University of Glasgow | School of Computing Science

Evaluating Uncertainty Estimation Methods For Deep Neural Network's In Inverse Reinforcement Learning

Joe Kadi (2261087k)

April 15, 2021

ABSTRACT

Inverse Reinforcement Learning (IRL) is a machine learning framework which enables an autonomous system to learn an agent's objectives, values, or reward function by observing its behavior. In the problems where IRL can assist - such as autonomous driving - safety is paramount to mission success, as incorrect decisions can be fatal. Deep neural networks (DNN) have achieved exceptional performance in a range of important domains due to their unprecedented representational capacity. Therefore if DNN's are to be leveraged for real-world IRL applications, it's essential for them to provide an accurate estimation of uncertainty in order to guarantee safety. Three state-of-the-art methods to calibrate uncertainty in DNN's are Monte-Carlo Dropout, Stochastic Weight Averaging Gaussian (SWAG) and Ensembling. As such, this paper contributes insight into how each of these methods can be configured to solve the IRL problem as well as an comparative evaluation of the quality of their uncertainty estimates. SWAG emerges as the most promising method as it achieves the greatest accuracy on the IRL benchmark and preferentially overestimates it's uncertainty; ensuring a higher level of safety in comparison to other methods which underestimate it.

1. INTRODUCTION

Robots inherently take action within an uncertain world. In order to plan and make decisions, autonomous systems can only rely on noisy input data and approximated models. Wrong decisions not only result in the failure of the task but might even put human lives at risk when the application is safety critical i.e if the robot is an autonomous car. Therefore, in order to fully integrate deep learning algorithms into safety critical robotic systems, there must be an accurate prediction of uncertainty. This would enable the system to mitigate the risk in its decision making and/or delegate to a human operative if the uncertainty surpasses a pre defined threshold.

Inverse Reinforcement Learning (IRL) is a stochastic decision making problem to infer a latent reward function r that a demonstrator subsumes through observing its demonstrations or trajectories D in the task. The learned reward function can then be used to uncover a policy π and reproduce the agents behaviour. IRL's state-of-the-art performance comes from the Adversarial Imitation Learning [38] and Deep Neural Network (DNN) approaches [36]. However these approaches do not calibrate uncertainty thus are unable to represent the models ignorance about the world. This renders them undesirable for deployment in safety critical robotic systems. The IRL problem has also been tackled using Gaussian Processes (GPs) [13, 22]. GPs are renowned for their ability to propagate uncertainty estimates through Bayesian inference of unknown posteriors, however [13, 22]'s research focuses on leveraging GP's to optimise non-linear IRL performance and does no exploration of calibrating the models uncertainty predictions for IRL. Moreover, GP's dependence on a kernel machine makes them not scale well to large state spaces with complex reward structures and prone to requiring a large number of demonstrations D [4], unless assisted by special algorithms and models such as sparse GP's [29]. This is reflected in their computation complexity of $O(n^3)$ at query time [35].

Alternatively, Neural Networks (NN) are deep learning (DL) models that can approximate the reward function r from the state feature representation X through by minimising the error between the predicted state rewards and the expected rewards, captured by the IRL maximum entropy log likelihood (eq 1). NN's already achieve state-of-the-art performance across a variety of tasks, including Computer Vision [33] and Reinforcement Learning [34], and provide a computational complexity of $O(1)$ at query time with respect to observed demonstrations D [36]. Bayesian Neural Networks (BNN) are regular NN's that represent their weights and biases through a probability distribution through leveraging a prior [15]. This enables them to provide uncertainty estimates about their predictions [16], lending them well for real-life IRL applications with large state spaces and complex reward structures. BNN's can be straightforward to construct but the Bayesian inference is the tricky part and can produce an excessive computational cost if the correct inference technique is not used [15]. Two conservative state of the art solutions for approximate Bayesian inference in BNN's is Monte Carlo (MC) Dropout [8] and Stochastic Weight Averaging Gaussian (SWAG) [25] which can be used to supply calibrated uncertainty estimates in DL models without the burden of undesirable computational complexity or test accuracy. Another less computationally conservative and not strictly Bayesian method to calibrate DNN uncertainty is Ensembling [6, 39, 33]. All three methods have been used and evaluated with respect to their ability to produce accurate uncertainty estimations on a range of DL problems [1, 16], but have never been rigorously explored within the IRL problem domain.

This work conducts experiments (section 6) in which heterogeneous noise is incrementally introduced into the expert demonstrations D and state feature representation X used by various IRL models to infer the latent reward function r *and* an associated uncertainty estimation. The evaluation

in section 7 finds that the every method is able to represent the uncertainty of its predictions. The results also indicate, according to the metrics (section 5), that MC Dropout, Ensembling and GP's generally underestimate the uncertainty with respect to their predictive error whereas SWAG to overestimates it. They also show that the uncertainty estimations are sensitive to the heterogeneous data noise, becoming more accurately calibrated when noise is added to more states (section 7).

As such, the main aim of this paper is to explore the hypothesis that MC Dropout, SWAG and Ensembling can enable a DNN model to calibrate accurate uncertainty related to its reward function prediction r. Through this investigation, this paper contributes the following:

- A novel evaluation of how SWAG, MC Dropout and Ensembling impact a DNN's ability to approximate a latent reward function r from expert demonstration D and a state feature representation X. (section 7)
- A novel evaluation of MC Dropout's, SWAG's and DNN Ensemble's ability to calibrate accurate uncertainty in IRL. (section 7)
- Insight into if the calibrated uncertainty estimations are representational of both the epistemic and aleatoric uncertainty (section 7) and a clear process to learn them individually is outlined (section 8.2)
- Novel proof that SWAG can work with and represent reasonably accurate uncertainty in IRL using the first order stochastic optimiser: Adam [14] (section 7)
- A visual explanation of how DNN and BNN's can be configured to solve the IRL problem and estimate the epistemic uncertainty of their predictions. (section 4)

In order to gain these insights, the following artefacts have been created and can be viewed as technical contributions:

- A novel Python framework to train, tune and evaluate the various DNN and BNN models on the Objectworld IRL benchmark problem (section 6). The framework allows noise to be added into the state features X and/or demonstrations D in order to prime for an uncertainty calibration. It has been built to easily enable the integration of other models, objective functions and benchmark problems.
- An IRL specific metric Expected Normalised EVD Error (section 6) to evaluate the quality of uncertainty estimations.

2. INVERSE REINFORCEMENT LEARNING

This section provides an overview of the IRL problem and the many proposed approaches to tackle it. IRL was spawned as a response to the reward engineering principle present in traditional Reinforcement Learning (RL). Dewey [7] first coined the phrase, defining it as: *"As reinforcement-learning-based AI systems become more general and autonomous, the design of reward mechanisms that elicit desired behaviours becomes both more important and more difficult."*

2.1 Reinforcement Learning

RL is an area of Machine Learning which aims to solve sequential decision making problems under uncertainty [34]. The problem is generally modelled as a Markov Decision Process (MDP). A MDP can be characterized by the tuple $M = \{S, A, \tau, \gamma, r\}$, where S is the state space, A is the set of actions, $\tau_{s'}^{sa}$ is the probability of transitioning from $s \in S$ to $s \in S'$ under the action $a \in A$, $\gamma \in [0, 1]$ is the discount factor, and r is the reward function. The optimal policy π^* maximizes the expected discounted sum of rewards $E[\sum_{t=0}^{\infty} \gamma^t r_{s_t} | \pi^*]$ [21]. However, RL assumes the existence of a pre-defined reward function. This is problematic as some reward functions are too abstract thus too difficult to define. For large environments the agent's reward function can be extremely complex, thus too computationally intense to define [7]. Moreover, handcrafting useful reward functions can require expert knowledge - especially when the policy we wish to uncover is extremely specific to the actions i.e a small change in the selected actions can produce a large change in the end result. [22]

2.2 Early Inverse Reinforcement Learning

IRL was first described by [27]. IRL's goal is to infer a reward function r that produces an optimal policy which matches the supplied demonstrations D. By learning the reward function this way, IRL was seen to have the potential to overcome RL's dependency on a pre-defined reward function. Given that an MDP specification is available, IRL is defined as $M = \{S, A, T, \gamma, D\}$ where D represents the experts demonstrations and can be denoted as $D = \{\zeta_1, ..., \zeta_N\}$ where path ζ_i takes the form $\zeta_i = \{(s_{i0}, a_{i0}), ..., (s_{iT}, a_{iT})\}$ [27].

Although the original formulations of IRL [27] can somewhat achieve their objectives, they are still restrained by some major assumptions that make them not suited for practical applications:

1. The experts demonstrations are assumed to be optimal. This is usually not true in practice, especially when learning from human demonstrations. An ideal IRL algorithm should be able to handle non-optimal and noisy demonstrations.
2. The demonstrations are assumed to be complete and plentiful. Sometimes only a few demonstrations can be supplied, and these may be incomplete. Thus effective IRL algorithms should be able to generalize demonstrations to uncovered areas and be able to learn from few demonstrations.

Enabling a model to provide an accurate representation of uncertainty, (sections 3 and 4), appears a promising solution to handle non-optimal and noisy demonstrations (assumption 1). Levine attempted to overcome assumption 2 through building a GP [22] (section 2.4) but this paper demonstrates how a DNN's exceptional representational capacity can be leveraged to learn from fewer demonstrations than Levine's GP and provide more accurate uncertainty estimations (sections 4 and 7). Prior work has demonstrated DNN's ability to generalise to unseen data once trained [4, 34, 36].

