Chapter 5

Applications

In previous chapters we linked stochastic regularisation techniques (SRTs) to approximate inference in Bayesian neural networks (NNs), and studied the resulting model uncertainty for popular SRTs such as dropout. We have yet to give any real-world *applications* stemming from this link though, leaving it somewhat in the realms of "theoretical work". But a theory is worth very little if we can't use it to obtain new tools, or shed light on existing ones. In this chapter we will survey recent literature making use of the tools developed in the previous chapters in fields ranging from language processing to computer vision to biomedical domains. This is followed by more use cases I have worked on recently with the help of others. We will see how model uncertainty can be used to choose what data to learn from (joint work with Riashat Islam as part of his Master's project). Switching to applications in deep reinforcement learning, we will see how model uncertainty can help exploration. This is then followed by the development of a data efficient framework in deep reinforcement learning (joint work with Rowan McAllister and Carl Rasmussen). A study of the implications of this work on our *understanding* of existing tools will be given in the next chapter.

5.1 Recent literature

We begin with a quick literature survey of applications making use of the tools brought above (and presented previously in [Gal, 2015; Gal and Ghahramani, 2015a,b,c,d, 2016a,b,c]). This survey offers support to our main claim of the tools developed being *practical*, and can be used as a starting point for further applications on related tasks. The key contributions of each work will be given, and research reproducing our results in previous chapters will be highlighted. We discuss research in three application areas: language, biology and medicine, and computer vision.

5.1.1 Language applications

Working on machine translation as part of the WMT16 shared task, the Edinburgh team [Sennrich et al., 2016] has made use of deep network architectures and specifically recurrent NNs. Machine translation is a challenging task where we wish to build a system to translate from one language to another, given only sentences in one language and their translations in the other. Sennrich et al. [2016] found that using naive dropout in RNNs their model would overfit, and reported that with the Bayesian variant of dropout proposed in §3.4.2 the model did not. Sennrich et al. [2016]'s model achieved state-of-the-art results on Czech–English, Romanian–English, and German–English language pairs, beating the other 31 submissions in the shared task [Bojar et al., 2016].

It is worth mentioning the work of Press and Wolf [2016] as well. Press and Wolf [2016], while working on language modelling, have reproduced our language modelling results presented in §3.4.2. Our language modelling results were further reproduced in the work of Léonard et al. [2015]—an open source Torch library for RNNs. Our dropout variant for RNNs is included in this library, as well as in the TensorFlow and Keras libraries [Abadi et al., 2015; Chollet, 2015] as the default RNN dropout implementation.

5.1.2 Medical diagnostics and bioinformatics

In medical diagnostics Yang et al. [2016] have worked on image registration. This task involves "stitching" together images obtained from brain scans for example. This is a challenging task as images might be taken at different points in time, and a patient's movement due to breathing for example could mean that the images would not be aligned. As a result there is an inherent uncertainty in the reconstruction process. Yang et al. [2016], relying on recent CNN tools, proposed a model that improved on existing techniques. Relying on our tools developed in section §3.4.1, Yang et al. [2016] showed how model uncertainty can be obtained in their setting, and empirically evaluated their model. For example, stitching MRI brain scans, they showed how shape changes in the anterior edge of the ventricle and the posterior brain cortex (changes due to blood flow at different points in time for example) led to high uncertainty in those regions.

As another example, Angermueller and Stegle [2015] attempted to predict methylation rate in embryonic stem cells. In DNA methylation, methyl groups are added to nucleotides and affect the transcription of genes. This process controls gene regulation, and affects the development of disease for example. Angermueller and Stegle [2015] fitted a neural network to their data, and showed an improvement over standard techniques in the field.

Using the techniques developed in §3.3 they showed that model uncertainty increased in genomic contexts which are hard to predict (e.g. LMR or H3K27me3).

5.1.3 Computer vision and autonomous driving

Numerous applications of the methods involving model uncertainty for image data (introduced in §3.3–§3.4) have been developed in recent literature [Bulò et al., 2016; Kendall and Cipolla, 2016], with specific attention paid to model uncertainty in image segmentation [Furnari et al., 2016; Kampffmeyer et al., 2016; Kendall et al., 2015]. This might be because model uncertainty is hard to obtain for image data, and with the tools developed in §3.4.1 obtaining this information is easier than with previous tools. In this section three main applications will be surveyed: new tools [Bulò et al., 2016], computer vision systems for autonomous driving [Kendall and Cipolla, 2016; Kendall et al., 2015], and image segmentation [Furnari et al., 2016; Kampffmeyer et al., 2016; Kendall et al., 2015].

