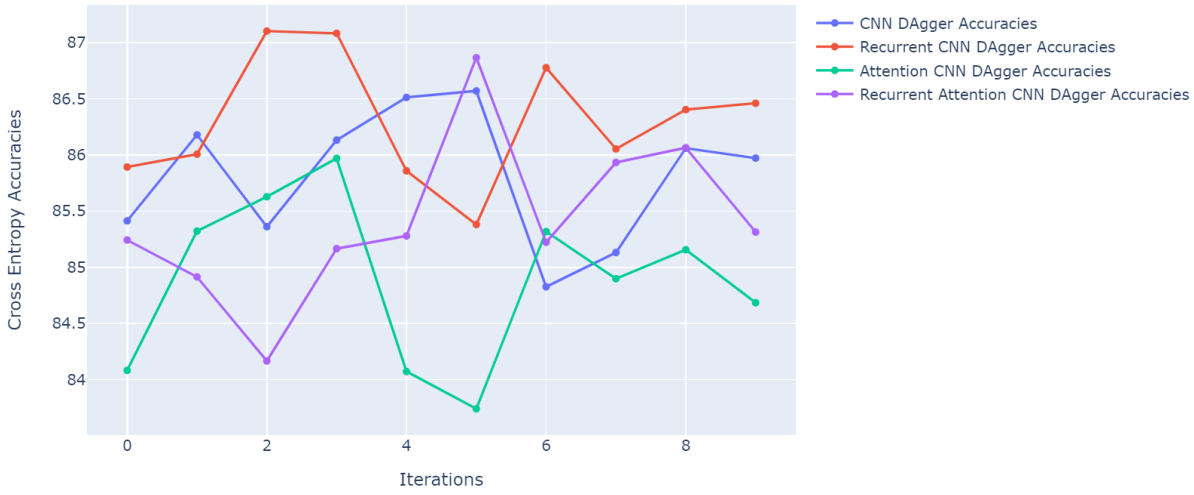

Loss with Dropout regularization after 50 epochs



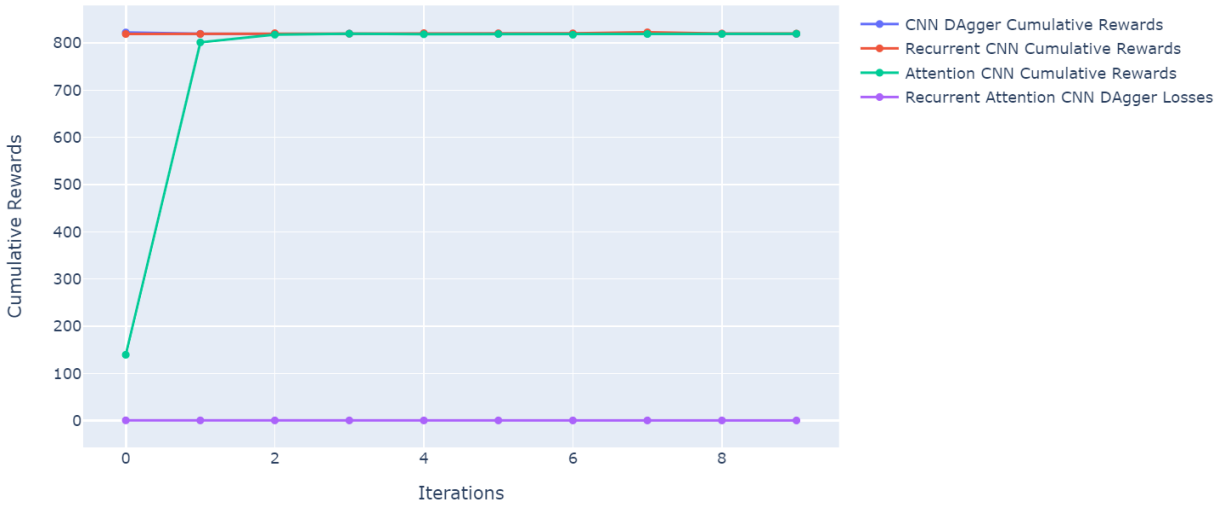

Validation Accuracies with Dropout regularization after 50 epochs




Losses after 9 Dagger Iterations with L2 Regularization




Validation Accuracies after 9 Dagger Iterations with L2 Regularization




Validation Accuracies after 9 Dagger Iterations with L2 Regularization

# References


[1] S. Schal, " Is imitation learning the route to humanoid," in *Trends in Cognitive Sciences*, 1999.