2.3 Maximum Entropy IRL

To learn from noisy demonstrations, the Maximum Entropy approach regards the reward function as the parameters for the policy class [40]. Specifically, it applies the principle of maximum entropy - which gives the least bias estimate based on the information supplied [31] - to IRL. This method allows for the likelihood of observing the demonstrations to be maximised under the true reward function and by following [22], the complete log likelihood of the experts demonstrations D under reward function r can be expressed as:

$$logP(D|r) = \sum_i \sum_t logP(a_{i,t}|s_{i,t}) = \sum_i \sum_t (Q^r_{s_{i,t},a_{i,t}} - V^r_{s_{i,t},a_{i,t}}) \quad (1)$$

This log likelihood, eq 1, is the objective function that every BNN built (section 4) will seek to minimise in order to uncover the true r. Using this technique, the probability of taking a path ζ is proportional to the exponential of the rewards encountered along that path. The above equation neatly captures this logic as the likelihood of observing action a in state s is shown to be proportional to the expected total reward after taking action a, denoted by $P(a|s) \propto exp(Q^r_{sa})$ where $Q^r = r + \gamma\tau V^r$. The value function V is computed by a "soft" version of the well-known Bellman's backup operator: $V^r_s = log \sum_a expQ_{sa}$. Therefore, the likelihood of a in state s is normalized by $expV^r$, thus $P(a|s) = exp(Q^r_{sa} - V^r_s)$.

The true r that the demonstrating agent D was equipped with can be found through maximising eq 1. This approach has been shown able to learn from sub-optimal human demonstrations [40, 36], mainly due to it's ability to overcome the problem of label-bias which previous IRL algorithms struggled with. Label-bias is when portions of the state space with many branches will each be biased to being less likely whilst areas with fewer branches will have higher probabilities (locally greedy) [40]. The consequence of this is that the most rewarding policy may not obtain the greatest likelihood. Maximum Entropy IRL avoids label bias as it focuses on the distribution over trajectories rather than actions, as we can see in eq 1.

However, this approach still represents r as a linear combination of provided state features, thus is not expressive enough to accurately represent a reward function which is non-linear in features [40].

2.4 Gaussian Process IRL

GP's have been widely applied within IRL to represent non-linear reward functions. Levine et al [22] initially did this using a Bayesian GP framework since it supplies a systematic method for learning the kernel's hyper-parameters, in turn learning the structure of the latent reward function. Eq 1 is then used to define a distribution over the GP output, thus learning the output values and kernel function. The GP [22] creates for IRL is represented in:

$$P(\mu, \theta|D, X_u) \propto P(D, \mu, \theta|X_u) = \left[\int_r P(D|r)P(r|\mu, \theta X_u)dr\right] P(\mu, \theta|X_u) \quad (2)$$

Typically in GP regression noisy observations, y, of the true outputs, u, are used. From inspecting eq 2 we see that the rewards of states, u, are learned from the corresponding state feature set X_u.

The log of $P(D|r)$ is supplied through eq 1, the GP posterior $P(r|u, \theta, X_u)$ is the probability of a reward function given the current values of u and θ. The prior probability, $P(u, \theta|X_u)$ is the probability of the current values of u and θ given the state feature set X_u. The GP log marginal likelihood $P(u, \theta|X_u)$ has a preference for simple kernel functions and values of u that align with the current kernel matrix [35].

The GP posterior is Gaussian with mean of $K^T_{r,u}K^{-1}_{u,u}U$ and co-variance $K_{r,r} - K^T_{r,u}K^{-1}_{u,u}K_{r,u}.K_{r,u}$ which represents the co-variance of the rewards at all states with inducing points u, located at X_r and X_u [35]. Since the reward function structure is unknown the GP keeps it flexible and non-linear by modelling it with a mean θ value of 0 [22]. Due to the complexity of the $P(D|r)$ term, it's integral - seen in eq 2 - cannot be calculated in closed form. As a result Levine [22] performs a sparse GP regression approximation [29], with the training conditional being the Gaussian posterior distribution over r.

In doing so, the training conditional approximate to be deterministic thus has a zero variance [35]. Using this approximation we can remove the integral and set $r = K^T_{r,u}K^{-1}_{u,u}U$. Following [22], the final log likelihood can then be calculated from the sum of the IRL and GP log likelihoods, expressed by:

$$logP(D, \mu, \theta|X_u) = logP(D|r = K^T_{r,u}K^{-1}_{u,u}U) + logP(\mu, \theta|X_u) \quad (3)$$

This GP implementation is able to approximate the reward for states that aren't represented in the feature set, preserving computation when dealing with complex state spaces and enables the GP to work on partially-observable environments.

Although, the computation saved here is minute in comparison to the computation expended through the GP's use of a kernel machine. Eq 2 learns the kernels hyper-parameters θ in order to uncover the structure of the latent reward function. The final values of u and θ are approximated through maximising their likelihood under the expert demonstrations D [22]. The use of a kernel machine enables the GP to capture complex, non-linear reward functions from large state spaces, but simultaneously disables it to scale well to larger state spaces with complex reward structures highlighted in it's undesirable computation complexity at query time of $O(n^3)$, where n denotes the number of unique states in the current set [35].

While GPIRL can, in theory, still model these nonlinear, complex reward functions, the cardinality of the state space can quickly become excessive which at best could place GPIRL at a significant computational disadvantage or ultimately render it intractable. These computational limit is overcame when using DNN's (section 4). Although, GP's inherent ability to propagate accurate uncertainty estimates through Bayesian inference of unknown posteriors makes Levine's GPIRL's [22] an appropriate model to use as a baseline comparison (section 7)

3. UNCERTAINTY IN DEEP NETWORKS

This section will review other instances of uncertainty evaluation in DNN's to motivate the method (section 4). Also due to the current inconclusive nature surrounding concrete definitions of epistemic and aleatoric uncertainty, this section will clearly outline the definitions of each within the

IRL setting that will be used throughout this paper.

3.1 Uncertainty Calibration In Deep Networks

Uncertainty calibration is a well researched field for classification problems, such of weather forecasting [32] and DL [16]. However, uncertainty calibration for regression is much less studied and only recently has research emerged offering methods to calibrate and evaluate it [8, 25, 20]. As observed in section 2, current IRL research solely focuses on optimising performance whilst offering no insight into calibrating accurate uncertainty, which has been discussed to be crucial for stochastic decision making problems in order to mitigate the risk in the autonomous systems' decision making.

As such, the proposal to evaluate uncertainty estimation methods for DNN's within IRL is motivated. From reviewing literature on the topic [16, 1, 11, 20], it appears that the state of the art, and most popular, methods for calibrating uncertainty in deep networks are Monte-Carlo Dropout [8], Stochastic Weight Averaging Gaussian [25] and Ensembles [33]. Although other methods prevailed in the literature - such as Concrete Dropout [10], Temperate Scaling Dropout [18] and Bootstrapping Ensembling [30] - this paper will out scope and sake focus on examining the top three methods for the sake of ensuring a rigorous investigation.

In DNN's, uncertainty can be modelled by alternative probability distributions over the networks parameters. For example, a Gaussian prior distribution could be placed over the networks weights and instead of optimising the network weights directly, an average over all possible weights would be taken. This process is referred to as marginalisation and the resulting model is referred to as a Bayesian Neural Network [15]. From the techniques evaluated in this research, sections 4 and 7, SWAG 4.4, MC Dropout 4.3 and GP's [22] can be regarded as Bayesian approximators whereas Ensembles 4.2 cannot. Section 4 details how each method was configured for IRL.

3.2 Types of Uncertainty

Total uncertainty can generally be seen as the combination of two sub-types, aleatoric and epistemic:

1. **Epistemic Uncertainty** refers to *model uncertainty* arising from an inaccurate measure of the world which captures a models ignorance about which model generated the input data. In the context of IRL the input data is the demonstrations D, generated by the observing an agent performing a task and the state feature representation X, generated by the ObjectWorld state feature generating algorithm. Epistemic uncertainty is also broadly known as the *reducible* part of the total uncertainty as it can be explained away with more or better quality information since it represents things one could know in principle but do not know in practice [11].

2. **Aleatoric Uncertainty** refers to *data uncertainty*. It represents a measure of noise that is inherent in the observations i.e the variability in the outcome of an experiment which is brought about by inherently random effects. In the context of IRL this would refer to noise inherent in the demonstrations D and state feature representation X. For this reason, aleatoric uncertainty refers to the *irreducible* part of the total uncertainty as it can only be modelled, but not reduced since it represents knowledge of that which cannot be determined sufficiently [11]. Homoscedasitc aleatoric uncertainty indicates homogeneous data noise whereas the heteroscedastic aleatoric uncertainty indicates heterogeneous data noise, which will be more prevalent in these experiments given that hetereogenus data noise will be added into the state feature representation X and demonstrations D.

Using the above definitions, the experimental procedure (section 6) and the metrics (section 5), this paper will investigate if the total estimated uncertainty from the various methods is accurately calibrated with respect to the predictive error and with respect to the noise added into the state features X and demonstrations D (section 7). The ability of the estimated uncertainty to represent both the epistemic and aleatoric uncertainty will also be evaluated under the following observation: more sampled paths should reduce the epistemic part of this total uncertainty but a lower bound of uncertainty should exist for the states with added noise which represents to the aleatoric uncertainty related to the introduced noise.

4. METHODOLOGY

This section will firstly describe how a DNN can be built for reward function and approximation in IRL based on the feature representation of MDP states. It will then describe how each uncertainty calibration method was implemented into the model. All design choices made during implementation have been motivated and justified.