Bulò et al. [2016] developed tools to efficiently approximate MC dropout. Even though MC dropout results in lower test RMSE, it comes with a price of prolonged test time. This is because we need to evaluate the network stochastically multiple times and average the results. Instead, Bulò et al. [2016] fit a secondary model to predict (with a single forward pass) the MC output of the primary model. Bulò et al. [2016] further reproduced our experimental results in §3.4.1 with the NiN model on CIFAR10. However compared to our results with lenet-all (applying MC dropout after every convolution layer on MNIST) where we observed a big improvement for MC dropout over standard dropout, Bulò et al. [2016]'s results suggested similar performance for both techniques.

Working on autonomous driving applications, Kendall and Cipolla [2016] developed tools for the localisation of a car given a photo taken from a front-facing camera installed in the vehicle. This is an important application since GPS systems cannot always be trusted and give low-accuracy localisation. When used in autonomous driving, this localisation information can be used to get information such as the distance of the car from the sidewalk for example. Kendall and Cipolla [2016] assessed model uncertainty following the tools developed in the previous chapters, and found model uncertainty to correlate to positional error. They found that test photos with strong occlusion from vehicles, pedestrians, or other objects resulted in high uncertainty, with model uncertainty showing a linear trend increasing with the distance from the training set (after calibration). In further work Kendall et al. [2015] developed models for scene understanding, mapping objects in the photos taken by the car to labels such as "pedestrian" or "cyclist" on

a pixel level. Extracting model uncertainty they showed that object edges have lower model confidence for example.

One last application is image segmentation. Kampffmeyer et al. [2016] for example looked at semantic segmentation in urban remote sensing images. They used CNNs with the techniques described in §3.4.1 and analysed the resulting segmentation uncertainty. Kampffmeyer et al. [2016] computed the standard deviation over the softmax outputs of 10 Monte Carlo samples. They then averaged the standard deviation over all classes to obtain a single scalar value[1]. They found that model uncertainty increased at object boundaries and in regions where the model misclassified, validating the hypothesis that pixels with high model confidence are classified correctly more often. Kampffmeyer et al. [2016] further gave precision-recall plots showing that when pixels with higher uncertainty were removed, classification accuracy increased. They concluded that uncertainty maps are a good measure for the pixel-wise uncertainty of the segmented remote sensing images.

I next give a more in-depth review of several applications developed in collaboration with others.

5.2 Active learning with image data

This has been joint work with Riashat Islam as part of his Master's project.

A big challenge in many applications is obtaining labelled data. This can be a long and laborious process, which often makes the development of an automated system uneconomical. A framework where a system could learn from small amounts of data, and choose by itself what data it would like the user to label, would make machine learning applicable to a wider class of problems. Such a framework is referred to as *active learning* [Cohn et al., 1996]. In this setting a model is trained on a small amount of data (the initial training set), and an *acquisition function* (often based on the model's uncertainty) decides what data points to ask an external *oracle* for a label. The acquisition function selects several points from a *pool* of data points, with the pool points lying outside of the training set. An oracle (often a human expert) labels the selected data points, these are added to the training set, and a new model is trained on the updated training set. This process is then repeated, with the training set increasing in size over time.

Even though active learning forms an important pillar of machine learning, deep learning tools are not prevalent within it. Deep learning poses several difficulties when

[1] Note that this is not necessarily the best uncertainty summary for classification, and the techniques discussed in §3.3.1 might give more sensible results. We will get back to this problem in the next section.

used in an active learning setting. First, we have to handle small amounts of data. Second, many acquisition functions rely on model uncertainty. Luckily Bayesian approaches to deep learning make this task more practical. Even more exciting, taking advantage of specialised models such as Bayesian convolutional neural networks, we can perform active learning with *image data*. Here we will develop an active learning framework for image data, a task which has been extremely challenging so far with very sparse existing literature [Holub et al., 2008; Joshi et al., 2009; Li and Guo, 2013; Zhu et al., 2003].