[2] P. Abeel and A. Y. Ng, "Apprenticeship learning via inverse reinforcement learning," in *ICML*, 2004.

[3] J. A. Bagnell, M. Zinkevich and N. Ratliff, "Boosting structured prediction for imitation learning," in *Advances in Neural Information Processing Systems (NIPS)*, 2006.

[4] D. Silver, J. A. Bagnell, "High performance outdoor navigation from overhead data using imitation learning," in *Proceedings of Robotics Science and Systems (RSS)*, 2008.

[5] B. D. Argall, S. Chernova, M. Veloso, and B. Browning, "A survey of robot learning from demonstration," in *Robotics Autonomous Systems* , 2009.

[6] S. Chernova and M. Veloso, " Interactive policy learning through confidence-based autonomy," 2009.

[7] S. Ross and J. A. Bagnell, "Efficient reductions for imitation learning," in *n Proceedings of the 13th International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2010.

[8] A. Beygelzimer, V. Dani, T. Hayes, J. Langford and B. Zadrozny, "Error limiting reductions between classification tasks," in *Proceedings of the 22nd International Conference on Machine Learning*, 2005.

[9] P. Dayan and L. F. Abbott, Theoretical Neuroscience, Cambridge MA: Cambridge, MA: MIT Press, 2001.

[10] T. D. Albright and G. R. Stoner, "Contextual influences on visual processing," in *Annual review of neuroscience*.

[11] Ozan Oktay, Jo Schlemper et al, "Attention U-Net: Learning Where to Look for the Pancreas".

[12] Alex Krizhevsky, Ilya Sutskever, Geoffrey Hinton "ImageNet Classification with Deep Convolutional Neural Networks," in *Advances in Neural Information Processing Systems*, 2012.

[13] M. Bojarski, David Del Testa, Daniel Dworaokowski et al "End to End Learning for Self-Driving Cars," 2016.

[14] Max Bajracharya, Andrew Howard, Larry H. Matthies, Benyang Tang & Michael Turmon, "Autonomous Off-Road Navigation with End-to-End Learning for the LAGR Program," *Journal of Field Robotics,* January 2009.

[15] K. Fukushima, "A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position," in *biological cybernetics*, 1980.

[16] M. Riesenhuber and T. Poggi, "Hierarchical models of object recognition in cortex," *Nature Neuroscience,* 1999.

[17] D. H. Hubel and T. N. Wiesel, "Receptive fields of single neurones in the cat's striate cortex," *The Journal of physiology,* pp. 571- 591, 1959.

[18] D. H. Hubel and T. N. Wiesel, "Receptive fields, binocular interaction and functional architecture in the cat's visual cortex," *The Journal of Physiology,* pp. 106-154-2, 1962.

[19] Xavier Glorot, Antoine Bordes and Yoshua Bengio, "Deep sparse rectifier networks," in *Proceedings of the 14th International Conference on Artificial Intelligence and Statistics*, 2011.

[20] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, "Dropout: A simple way to prevent neural networks from overfitting," in *Journal of Machine Learning Research* , 2014.

[21] Karen Simonyan, Andrew Zisserman, "Very deep convolutional networks for large-scale image recognition," in *CoRR*, 2014.

[22] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet et al " Going deeper with convolutions," 2014.

[23] P. Dayan and L. F. Abbott, Theoretical Neuroscience, Cambridge, MA: MIT Press, 2001.

[24] Thomas D. Albright and Gene R. Stoner, Contextual influences on visual processing, Annual review of neuroscience, 2002.