4.1 Reward Function Approximation with Deep Networks

Figure 1 depicts the overarching network architecture and schema used when training the various DNN's, described in the proceeding sections, to solve the IRL problem. While many choices pertain for the individual components of the deep architecture, it has been demonstrated that a relatively large network with as few as two layers and a sigmoid activation function can capture any binary function and thus can be considered a universal approximator [3]. While this is true, it is much more computationally efficient to increase the depth of the network (add more layers) in order to decrease the number of needed computations [3]. Moreover, much work has shown that the Rectified Linear Unit (ReLU) activation function is more computationally efficient than the sigmoid activation [26], as it mitigates the vanishing gradient problem prevalent in sigmoid and tends to achieve better convergence performance than sigmoid [26]. Although the ReLU activation function has been shown to have a tendency to blow up activation since it does not provide a mechanism to constrain the output of the neuron [24]. To mitigate this, the step size can be lowered and an L2 regularizer (weight decay) can be used [24].

As such the network architecture chosen (depicted in figure 1) has an input layer with a neuron for each feature, two hidden layers with a linear ReLU activation function after each and an output layer with a neuron for each feature. This network architecture was necessary in order to fit the IRL back propagation function proposed by [22] to calculate the gradient of the log likelihood function with respect to the network's weights, which is essential to solve the ObjectWorld IRL benchmark problem.

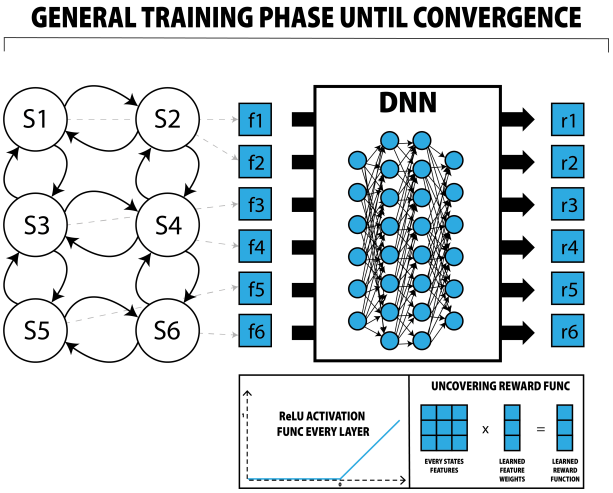

Figure 1: General training schema for DNN IRL reward function approximation based on the feature representation of MDP states. S represents a state, F represents the state feature description and R represent the estimated reward of that state.

The function depends on a feature expectation matrix (muE) which has the state visitation frequency D subtracted from it to obtain the true gradient. A bottleneck architecture with one output neuron has been shown to improve DNN task accuracy [36], but to enable this here would require the size of (muE) to match the number of states. Doing this causes muE to lose all its meaning thus rendering the gradient value incorrect. PyTorch's auto gradient capabilities [28] were trialed as a replacement but always obtained sub-optimal results in comparison to when using Levine's ObjectWorld specific gradient function. Thus in order to enable a single output neuron architecture that improves performance, a new back propagation function would have to be built which was outwith the scope of this project.

Using this architecture, the network learns a feature weight distribution W_f which assigns an importance value to each state feature. The product of W_f and the entire state feature set X is taken to uncover the learned reward r (figure 1).

Finally, l2 regularizer (weight decay) is used to avoid the exploding gradient problem that can occur when using the ReLU activation function as previously discussed. Before training, a random search was used to tune the networks hyper-parameter values.

The training phase (figure 1) seeks to minimise the MaxEnt log likelihood function (eq 1). The back propagation function proposed by Levine in [22] (discussed above) is used alongside an Adam optimizer [14] with a decaying learning. This training process is repeated until the networks weight values converge. One important thing to consider is that DNNs naturally suit training in the MaxEnt IRL framework and the network architecture can be adapted to fit individual tasks without confusing or invalidating the main IRL learning mechanism. This is a crucial consideration for the proceeding sections as slight modifications will be made to the networks architecture in order to suit each uncertainty calibration method (section 4.3, 4.4).

4.2 Ensembles

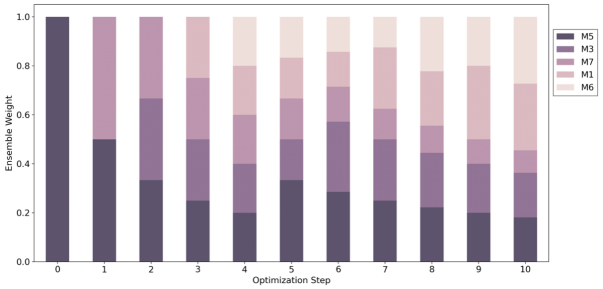

Figure 2: Example ensemble model weights selection process using [6]'s ensemble selector algorithm built from the best performing out of 10 pre-trained models

Ensembling was first introduced in [39] as an alternative to Bayesian inference techniques, due to its implementation simplicity, the fact it requires very little hyper-parameter tuning and has displayed better predictive accuracy than popular Bayesian methods in many cases [1, 39, 6, 17]. This could be because ensembles inherently explore diverse modes in function space which is contrary to some Bayesian methods which are inclined to focus on a single mode [1].

Generally ensemble techniques fall into two main classes: randomization-based where the ensemble members can be trained in parallel without interacting with each other and boosting-based where the ensemble members are fit sequentially. This work will implement and evaluate the boosting-based technique proposed by [6] which can generally be described as a model-free greedy stacking. This is because at every optimization step [6]'s algorithm either adds a new model to the ensemble or changes the weights of the current members to minimise the total loss without any individual model guiding the selection process. Given a set of trained networks and their predictions, their ensemble construction algorithm is as follows:

1. Set init_size — the number of models in the initial ensemble and max_iter — the maximum number of iterations
2. Initialize the ensemble with init_size best performing models by averaging their predictions and computing the total ensemble loss
3. Add to the ensemble the model in the set (with replacement) which minimizes the total ensemble loss
4. Repeat Step 3 until max_iter is reached

This method guarantees a strong ensemble as it initializes the ensemble from several high accuracy networks and drawing models with replacement guarantees that the ensemble loss will not increase as the optimization steps progress. In the case where no additional model will improve the total ensemble loss, the algorithm will add copies of the existing ensemble members with adjusted weights. This feature allows the algorithm to be thought of as "model-free stacking". The trained building block networks had their initial parameters randomised in order to de-correlate the predictions of the ensemble members [19]. An outline of the ensemble model's weight selection process can be seen in figure 2.

Breiman [5] showed that correlation between ensemble members leads to an upper-bound of their accuracy, thus

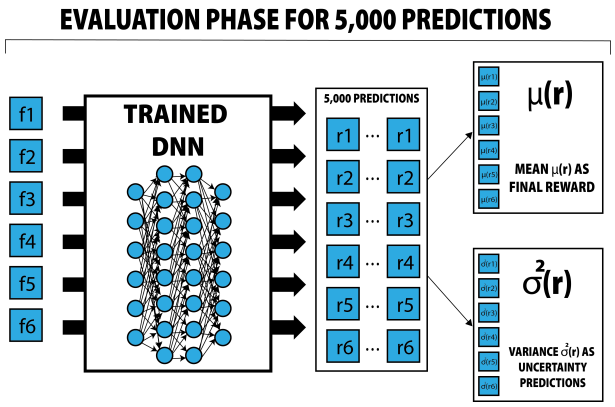

Figure 3: Evaluation process for each network in the ensemble. F represents the state feature description and R represent the estimated reward of that state.

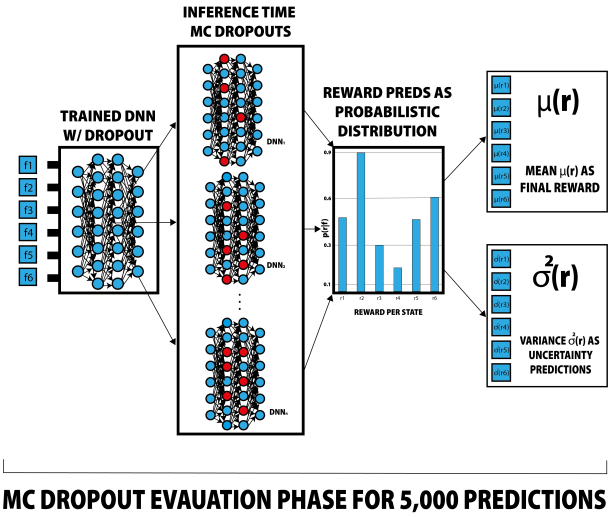

Figure 4: Schema for DNN IRL reward function and uncertainty approximation using MC Dropout based on the feature representation of MDP states. S represents a state, F represents the state feature description and R represent the estimated reward of that state. A red neuron represents that is "deactivated".

it is advantageous to use a randomization method that decorrelates the predictions of the ensemble members and guarantees they are accurate. Breiman [5] recommends bootstrapping [30] to achieve this, where the individual models are trained on different bootstrap samples of the original training set. Bootstrapping is useful to encourage variety but can hinder uncertainty estimate accuracy since a model trained on a bootstrap sample sees at most only 63% unique data points [17]. [19] later demonstrated that training on an entire data set with random weight initialization was better than bootstrapping for ensembles. However the goal of that research was to improve predictive accuracy not predictive uncertainty, thus it is worthwhile to explore if the finding is consistent for predictive uncertainty.