To perform active learning with image data we make use of the Bayesian CNNs developed in section §3.4.1. These work well with small amounts of data (as seen in section §4.4.1), and possess uncertainty information that can be used with existing acquisition functions. With regression, acquisition functions can be formed by looking at the sample variance. But CNNs are often used in the context of classification, and in this setting we shall make use of the uncertainty measures discussed in §3.3.1. More specifically we will analyse acquisition functions based on a random acquisition (referred to as *Random*), functions maximising the predictive entropy (referred to as *Max Entropy*, [Shannon, 1948]), maximising the *variation ratios* [Freeman, 1965], and maximising the mutual information between the predictions and the model posterior (referred to as *BALD*, "Bayesian active learning by disagreement", [Houlsby et al., 2011]).

Bayesian CNN approximate inference can be done using various approximating distributions. We experiment with two approximating distributions: a product of Bernoullis (implemented as dropout before each weight layer, referred to as *Dropout*), and a delta approximating distribution (implemented as a standard CNN with a deterministic softmax probability vector, referred to as *Softmax*). All acquisition functions are assessed with the same model structure: convolution-relu-convolution-relu-max pooling-dropout-dense-relu-dropout-dense-softmax, with 32 convolution kernels, 4x4 kernel size, 2x2 pooling, dense layer with 128 units, and dropout probabilities 0.25 and 0.5. All models are trained on the MNIST dataset [LeCun and Cortes, 1998] with a (random but balanced) initial training set of 20 data points, a validation set of 5K points, the standard test set of 10K points, and the rest of the points used as a pool set. All models were assessed after each acquisition using the dropout approximation, with the Softmax models using the dropout approximation to evaluate model output for acquisition, and the Dropout models using MC dropout to evaluate model output for acquisition. We repeated the acquisition process 100 times, each time acquiring the 10 points that maximised the

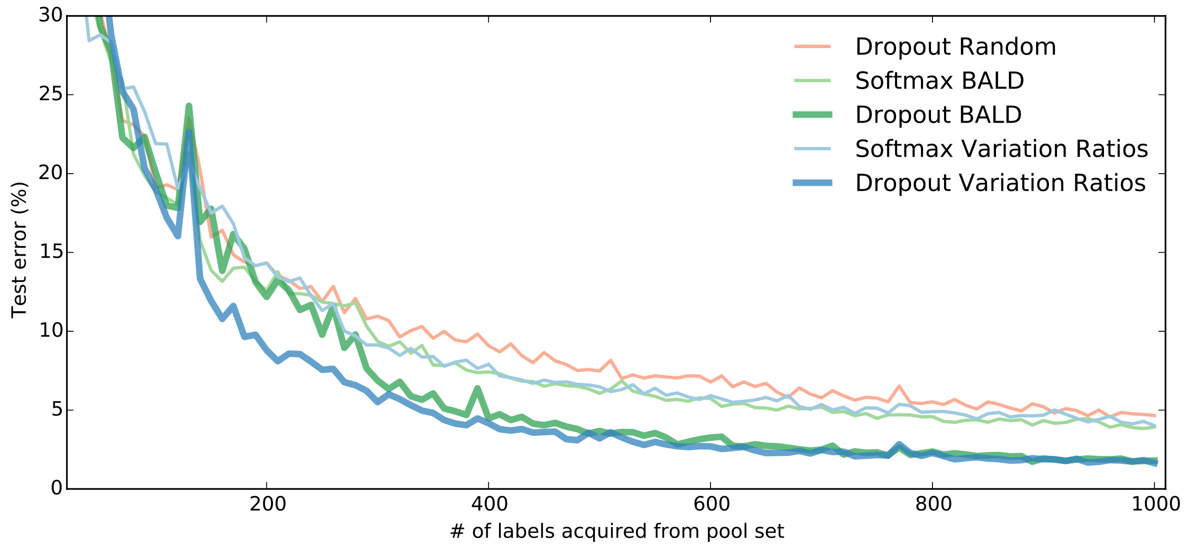


Fig. 5.1 **Test error on MNIST as a function of number of labels acquired from the pool set.** Two acquisition functions (*BALD* and *Variation Ratios*) evaluated with two approximating distributions—delta (*Softmax*) and Bernoulli (*Dropout*)—are compared to a *random* acquisition function.