[25] X. H. Ming Liang, Recurrent Convolutional Neural Network for Object Recognition, 2015.

[26] Tai Sing Lee and David Mumford, "Hierarchical bayesian inference in the visual cortex," 2003.

[27] Honglak Lee , Roger Grosse, Rajesh Ranganath, Andrew Y. Ng, "Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations," in *Proceedings of the 26th Annual International Conference on Machine Learning (ICML)*, 2009.

[28] B. Fernandez,A.G. Parlos, W.K. Tsai " Nonlinear dynamic system identification using artificial neural networks," in *International Joint Conference on Neural Networks (IJCNN)*, 1990.

[29] G.V. Puskorius, L.A. Feldkamp, "Neurocontrol of nonlinear dynamical systems with kalman filter trained recurrent networks," in *IEEE Transactions on Neural Networks*.

[30] D. E. Rumelhart, G. E. Hinton, Ronald J William, "Parallel distributed processing vol. 1. chapter Learning Internal Representations by Error Propagation," in *Explorations in the microstructure of cognition*, 1986, pp. 318-362.

[31] Wikipedia, 2020. [Online]. Available: https://en.wikipedia.org/wiki/Recurrent_neural_network.

[32] Richard Socher, Christopher D. Manning, Andrew Y. Ng, "Learning continuous phrase representations and syntactic parsing with recursive neural networks," in *Advances in neural information processing systems (NIPS), Deep Learning and Representation*, 2010.

[33] Richard Socher, Cliff Chiung-Yu Lin, Andrew Y. Ng and Christopher D. Manning, "Parsing natural scenes and natural language with recursive neural networks," in *Proceedings of the 28th International Conference*, 2011.

[34] Volodymyr Mnih, Nicolas Heess, Alex Graves, Koray Kavukcuoglu, "Recurrent models of visual attention," in *NIPS*, 2014.

[35] Jonathan Long, Evan Shelhamer, Trevor Darrell, "Fully convolutional networks for semantic segmentation," in *CVPR*, 2015.

[36] Hyeonwoo Noh, Seunghoon Hong, Bohyung Han, "Learning deconvolution network for semantic segmentation," in *ICCV*, 2015.

[37] Rupesh Kumar Srivastava, Klaus Greff, Jurgen Schmidhuber, "Training very deep networks," in *NIPS*, 2015.

[38] Hugo Larochelle, Geoffrey E. Hinton, "Learning to combine foveal glimpses with a third-order boltzmann machine," in *NIPS*, 2010.

[39] Jin-Hwa, Kim Sang-Woo, Lee Donghyun Kwak et al, "Multimodal residual learning for visual qa," in *Advances in Neural Information Processing Systems*, 2016.

[40] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun, "Deep residual learning for image recognition," in *CVPR*, 2016.

[41] Volodymyr Mnih, Nicolas Heess, Alex Graves, koray kavukcuoglu, "Recurrent models of visual attention. In: Advances in neural information processing systems," 2014.

[42] Petros-Pavlos Ypsilantis, Giovanni Montana, "Learning what to look in chest X-rays with a recurrent visual attention model," 2017.

[43] Jie Hu, Li Shen, Samuel Albanie et al, "Squeeze-and-excitation networks," 2017.

[44] Saumya Jetley, Nicholas A. Lord et al, "Learn to pay attention," in *International Conference on Learning Representations* , 2018.

[45] Xiaolong Wang, Ross Girshick, Abhinav Gupta, Kaiming He, "Non-local neural networks," 2017.

[46] Fei Wang, Mengqing Jiang, Chen Qian et al. , "Residual attention network for image classification," in *IEEE CVPR*, 2017.

[47] Minh-Thang Luong, Hieu Pham, "Effective approaches to attention-based neural machine translation.," 2015.

[48] Hal Daumé III, John Langford, Daniel Marcu, "Search-based structured prediction," in *Machine Learning*, 2009.

[49] S. Kakade and J. Langford, "Approximately optimal approximate reinforcement learning," in *Advances in Neural Information Processing Systems*, 2009.