Deep ensembles is a popular, sophisticated ensembling method [17] that has been proven to yield high quality predictive uncertainty estimates on a range of problems [33, 17]. Deep ensembles was considered in this research but including learning the variance σ^2 in the loss function to achieve "the proper scoring rule" would require refactoring the Max Ent IRL log likelihood function (1) and since it has already been shown to produce calibrated uncertainty in many regression problems, it seems more worthwhile to explore a less researched ensembling method, like [6] ensemble selector method. Only 10 trained models were used to construct the ensemble in this work to preserve computation and since Ashuka proved [1] that in many cases an ensemble of only a few independently trained networks is equivalent using many.

The network architecture and training protocol for each ensemble member is described in section 4.1 and seen in figure 1. Once trained, each model is to make 5,000 reward function predictions and the mean, $\mu(r)$, of all predictions is taken as the final prediction and the variance of each, $\sigma^2(r)$, as the predictive uncertainty estimates, depicted in figure 3.

4.3 Monte Carlo Dropout

Monte-Carlo Dropout Variational Inference was first proposed in [9] as a practical method to estimate model uncertainty built upon the popular dropout regularization technique. It has been used and evaluated on a wide variety DL problems [8, 23, 1, 18, 20] and has been shown to underestimate the predictive uncertainty - a property many variational inference methods share [2]. The premise of dropout is to deactivate a portion of random neurons every forward pass. Regular dropout is only applied at training time to serve as a regularization technique to avoid over fitting, thus the learned model can be seen as an average of an ensemble and its predictions are deterministic. MC Dropout differs as it is applied at both training and test time. The models predictions are no longer deterministic as a random subset of neurons are deactivated every prediction thus it's predictions can be represented as a probabilistic distribution. The authors call this Bayesian interpretation [9]. The process to obtain IRL policy predictions and subsequent uncertainty estimates used in this research can be seen in figure 4.

The network architecture and training protocol used here is in line with the section 4.1 and figure 1 except a dropout layer is added before every weight layer.

4.4 Stochastic Weight Averaging Gaussian

Stochastic Weight Averaging (SWA) was first proposed in [12] and has been shown to improve network generalization and performance in a wide variety of applications with no extra computational cost [37, 25]. There are two main components at play to enable this regularizer. 1) A modified learning rate which makes the optimizer (Adam in this case) bounce around the optima and explore a variety of models instead of converging to a single solution once the optima is reached and 2) An average of all model weights learned and stored at the end of every epoch within the last 25% of training. These averaged weights then create the final model.

Stochastic Weight Averaging Gaussian (SWAG) does everything SWA does but also calculates a low rank plus diagonal approximation to the co-variance of the SWA model weights, which is used together with the SWA mean, to define a Gaussian posterior approximation over NN weights [25].

Thus, SWAG is can be seen as an approximate Bayesian inference algorithm and the subsequent uncertainty estimates

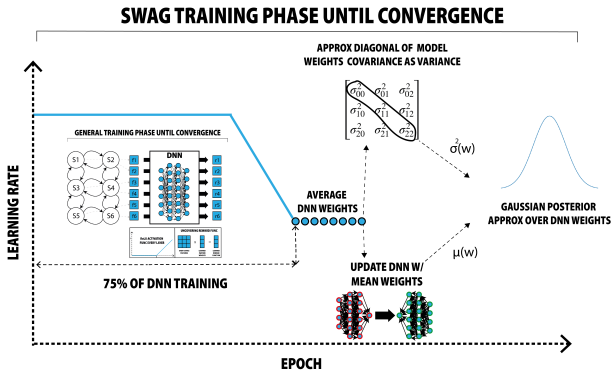

Figure 5: SWAG training procedure for BNNIRL reward function and uncertainty approximation. Can see an average of all model weights being taken in the last 25% of training where the optimiser bounces around the optimal learning rate instead of converging to single solution. A low rank plus diagonal approximation to the co-variance of the SWA model weights is used as the estimated uncertainty.

produced when applied to DNN's (section 7) can be regarded as Bayesian. This training process is depicted in figure 5.

The learned BNN's policy predictions π^* and uncertainty estimates are obtained through evaluated depicted in figure 3. The creators of SWAG proposed it to work well with the SGD optimizer but proceeding work has shown it to also successfully work with other first order stochastic optimizers such as Adam [14]. To maintain consistency and comparability in the experiments outlined in section 6, SWAG was implemented with an Adam optimizer. This research can also be seen as further insight into SWAG's ability to work with first order stochastic optimizers outwith SGD.

5. METRICS

This section discusses the details of each metric used to evaluate each model's performance and the quality of it's subsequent uncertainty estimates, seen in chapter 7.

5.1 IRL Performance

5.1.1 *Expected Value Difference*

Expected Value Difference (EVD), initially proposed by Levine et al in [22], will be the main metric used to measure the optimality of the approximated r, expressed by:

$$EVD = E\left[\sum_{t=0}^{\infty}\gamma^t r(s_t)|\pi^*\right] - E\left[\sum_{t=0}^{\infty}\gamma^t r(s_t)|\hat{\pi}\right] \quad (4)$$

Which can be interpreted as the difference between the expected reward given by true optimal policy π^* and the policy learned, $\hat{\pi}$, from the IRL rewards r. It is preferable to compare the differences in learned policies rather than reward functions as many reward functions - varying in scale - can produce identical policies. A lower EVD represents a more accurate predicted policy.

5.2 Predictive Error Calibrated Uncertainty

Error-based metrics directly compare the empirical prediction error to the uncertainty estimate and will be used a measure of the total discrepancy with respect to perfect calibration: when the empirical error equals the uncertainty estimate at every state.

5.2.1 *Expected Normalized Calibration Error (ENCE)*

Originally proposed by Levi et al in [20], ENCE is calculated by first dividing the predicted action per state, π^*, and subsequent uncertainty estimate of the prediction, σ^2, into N non-overlapping bucket. For each bucket j, the root mean variance (RMV): $RMV(j) = \sqrt{\frac{1}{|B_j|}\sum_{t\epsilon B_j}\sigma_t^2}$ is compared to the empirical root mean square error of the predicted and true policy: $RMSE(j) = \sqrt{\frac{1}{|B_j|}\sum_{t\epsilon B_j}(y_t - \hat{y}_t)^2}$ in:

$$ENCE = \frac{1}{N}\sum_{j=1}^{N}\frac{|RMV(j) - RMSE(j)|}{RMV(j)} \quad (5)$$

The RMSE is computed for IRL based on the predicted policy π^* not the reward r as many reward functions - varying in scale - can produce identical policies.

For well calibrated uncertainty predictions we would expect the RMSE of each buckets policy predictions to be equal to the RMV of each buckets uncertainty predictions. This measure is analogous to the Expected Calibration Error (ECE) used in classification, a lower ENCE represents a better calibrated uncertainty estimator. The main advantage of ENCE is that it directly relates estimated uncertainty to expected error thus reflecting what the user expects, seen in the reliability diagram in section 7. The main limitation is that since only a subset of the uncertainty estimates contribute to each bucket and the uncertainty estimates are not uniformly distributed, the subsets used to compute the different buckets are not homogeneous. Which brings to bear the need for a second error-based metric.

5.2.2 *Expected Normalised EVD Error (ENEE)*

ENEE was created as a a support metric for ENCE. It is essentially identical to ENCE (eq 5) but substitutes the general regression error metric RMSE with an IRL specific metric EVD (eq 4). ENEE is described in eq 6.

$$ENEE = \frac{1}{N}\sum_{j=1}^{N}\frac{|RMV(j) - EVD(j)|}{RMV(j)} \quad (6)$$

5.3 Input Noise Calibrated Uncertainty

ENCE and ENEE only give insight into if the estimated uncertainty is calibrated per predictive error i.e is the model aware of it's own mistakes? This is useful as an overall measure of calibration, but does not give insight into if the uncertainty is calibrated with respect to the input noise. Given the various ways noise was incrementally introduced 3 subsets of states (section 6), in order to determine this type of calibration Welch's t-test will be used to discover if there is a significant statistical difference between the average uncertainty estimate for the states with added noise and the average uncertainty estimate for every state. Welch's variation of the t-test will be used since the population size, thus variances, of the uncertainty estimates are not equal. A regular students t-test was used in the case where noise was added to 50% of the states, since population sizes would be equal. The null hypothesis at test here is "the mean uncertainty for the 2 sets of states are equal". Following conventional criteria a significance level of 5% will be used and combined with the degrees of freedom of 254 gives us a critical t value

of 1.98. Given all the above, a t value > 1.98 would indicate that the uncertainty **was** calibrated with respect to the input noise with 95% confidence and a t value < 1.98 would indicate that it **was not**. If the t value drops below -1.98 this would indicate that the mean uncertainty was significantly greater for the non-noisy states, demonstrating a very poorly calibrated uncertainty estimator w.r.t to the input noise.

5.4 Dispersion

ENCE and ENEE alone could be insufficient to fully evaluate a set of uncertainty estimates. If a model was to predict a constant uncertainty for every state which equalled it's empirical prediction error, it would obtain an ENCE and ENEE of 0: a perfectly calibrated uncertainty estimator but not a very descriptive one since it isn't contingent on the input data. In this case, a measure of estimated uncertainty dispersion would be useful. Levi et al introduced this concept in [20] where the Coefficent of Variation, C_v, was used as a measure of dispersion of the estimated uncertainties:

$$c_v = \frac{\sqrt{\frac{\sum_{t=1}^{T}(\sigma_t - \mu_\sigma)^2}{T-1}}}{\mu_\sigma} \tag{7}$$

A higher C_v corresponds to more heterogeneous uncertainty estimates for different inputs, which should be the case for state spaces with a diverse spread of input noise. Thus for the experiments seen in Chapters 6 and 7 a higher C_v is better as it would indicate that the model captured the range of noise present in the input data. A lower C_v would represent homogeneous uncertainty estimates indicating that the model is equally unsure about all its predictions, which shouldn't be the case in these experiments.