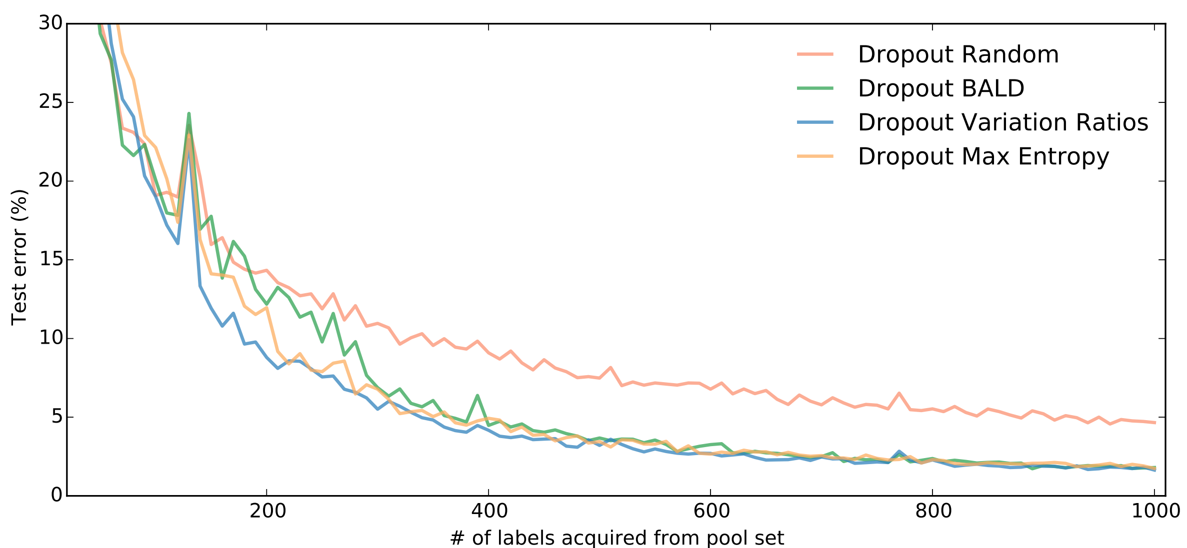


Fig. 5.2 **Test error on MNIST as a function of number of labels acquired from the pool set.** Four acquisition functions are shown (*Random*, *BALD*, *Variation Ratios*, and *Max Entropy*) evaluated with a Bernoulli approximating distribution (*Dropout*).

	Random	Max Entropy	Variation Ratios	BALD
Softmax	4.57	2.18	4.02	3.83
Dropout	NA	1.74	1.64	1.72

Table 5.1 **Model error (%)** on MNIST with 1000 training points for different acquisition functions.

acquisition function over the pool set. Each experiment was repeated 3 times and the results averaged[2].

A comparison of the Dropout acquisition functions to the Softmax acquisition functions for Variation Ratios and BALD is given in fig. 5.1. Both acquisition functions (*BALD* and *Variation Ratios*) were evaluated with both approximating distributions and compared to a random acquisition function where new points are sampled uniformly at random from within the pool set. The Dropout acquisition functions, propagating uncertainty throughout the model, attain a smaller error early on, and converge to a lower error rate overall. This demonstrates that the uncertainty propagated throughout the Bayesian models has a significant effect on the acquisition functions' measure of model confidence. Variation Ratios seems to obtain lower test error faster than BALD for acquisition points ranging between 150 and 400, but BALD outperforms Variation Ratios on the range 0 to 150 (not shown).

	Random	Max Entropy	Variation Ratios	BALD
Softmax	90	53	76	68
Dropout	NA	35	32	36

Table 5.2 **Number of acquisition steps** to get to model error of 5% on MNIST, for different acquisition functions.

We further compared the acquisition functions Random, Variation Ratios, and BALD to an acquisition function based on the predictive entropy (Max Entropy). We found Random to under-perform compared to BALD, Variation Ratios, and Max Entropy (figure 5.2). The converged test error for all acquisition functions after the acquisition of 1000 training points is given in table 5.1.

Lastly, in table 5.2 we give the number of acquisition steps needed to get to test error of 5%. As can be seen, Dropout Max Entropy, Dropout Variation Ratios and Dropout

[2]The code for these experiments is available at `https://github.com/Riashat/Active-Learning-Bayesian-Convolutional-Neural-Networks/tree/master/ConvNets/FINAL_Averaged_Experiments/Final_Experiments_Run`.

BALD attain test error of 5% within a much smaller number of acquisitions than their Softmax equivalents.

We next switch to the use of model uncertainty in the setting of reinforcement learning, on a task similar to that used in [Mnih et al., 2015].