6. EXPERIMENTAL OUTLINE

This section provides an outline of each experimental protocol and the expected results. Each experiment was run 3 times until convergence and the BNN architectures, training and evaluation procedures were consistent as per section 4.

6.1 Objectworld Benchmark

In order to evaluate each methods ability to represent both linear and non-linear reward functions, they will be experimentally evaluated on a simple linear MDP environment and an established IRL benchmark problem "Objectworld". Objectworld was initially proposed by Levine et al in [22] and consists of an M x M grid space with the possible actions being movements in any of the 4 directions, as well as not moving, thus remaining in the same state. Dots of primary and secondary colours are randomly placed on the grid. The binary features representing each state simply describe the shortest distance to dots of each colour. The hidden reward function is allocated as such: if a state is 1 step from a red dot and 3 steps from a blue dot, the reward is 1; if it is 3 steps from a blue dot only the reward is -1; otherwise the reward is 0. The IRL agent then learns a reward function solely from seeing demonstrations through the object world, given a state feature representation. This reward function is then used to generate a policy which provides a probability of each action at each state, with respect to transition probabilities. This policy can then be used by the agent to navigate the grid world with the goal of maximising the accumulated reward. The "Object World" benchmark configuration can be seen in Figure 6.

ObjectWorld lends itself well to evaluate IRL algorithms' ability to learn and represent abstract reward structures. Due to the large number of irrelevant features and the non-linearity of the reward [22], this benchmark is challenging for methods that learn linear reward functions such as the Maximum Entropy [40].

6.2 Evaluating IRL Performance

To evaluate each models' ability to solve the IRL problem, they will be evaluated on ObjectWorld with an increasing number of paths, D, and the quality of their learned reward function determined through their obtained Expected Value Difference from eq 4. Every model will see the same set of paths, D, and be trained until convergence. They will then be evaluated by making 5,000 reward function predictions, r, on a similar set of features and the mean μ will be taken as their final reward function prediction and the variance, σ^2, as their uncertainty estimate. The DNN and GP based models would be expected outperform the MaxEnt models with respect to the EVD of their uncovered reward function. The regularization methods (4) should result in a longer training time until convergence for the DNN's but should enable greater evaluation accuracy vs a regular DNN.

6.3 Evaluating Uncertainty Estimations

To evaluate each models' ability to calibrate accurate epistmeic uncertainty estimates, a modified version of a 256x256 state ObjectWorld benchmark was used. Heterogeneous noise was introduced into the demonstrations D and into the state feature representation X to prime for and give a basis to evaluate the uncertainty estimations. All noise was incrementally introduced and experimented on the same subset of states close in feature space - 12.5%, 25% and 50% of states - in order to make results comparable. Every set of uncertainty estimates were evaluated and compared with respect to the metrics in section 5, using Levine's GPIRL [22] as a baseline. Visual insights were also used to aid the evaluation. The results are shown and analysed in section 7.

6.3.1 Noisy Paths

Noise was introduced into 50% of the sampled paths by replacing the desired noisy state with a state sampled from another MDP with similar transition dynamics in order to keep the paths as natural as possible whilst still adding noise. The replacement state had to be outwith the current subset of desired noisy state but within the set of possible 256 states to be a valid replacement. In the case where a valid state was unable to be found, a random valid state was selected as the replacement. To maintain some natural behaviour, only 50% of paths were made noisy so that the loss and gradient functions could be some-what accurately calibrated thus giving the models a higher chance of actually detecting the noise.

6.3.2 Noisy Features

Noise was introduced into the states feature description X by simply inverting the feature based on a criteria. Features were inverted, not randomized, since they are binary. For each state in the current subset of states to add noise to, there was a 97% chance the state was to be made noisy then for each of it's 60 binary features, there was a 22% chance of inversion. The filters percentage of 97% and 22% were learned to be the minimal amount of noise filtering

required in order to prevent the loss and gradient values from exploding and preventing the models from solving the IRL problem.

The increased sensitivity to state feature noise, when compared to noise in the demonstrations D, could be explained by the fact the features are used to calibrate the loss and gradient functions ***and*** the model directly uses them to predict a reward value per state as apposed to the demonstrations D that are only used to calibrate the loss and gradient functions. This sensitivity should also cause feature noise to be more readily detected thus yielding more accurate uncertainty estimates than those arising from noise in the sampled paths D, which are only used to calibrate the loss and gradient functions.

Similarly to the noisy paths, the same set of noisy features were used for every experiment and each set of uncertainty estimates will be evaluated in an identical fashion, using metrics from section 5, in order to render a valid comparison.

6.3.3 *Noisy Paths & Features*

Noise will be added into the features and paths simultaneously in order to prime for an approximated "total" uncertainty [16]. New models of each method (section 4) will be built and trained with the noisy states and features and then evaluated on the true state feature representation X.

6.3.4 *Evaluation & Expected Results*

How each method impacts a DNN's ability to solve the IRL problem will be evaluated visually and with respect to the EVD metric seen in eq 4 and required number of demonstrations D. All uncertainty estimations will be evaluated against the ENCE and ENEE metrics, (eq 5 and 6), in order to gain insight into if they are accurately calibrated with respect to their predictive error. Variations of a students t-test and the coefficient of variation metric, (section 5.3 and eq 7), will be used to evaluate the accuracy of uncertainty calibration with respect to the added noise in the states features X and demonstrations D.

In order to determine if the calibrated uncertainty represents both the aleatoric and epistemic uncertainty, the following observation can me made: a lower number of noisy sample paths should lead to higher variability of every models predictions and uncertainty-estimations. As per the definitions in section 3.2, more sampled paths should reduce the epistemic part of this total uncertainty but a lower bound of uncertainty should exist for the states with added noise which represents to the aleatoric uncertainty related to the introduced noise.

6.4 Configuration

Every method (section 4) and all the experiments (section 6) were built in a novel implementation using the PyTorch framework [28]. In every case, each model's hyperparameters were tuned with a random grid search with the dropout probabilities being fixed at 0.5 and 1.0 to conserve computation. For every experiment, each model was trained until convergence and the experiment was run 3 times with different random seeds to ensure experiment validity. The mean, μ, was used as the true result values and the variance of the results, σ^2, was communicated in the plots (section 7). Each experiment was run on Macbook Pro (M1, 2020) with 16GB RAM and an 8-core AMD M1 CPU (PyTorch did not support execution on AMD GPU's at the time of experiments). Source code of can be viewed at: https://github.com/joekadi/Inverse-Reinforcement-Learning-Bayesian-Uncertainty.

7. RESULTS & DISCUSSION

Firstly error performance is detailed for the models under consideration in section 7.1. Next, the results for uncertainty estimation evaluation are presented and discussed in sections 7.3 and 7.4.

7.1 IRL Performance

It is interesting to investigate how each of the methods (section 4) impact the performance of solving the IRL problem. Figures 6 and 7 give some insight into this.

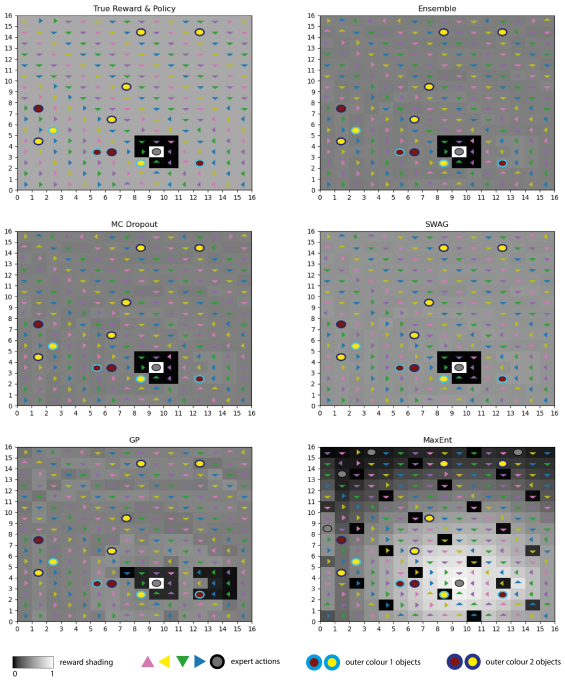

Figure 6: Learned r and π^* from each model on the OW IRL benchmark, evaluated with 1024 sampled paths D. The legend indicates what each object represents.

7.1.1 *Learned reward functions and policies*

Figures 6 shows that the various DNN models learn a smooth reward function (square shading) that better matches the true reward function compared to the MaxEnt and GP models. The MaxEnt's inability to uncover the non-linear reward function is expected since it assumes a linear reward structure, as described in section 2.3. GPIRL learns a slightly more accurate, yet still noisy reward function which is limited by feature discriminability and the number of demonstrations (sampled paths) D.

Conversely each of the DNN models demonstrate their exceptional representational capacity and infer a reward function and policy closest to the truth with the same binary state feature description X and number of demonstrations D used by GPIRL, as seen in figure 6. Although it has difficult to discern from 6 which DNN regularisation method yields the greatest accuracy. The EVD's achieved by each model seen in figure 7 offers this insight.