5.3 Exploration in deep reinforcement learning

In reinforcement learning an agent receives various rewards from different states, and its aim is to maximise its expected reward over time. The agent tries to learn to avoid transitioning into states with low rewards, and to pick actions that lead to better states instead. Uncertainty is of great importance in this task—with uncertainty information an agent can decide when to exploit rewards it knows of, and when to explore its environment.

Recent advances in RL have made use of NNs to estimate agents' Q-value functions (referred to as Q-networks), a function that estimates the quality of different actions an agent can take at different states. This has led to impressive results on Atari game simulations, where agents superseded human performance on a variety of games [Mnih et al., 2015]. Epsilon greedy search was used in this setting, where the agent selects the best action following its current Q-function estimation with some probability, and explores otherwise. With our uncertainty estimates given by a dropout Q-network we can use techniques such as Thompson sampling [Thompson, 1933] to converge faster than with epsilon greedy while avoiding over-fitting.

We use code by Karpathy et al. [2014–2015] that replicated the results by Mnih et al. [2015] with a simpler 2D setting. We simulate an agent in a 2D world with 9 eyes pointing in different angles ahead (depicted in fig. 5.3). Each eye can sense a single pixel intensity of 3 colours. The agent navigates by using one of 5 actions controlling two motors at its base. An action turns the motors at different angles and different speeds. The environment consists of red circles which give the agent a positive reward for reaching, and green circles which result in a negative reward. The agent is further rewarded for not looking at (white) walls, and for walking in a straight line[3].

We trained the original model, and an additional model with dropout with probability 0.1 applied before every weight layer. Note that both agents use the same network structure in this experiment for comparison purposes. In a real world scenario using dropout we would use a larger model (as the original model was intentially selected to

[3]The code for this experiment is given at
https://github.com/yaringal/DropoutUncertaintyDemos.

be small to avoid over-fitting). To make use of the dropout Q-network's uncertainty estimates, we use Thompson sampling instead of epsilon greedy. In effect this means that we perform a single stochastic forward pass through the network every time we need to take an action. In replay, we perform a single stochastic forward pass and then back-propagate with the sampled Bernoulli random variables.

In fig. 5.4 we show a plot of the average reward as a function of learning time (on a log scale) obtained by both the original implementation (in green) and our approach (in blue). Not plotted is the burn-in intervals of 25 batches (random moves). Thompson sampling gets reward larger than 1 within 25 batches from burn-in. Epsilon greedy takes 175 batches to achieve the same performance[4].

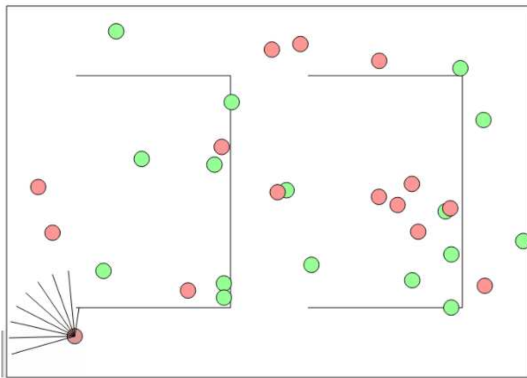


Fig. 5.3 Depiction of the reinforcement learning problem used in the experiments. The agent is in the lower left part of the maze, facing north-west.

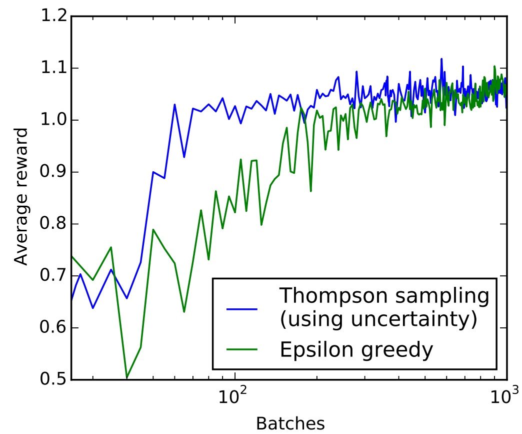


Fig. 5.4 **Log plot of average reward** obtained by both epsilon greedy (in green) and our approach (in blue), as a function of the number of batches.

Remark. We presented these results initially in [Gal and Ghahramani, 2015c]. Stadie et al. [2015] have reproduced our experiments while comparing to their model-based exploration technique (using an auto-encoder to encode the states). They compared to our model-free Thompson sampling, the epsilon greedy based DQN [Mnih et al., 2015], as well as to Boltzmann exploration (which samples an action from the softmax output, annealing the probabilities to smoothly transition from

[4] It is interesting to note that our approach seems to stop improving after 1K batches. This is because we are still sampling random moves, whereas epsilon greedy only exploits at this stage.

exploration to exploitation: $a \sim \text{Categorical}(\exp(Q(s,a)/\tau)/(\sum_a \exp(Q(s,a)/\tau))$ with $\tau \to 0$).
Stadie et al. [2015] experimented with the Atari framework, in which they chose several games that existing methods found challenging, and where human performance superseded that of DQN. They found that after 100 epochs their model-based approach outperformed existing model-free techniques. Comparing DQN to our Thompson technique, they found DQN to get higher scores than Thompson on 6 games, whereas Thompson got higher scores than DQN on 7 games. Looking at the area under the curve (AUC) for the first 100 epochs—a measure of how quickly an agent learns—Thompson beat DQN on 8 games, whereas DQN beat Thompson on 5 games, emphasising the data efficiency of Thompson sampling compared to epsilon greedy. Thompson achieved best score out of all approaches on one of the games. It is interesting that Boltzmann exploration outperformed Thompson sampling quite considerably: on 4 games Thompson got better score than Boltzmann, whereas Boltzmann got better results on 8 games. Assessing AUC, Thompson beat Boltzmann on 3 games, with Boltzmann beating Thompson on 9 games. This might be because Thompson sampling still explores after 100 epochs, whereas Boltzmann exploration exploits by then. It would be interesting to combine the two techniques, and sample from an annealed probability obtained from a Bayesian NN.

5.4 Data efficiency in deep reinforcement learning

This has been joint work with Rowan McAllister and Carl Rasmussen [Gal, McAllister, and Rasmussen, 2016].

Compared to the previous model-free approach, *PILCO* is a *model-based* policy search RL algorithm [Deisenroth and Rasmussen, 2011]. PILCO achieved unprecedented data-efficiency of several control benchmarks including the cartpole swing-up task. But PILCO is limited by its use of a Gaussian process (GP) to model its distribution over model dynamics. GPs are hard to scale to large quantities of data and high dimensional data [Gal et al., 2014]. Instead, we might want to replace the GP with a Bayesian NN. But this task poses several interesting difficulties. First, we have to handle *small data*, and neural networks are notoriously known for their tendency to overfit. Furthermore, we must retain PILCO's ability to capture 1) dynamics model output uncertainty and 2)

input uncertainty. Output uncertainty can be captured with a Bayesian neural network (BNN), but end-to-end inference poses a challenge. Input uncertainty in PILCO is obtained by analytically propagating a state distribution through the dynamics model. But this can neither be done analytically with NNs nor with BNNs. Our solution to handling output uncertainty relies on dropout as a Bayesian approximation to the dynamics model posterior. This allows us to use techniques proven to work well in the field, while following their probabilistic Bayesian interpretation. Input uncertainty in the dynamics model is captured using particle techniques. To do this we solve the difficulties encountered by McHutchon [2014] when attempting this particle technique with PILCO in the past. Interestingly, unlike PILCO, our approach allows us to sample dynamics functions, required for accurate variance estimates of future state distributions.

Our approach has several benefits compared to the existing PILCO framework. First, as we require lower time complexity (linear in trials and observation space dimensionality), we can scale our algorithm well to tasks that necessitate more trials for learning. Second, unlike PILCO we can sample dynamics function realisations. The use of a NN dynamics model comes at a price though, where we need to use a slightly higher number of trials than PILCO. Lastly, our model can be seen as a Bayesian approach to performing data efficient deep RL. In section §5.4.3 we compare our approach to that of recent deep RL algorithms [Gu et al., 2016; Lillicrap et al., 2015], showing orders of magnitude improvement in data efficiency on these.