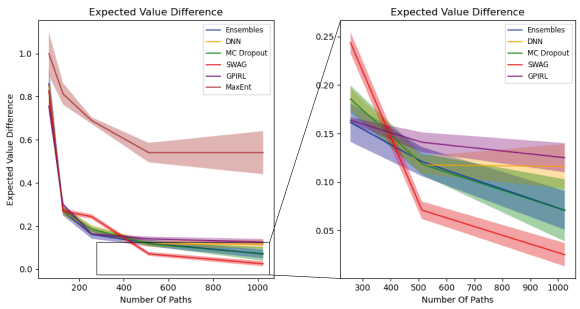

Figure 7: EVD values obtained from the OW benchmark from each model using an increasing number of demonstrations D

7.1.2 Expected Value Differences

The EVD's achieved by each model (figure 7) and give insight into which DNN method most accurately uncovers a reward function closest to the truth. It can be observed that SWAG is the most accurate, closely followed by MC Dropout then Ensembles. Figure 7 also shows that every regularisation technique actually improves performance when compared to a regular DNN, exhibited through lower EVD scores. Each of the methods have already been shown to increase a DNN's robustness and predictive accuracy, so it's intuitive that this is consistent with IRL. As expected, each models' EVD improves linearly alongside the number of sampled paths seen by the model.

Every DNN model achieves a lower EVD than GPIRL [22], which is consistent with figure 6 showing GPIRL converging with a less smooth reward function than the each DNN model and the true r. MaxEnt's inability to handle the non-linearity's present in the ObjectWorld benchmark's true reward function seen in figure 6 is consistent with it's high EVD score seen in figure 7. The shaded area around the lines (figure 7) represent the standard deviation of EVD's obtained from the three individual experiments run with different random seeds, indicating SWAG is the most reliable as well as the most accurate.

The analysis of IRL performance is not the main goal of this research but the insight is helpful in the proceeding discussion and should be considered as a significant consequence of uncertainty modelling.

7.2 Capturing Aleatoric & Epistemic Uncertainty

The bar charts seen in figure 8 confirm that the estimated total uncertainty does account for both the epistemic and aleatoric uncertainty through the following observation: more sampled paths D reduces the epistemic part of of the total uncertainty and the uncertainty that remains at 128 paths is accounted for primarily by the aleatoric part. The majority of this aleatoric uncertainty has a high chance of being the heteroscesdastic type since heterogeneous noise was added into the state feature representation X and sampled paths D.

In the case where no noise was added into the states, a lower total uncertainty is exhibited with the epistemic part being reduced with more demonstrations (paths) D. The uncertainty that remains as 128 paths when no noise is added has a high chance of being primarily comprised of the homoscesdastic aleatoric since the majority of epistemic uncertainty will have been explained away (reduced with more paths D) thus the remainder should represent the underlying homogeneous noise in the states (bearing in mind no heteregenous noise was added). Figure 8 also confirms that the added noise was responsible for a large portion of the calibrated uncertainty since the total uncertainty is consistently lower when noise was not added. It also indicates that MC Dropout and Ensembling capture more aleatoric uncertainty than the GP and SWAG through the smaller reduction of total uncertainty exhibited when increasing the number of paths from 64 to 128.

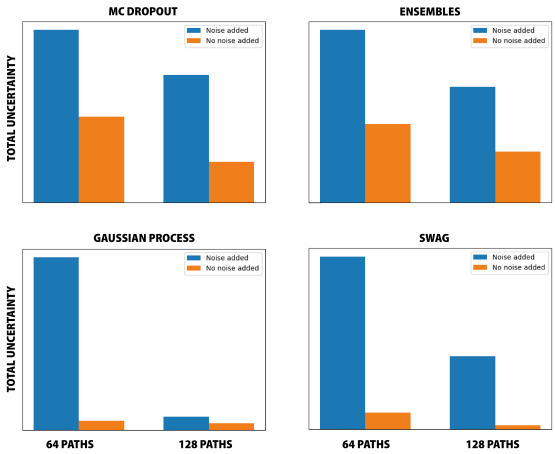

Figure 8: Simplistic bar charts depicting the total uncertainty captured by each model when using 64 and 128 paths with noise added into the states features X and sampled paths D and with no added noise. We can see epistemic part of the total uncertainty being reduced with more demonstrations D with the aleatoric part of the total uncertainty remaining. The total uncertainty is greater when noise is added into the states in every case.

7.3 Predictive Error Calibrated Uncertainty

For an uncertainty estimator to be perfectly calibrated with respect to its predictive error, we would expect the error of its predictions to equal its estimated variance (which is interpreted as the uncertainty prediction). In the ENCE case (eq 5) this would mean the RMV of every states reward prediction should match the RMSE. In the ENEE case (eq 6) this would mean the RMV of every reward prediction should match the EVD (eq 4).

A reliability diagram has been constructed to communicate the results for both metrics where a perfectly diagonal line would illustrate the RMV matching the prediction error at every state, hence a perfectly calibrated uncertainty approximator. The reliability diagram for each models ENCE scores (averaged across every experiment variant) can be seen in figure 9 and for the ENEE scores in figure 10. Figure 11 summarises all the averaged ENCE and ENEE scores, as well as the dispersion C_v (eq 7) of each models estimated uncertainty for every states reward prediction.

It's important to bear in mind that the results discussed in this section give direct insight into the accuracy of the calibrated uncertainty with respect to the model's predictive error, but not with respect to the to the noise introduced in the states features X and sampled paths D. Section 7.4

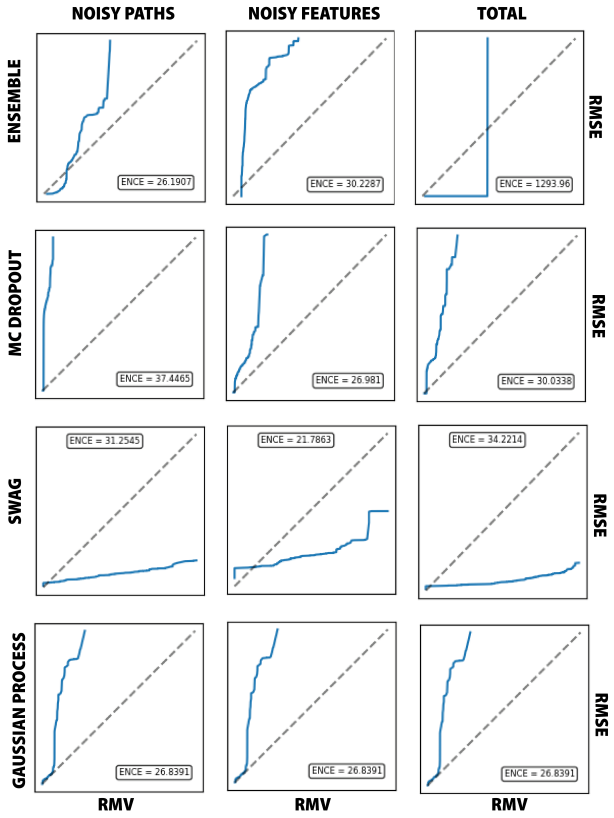

Figure 9: Reliability diagram for each models obtained ENCE averaged across every experiment. The RMV of the uncertainty predictions is plotted against RMSE. A perfectly diagonal line indicates RMV = RMSE for every state, hence a perfectly calibrated approximator.

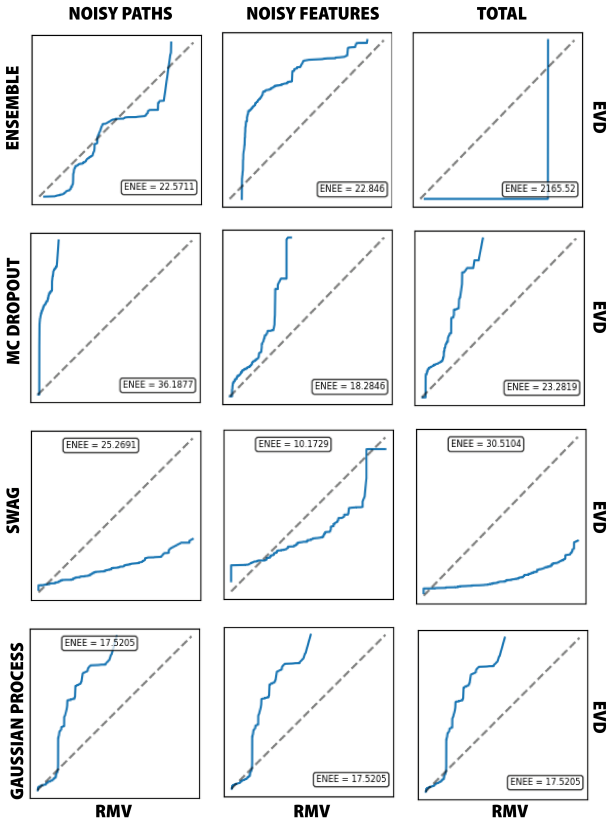

Figure 10: Reliability diagram for each models obtained ENEE averaged across every experiment. The RMV of the uncertainty predictions is plotted against EVD. A perfectly diagonal line indicates RMV = EVD for every state, hence a perfectly calibrated approximator.

	MC Dropout			Ensemble			SWAG			Gaussian Process		
	NP	NF	Total	NP	NF	Total	NP	NF	Total	NP	NF	Total
ENCE	37.45	26.98	30.03	26.19	30.23	1293.96	31.25	21.79	34.22	26.84	26.84	26.84
ENEE	36.19	18.28	23.28	22.57	22.85	2165.52	25.27	10.17	30.51	17.52	17.11	17.52
Cv	0.43	2.26	1.57	2.11	0.38	7400.29	1.16	0.82	0.65	1.86	1.86	1.86

Figure 11: Summary table showing each models obtained averaged ENEE, ENCE and Cv score for each type of experiment.

investigates the latter.

As expected due to their similar error based formulation, the ENEE and ENCE metrics obtain similar results. Both metrics account for the discrepancies between the error of models predicted r and uncertainty for every state, but ENCE uses RMSE as the error metric whereas ENCE uses EVD (eq 4). Given that EVD is an IRL specific error and more correctly captures the predictive policy error than RMSE, the ENEE results can be interpreted as a truer representation of the uncertainty calibration w.r.t to the predictive error than ENCE.

The reliability diagrams, 9 10, show that ensembles, MC dropout and GP severely under-estimate their predictive error whereas SWAG over-estimates. This underestimation of uncertainty has been exhibited in MC Dropout in numerous studies [20, 18, 1], and this research confirms this trend continues into the IRL regression case.

To speculate, this could be because deriving sub-networks from one main network through deactivating neurons doesn't introduce enough model diversity to accurately capture the uncertainty. The results show that ensembles was more accurately aware of its predictive error than MC Dropout in the noisy paths case, but performs dramatically worse in the other two cases, being completely unable to handle the volume of noise present when noise was simultaneously added into the state features X and the sampled paths D. This shows that Caruna's [6] ensemble selector technique as an uncertainty estimator does not scale well to large volumes of noisy input, which is intuitive since it was made to optimise performance, not uncertainty calibration. Alternatively, perhaps a randomization-based ensembling technique would scale better and more accurately capture the models predictive error at this level of noise since it would inherently explore more diverse models in function space. Caruna's ensemble technique does promote diversity through randomly initialising the parameters of the models used to construct the ensemble, but when selecting the ensemble from these trained models it favours the well performing networks which would reduce the ensemble diversity in since generally the best performance would come from similar models. Moreover, perhaps bootstrapping [30] the ensemble at training time would further increase this diversity and yield predictive error aware uncertainty estimates. [6]'s ensemble selector algorithm allows for completely different models with different loss functions to be selected from which would intuitively promote a lot variability in the ensemble. Future work could leverage this feature in order to improve the ac-

curacy of the uncertainty estimates obtained using the technique.

The GP achieves near identical results in all 3 cases over both metrics, slightly more accurately calibrating its uncertainty in the noisy features case. This could be because Levine's GPIRL implementation [22] did not propagate the input uncertainty through it's likelihood thus the model never tuned the uncertainty with respect its prediction error which would result in a similar estimated uncertainty regardless of the models error.

Conversely, SWAG is observed to over-estimate the uncertainty of a lower predictive error. Again, this finding is inline with the trend of previous studies which show SWAG produces more robust models than MC Dropout and ensembling [25]. The lower predictive error could be down to the fact that the final model is an average of the models in only the last 25% of training when the predictive error is at its lowest.

SWAG achieving the lowest averaged uncertainty dispersion (seen in table 11) indicates that the models used to create the the averaged model came from many similar, optimal, models. Interestingly, when considering the scores between ENCE and ENEE, every models calibrated uncertainty appears more accurate in their ENEE 10 scores. This is because EVD 5.1.1 is a more accurate metric than RMSE 5.2.1 in capturing the models predictive error. This highlights the magnitude of influence the metrics have on uncertainty calibration studies and brings to bear the need for research into better more accurate metrics to measure uncertainty calibration for regression problems.

Although the results from this study cannot definitely determine which method calibrates the most accurate uncertainty for IRL, it can be said that SWAG emerges as the most promising. Given the stochastic decision making nature of IRL, it is much more desirable and safer to overestimate uncertainty rather than underestimate. Uncertainty underestimation is equivalent to model over-confidence which could lead to fatal decisions given the safety critical applications IRL is currently used in.

7.4 Input Noise Calibrated Uncertainty

Figure 12 details the t-test results obtained by each model over each experiment variant when noise was added into 12.5%, 25% and 50% of the states. The details on the type of t test that were performed and justifications for using the significance level of 5% and critical value of 1.98 can be seen in section 5.3. A green box highlight the results that can be said to have a statistically significant larger average uncertainty estimation for the states with additional noise i.e the results where $t > 1.98$ & $p < 5\%$. The goal of this section is to give insight into if the estimated uncertainty from each model is calibrated with respect to the to the noise introduced in the states features X and sampled paths D

When noise was introduced to only 12.5% of states, SWAG on NF experiment is the only method that can be said to have a statistically significant larger average uncertainty estimation for the states with additional noise. This result is inline with SWAG's NF ENEE and ENCE score 11 which is consistently the lowest overall. If we loosened the significance level to 10% then the GP's uncertainty would also fall into this category.

When more the percent of states made noisy increases

% of states noise added to		MC Dropout			Ensemble			SWAG			Gaussian Process		
		NP	NF	Total	NP	NF	Total	NP	NF	Total	NP	NF	Total
12.5%	t	-2.40	1.87	0.26	-1.11	0.00	-3.06	1.31	2.58	2.83	1.94	1.94	1.94
	p	1.69%	6.56%	79.69%	27.25%	99.66%	0.34%	19.73%	1.39%	0.73%	6.19%	6.19%	6.19%
	Cv	3.22	2.89	2.94	0.48	1.54	0.36	0.69	0.70	0.76	7.73	7.73	7.73
25%	t	1.80	2.77	1.38	1.82	1.02	-2.62	1.82	1.02	5.30	1.82	1.02	4.26
	p	8%	1%	17%	7%	31%	1%	7%	31%	0%	7%	31%	0%
	Cv	3.68	2.89	3.32	3.26	1.12	0.28	3.26	1.12	0.98	3.26	1.12	3.86
50%	t	2.91	4.01	2.23	3.05	1.89	-3.66	3.05	1.89	9.37	3.05	1.89	4.19
	p	0%	0%	3%	0%	6%	0%	0%	6%	0%	0%	6%	0%
	Cv	1.97	2.89	2.36	3.32	1.12	9010.51	3.32	1.12	0.93	3.32	1.12	1.65

Figure 12: t-test t and p values for each model for experiment variant. Green boxes highlight the results that can be said to have a statistically significant larger average uncertainty estimation for the states with additional noise i.e the results where $t > 1.98$ & $p < 5\%$

from 12.5% to 25%, the overall uncertainty calibration with respect to the input noise also increases. This can be exhibited from the t-test results seen in figure 12 which indicate the number of methods that had statistically significant larger average uncertainty estimation for the states with additional noise increases from 1 to 3: MC Dropout NF; SWAG Total; GP Total.

This positive correlation between percent of states made noisy and uncertainty calibration with respect to the input noise continues when the percent states made noisy increases from 25% to 50% as figure 12 shows the number of methods that had statistically significant larger average uncertainty estimation for the states with additional noise increases from 3 to 8: MC Dropout NF, NP & Total; Ensembles NP; SWAG NP & Total; GP NP & Total.

From considering the results in sections 7.3 and 7.4 it can be seen that the uncertainty estimates were generally more accurately calibrated to the input noise than their predictive error. Speculatively, this could be because the total uncertainty is mainly comprised of heteroscedastic aleatoric uncertainty originating from the heterogeneous noise added introduced into the states features X and sampled paths D. Another explanation for this observation could be that the added noise didn't cause a greater error specific to the states with noise but just rendered a greater overall error.

The uncertainty became more calibrated to the states noise as a greater percent of states had noise added, exhibited through the t test results seen in figure 12. This indicates that larger *number* of states with added noise enabled the models to better identify discrepancies in their feature representation and the sampled paths which was exhibited through a greater variation of those states' predicted reward over the 5,000 predictions.This observation is not consistent with the amount of noise added per state as figure 12 shows every models uncertainty being less calibrated when both features and paths were made noisy (total) vs the individual cases (NF & NP). Although this could also be due to the nature of the IRL problem as a true state feature representation X would give the model a tool to detect noise in the paths through comparing what is expected and vice versa given a true set of sampled paths D. By introducing noise in both, the model would have less accurate tools to detect these discrepancies which could account for the less calibrated uncertainties in the "total" case.

8. CONCLUSIONS & FUTURE WORK

8.1 Conclusion

This work has argued that GP's are undesirable to solve

the IRL problem on large state spaces with complex reward structures due to their computational complexity of $O(n^3)$ coming from their inherent dependence on a kernel machine. Although their intrinsic ability to capture and represent accurate uncertainty of their predictions still renders them as feasible models for large-scale IRL as calibrated uncertainty estimations is essential for stochastic decision making problems like IRL in order to enable the autonomous system to mitigate the risk in its decision making.

With this in mind, this work was able to show that 1) DNN's can be built to solve the IRL problem with greater accuracy than Levine's GP 2 and 2) The DNN regularisation techniques Monte-Carlo Dropout, Ensembling and Stochastic Weight Averaging Gaussian can be used to capture reasonably calibrated uncertainty for the IRL problem which is representational of both the epistemic and aleatoric uncertainty. The results in section 7 show that this uncertainty appears to be more accurately calibrated to the input noise than their predictive error indicating that the total rendered uncertainty is mainly comprised of heteroscedastic aleatoric uncertainty originating from the heterogeneous noise added introduced into the states features X and sampled paths D.

This work has also provided a comparative evaluation giving insight into the quality of each method as an uncertainty estimator for IRL. From the results, SWAG emerges as the best approximator of uncertainty as it achieves the best scores across all metrics when considered for all experiments and is the only method in this study that overestimated the uncertainty with respect to its predictive error rather than underestimate it. Uncertainty underestimation is equivalent to model over-confidence which could lead to fatal decisions given the safety critical applications IRL is currently used in. A lot more research is required until a single method can definitely be said to be able to calibrate accurate uncertainty for IRL, but this work stands as a stepping stone by showing that it is possible to calibrate uncertainty using DNN in IRL whilst increasing performance accuracy. Finally, a product of this research is a novel Python framework enabling the training, tuning and evaluation of DL models on the IRL problem with respect to their performance and uncertainty calibration that has been built modularly to easily enable the integration of new benchmarks problems, models, objective functions.and metrics to evaluate.

8.2 Future Work

The primary goal for future work will be to address the limitations identified and discussed. With short term priority being focused around learning the epistemic and aleatoric uncertainty individually in order to enable a deeper evaluation. [11] suggest a method involving learning the total uncertainty and the aleatoric uncertainty, then subtracting the aleatoric from the total and the remainder can be seen as the epistemic uncertainty. In the context of IRL the total uncertainty could be learned using the same method outlined in this paper (section 4) and the aleatoric uncertainty can be estimated directly from the data by modifying the network to predict both the mean μ and variance σ^2 of the output. A modified Maximum Entropy IRL Log Likelihood function (eq 1), by performing maximum a posterior estimation (MAP), can then be used to tune the observation noise parameter σ^2 which will represent the learned heteroscedastic aleatoric uncertainty. The epistemic uncertainty could then be obtained by subtracting the aleatoric from the total uncertainty.

Whilst many metrics exist for evaluating uncertainty estimations for classification problems, very few exist for regression and none, apart from ENEE 6 proposed in this work, exist for IRL. Section 7 discussed the significance the metrics have on the evaluation outcome and as such this field of research would greatly benefit from a selection of agreed upon metrics to evaluate and compare the quality of uncertainty estimators in IRL, and regression problems in general. It seems intuitive that EVD 5.1.1 should be used as the error calibrating metric.

This study was also limited by compute power. If more was available it would be interesting to see how more accurately tuning each models hyper parameters and allowing them to train to a tighter convergence accuracy would affect the quality of uncertainty estimates. Especially for the SWAG cases since it emerged as the most promising method.

Moreover, it would be interesting and promising to evaluate the uncertainty predictions of variations of the current methods and entirely different methods. It can be argued as the state-space size increases bootstrapping could become more beneficial as each bootstrapped sample of the state's features becomes a better representation of the underlying distribution, thus worth exploring on IRL benchmarks with more states since model diversity is key to epistemic uncertainty approximation. The feature of Caruna's ensemble selector algorithm that allows for an ensemble to be built from different models with varying objective functions also seems an interesting avenue of future work as selecting from entirely different models would promote ensemble variability, thus could lead to more accurate epistemic uncertainty estimations. It would also be interesting to see if the findings of improved uncertainty calibration using deep ensembling [33] and concrete dropout [10] is consistent with IRL.

Finally, in order to make the case that any method can calibrate accurate uncertainty for real-world applications a real-world benchmark must be used. Given this, it would be interesting to evaluate DNN's calibrated uncertainty on larger, real-world IRL benchmark problems.

Acknowledgments.

I would like to thank my supervisor Dr Bjorn Jensen for his support and guidance throughout the past year. His suggestions and feedback were highly useful and critical to this projects fulfilment.

9. REFERENCES

[1] A. Ashukha, A. Lyzhov, D. Molchanov, and D. Vetrov. Pitfalls of in-domain uncertainty estimation and ensembling in deep learning. *arXiv preprint arXiv:2002.06470*, 2020.

[2] D. Barber, A. T. Cemgil, and S. Chiappa. *Bayesian time series models*. Cambridge University Press, 2011.

[3] Y. Bengio. *Learning deep architectures for AI*. Now Publishers Inc, 2009.

[4] Y. Bengio, P. Lamblin, D. Popovici, H. Larochelle, et al. Greedy layer-wise training of deep networks. *Advances in neural information processing systems*, 19:153, 2007.

[5] L. Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[6] R. Caruana, A. Niculescu-Mizil, G. Crew, and A. Ksikes. Ensemble selection from libraries of models.

In *Proceedings of the twenty-first international conference on Machine learning*, page 18, 2004.

[7] D. Dewey. Reinforcement learning and the reward engineering principle. In *2014 AAAI Spring Symposium Series*, 2014.

[8] Y. Gal and Z. Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*, pages 1050–1059, 2016.

[9] Y. Gal and Z. Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*, pages 1050–1059. PMLR, 2016.

[10] Y. Gal, J. Hron, and A. Kendall. Concrete dropout. *arXiv preprint arXiv:1705.07832*, 2017.

[11] E. Hüllermeier and W. Waegeman. Aleatoric and epistemic uncertainty in machine learning: An introduction to concepts and methods. *Machine Learning*, pages 1–50, 2021.

[12] P. Izmailov, D. Podoprikhin, T. Garipov, D. Vetrov, and A. G. Wilson. Averaging weights leads to wider optima and better generalization. *arXiv preprint arXiv:1803.05407*, 2018.

[13] M. Jin, A. Damianou, P. Abbeel, and C. Spanos. Inverse reinforcement learning via deep gaussian process. *arXiv preprint arXiv:1512.08065*, 2015.

[14] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[15] I. Kononenko. Bayesian neural networks. *Biological Cybernetics*, 61(5):361–370, 1989.

[16] Y. Kwon, J.-H. Won, B. J. Kim, and M. C. Paik. Uncertainty quantification using bayesian neural networks in classification: Application to ischemic stroke lesion segmentation. 2018.

[17] B. Lakshminarayanan, A. Pritzel, and C. Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. *arXiv preprint arXiv:1612.01474*, 2016.

[18] M.-H. Laves, S. Ihler, K.-P. Kortmann, and T. Ortmaier. Well-calibrated model uncertainty with temperature scaling for dropout variational inference. *arXiv preprint arXiv:1909.13550*, 2019.

[19] S. Lee, S. Purushwalkam, M. Cogswell, D. Crandall, and D. Batra. Why m heads are better than one: Training a diverse ensemble of deep networks. *arXiv preprint arXiv:1511.06314*, 2015.

[20] D. Levi, L. Gispan, N. Giladi, and E. Fetaya. Evaluating and calibrating uncertainty prediction in regression tasks. *arXiv preprint arXiv:1905.11659*, 2019.

[21] E. Levin, R. Pieraccini, and W. Eckert. Using markov decision process for learning dialogue strategies. In *Proceedings of the 1998 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP'98 (Cat. No. 98CH36181)*, volume 1, pages 201–204. IEEE, 1998.

[22] S. Levine, Z. Popovic, and V. Koltun. Nonlinear inverse reinforcement learning with gaussian processes. In *Advances in Neural Information Processing Systems*, pages 19–27, 2011.

[23] Y. Li and Y. Gal. Dropout inference in bayesian neural networks with alpha-divergences. In *International conference on machine learning*, pages 2052–2061. PMLR, 2017.

[24] I. Loshchilov and F. Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

[25] W. J. Maddox, P. Izmailov, T. Garipov, D. P. Vetrov, and A. G. Wilson. A simple baseline for bayesian uncertainty in deep learning. *Advances in Neural Information Processing Systems*, 32:13153–13164, 2019.

[26] V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Icml*, 2010.

[27] A. Y. Ng, S. J. Russell, et al. Algorithms for inverse reinforcement learning. In *Icml*, volume 1, page 2, 2000.

[28] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic differentiation in pytorch. 2017.

[29] J. Quiñonero-Candela and C. E. Rasmussen. A unifying view of sparse approximate gaussian process regression. *Journal of Machine Learning Research*, 6(Dec):1939–1959, 2005.

[30] Y. Raviv and N. Intrator. Bootstrapping with noise: An effective regularization technique. *Connection Science*, 8(3-4):355–372, 1996.

[31] J. Shore and R. Johnson. Axiomatic derivation of the principle of maximum entropy and the principle of minimum cross-entropy. *IEEE Transactions on information theory*, 26(1):26–37, 1980.

[32] J. Slingo and T. Palmer. Uncertainty in weather and climate prediction. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 369(1956):4751–4767, 2011.

[33] H.-I. Suk, S.-W. Lee, D. Shen, A. D. N. Initiative, et al. Deep ensemble learning of sparse regression models for brain disease diagnosis. *Medical image analysis*, 37:101–113, 2017.

[34] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[35] C. K. Williams and C. E. Rasmussen. *Gaussian processes for machine learning*, volume 2. MIT press Cambridge, MA, 2006.

[36] M. Wulfmeier, P. Ondruska, and I. Posner. Maximum entropy deep inverse reinforcement learning. *arXiv preprint arXiv:1507.04888*, 2015.

[37] G. Yang, T. Zhang, P. Kirichenko, J. Bai, A. G. Wilson, and C. De Sa. Swalp: Stochastic weight averaging in low precision training. In *International Conference on Machine Learning*, pages 7015–7024. PMLR, 2019.

[38] L. Yu, J. Song, and S. Ermon. Multi-agent adversarial inverse reinforcement learning. In *International Conference on Machine Learning*, pages 7194–7201. PMLR, 2019.

[39] Z.-H. Zhou, J. Wu, and W. Tang. Ensembling neural networks: many could be better than all. *Artificial intelligence*, 137(1-2):239–263, 2002.

[40] B. D. Ziebart, A. L. Maas, J. A. Bagnell, and A. K. Dey. Maximum entropy inverse reinforcement learning. In *Aaai*, volume 8, pages 1433–1438, 2008.