5.4.1 PILCO

PILCO is summarised by Algorithm 3. A policy π's functional form is chosen by the user (step 1), whose parameters ψ are initialised randomly (step 2). Thereafter

Algorithm 3 PILCO

1: *Define* policy's functional form: $\pi : z_t \times \psi \to u_t$.
2: *Initialise* policy parameters ψ randomly.
3: **repeat**
4:     *Execute* system, record data.
5:     *Learn* dynamics model.
6:     *Predict* system trajectories from $p(X_0)$ to $p(X_T)$.
7:     *Evaluate* policy:
$$J(\psi) = \sum_{t=0}^{T} \gamma^t \mathbb{E}_X[\text{cost}(X_t)|\psi].$$
8:     *Optimise* policy:
$$\psi \leftarrow \arg\min_{\psi} J(\psi).$$
9: **until** policy parameters ψ converge

PILCO executes the current policy from an initial state (sampled from initial distribution $p(X_0)$) until the time horizon T (defined as one trial, step 4). Observed transitions are recorded, and appended to the total training data. Given the additional training data, the dynamics model is re-trained (step 5). Using its probabilistic transition model, PILCO then analytically predicts state distributions from an initial state distribution $p(X_0)$ to $p(X_1)$ etc. until time horizon $p(X_T)$ making a joint Gaussian assumption (step 6). Prediction of future state distribution follows the generative model seen in Figure 5.5, where each system-state X_t defines an action U_t according to policy π, which determines the new state X_{t+1} according to the dynamics f. I.e. $X_{t+1} = f(X_t, U_t)$, where we train a GP model of f given all previous observed transition tuples $\{X_t, U_t, X_{t+1}\}$. Given the multi-state distribution $p(\{X_0, ..., X_T\})$, the expected cost $\mathbb{E}_X[\text{cost}(X_t)]$ is computed for each state distribution using a user-supplied cost function. The sum of expected costs is our minimisation objective J (step 7). Gradient information is also computed w.r.t. policy parameters $\mathrm{d}J/\mathrm{d}\psi$. Finally, the objective J is optimised using gradient decent according to $\mathrm{d}J/\mathrm{d}\psi$ (step 8). The algorithm then loops back to step 4 and executes the newly-optimised policy, which is locally optimal given all the data observed thus far.

PILCO's data-efficiency success can be attributed to its *probabilistic* dynamics model. Probabilistic models help avoid model bias—a problem that arises from selecting only a single dynamics model $\hat{f}$ from a large possible set and assuming that $\hat{f}$ is the correct model with certainty [Deisenroth et al., 2015]. Whilst such approaches can provide accurate short term state predictions e.g. $p(X_1)$, their longterm predictions (e.g. $p(X_T)$) are inaccurate due to the compounding effects of T-many prediction errors from $\hat{f}$. Since inaccurate predictions of $p(X_T)$ are made with high-confidence, changes in policy parameters ψ are (falsely) predicted to have significant effect on the expected cost at time T. Since optimising total expected cost J must balance the expected costs of states, including $p(X_1)$ and $p(X_T)$, the optimisation will compromise on the cost of $p(X_1)$ based on perceived cost of $p(X_T)$—even though the prediction $p(X_T)$ is effectively random noise with $p(X_T)$ having a broad distribution almost invariant to policy π. Such undesirable behaviour hampers data efficiency. Optimising data-efficiency exacerbates the negative effects of model bias even further, since the smaller the data, the larger the set of plausible models that can describe that data. PILCO uses probabilistic models to avoid model bias by considering *all* plausible dynamics models in prediction of all future states. In cases as the above, PILCO optimises the policy based only on the states X_t it can predict well.

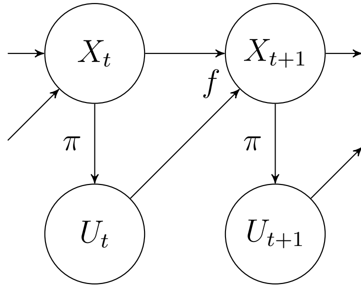


Fig. 5.5 **Prediction model of system trajectories** (step 6 in Algorithm 3). The system state X_t generates action U_t according to policy π, both of which result in a new state X_{t+1} as predicted by dynamics model f (a model trained given all previously observed X and U).

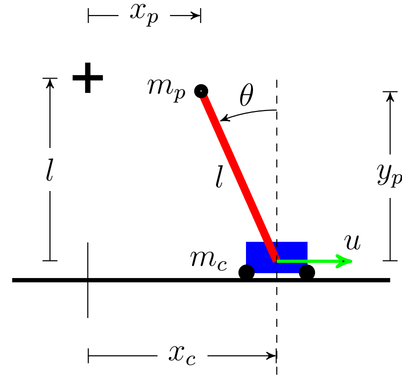


Fig. 5.6 **The cartpole swing-up task.** A pendulum of length l is attached to a cart by a frictionless pivot. The system begins with cart at position $x_c = 0$ and pendulum hanging down: $\theta = \pi$. The goal is to accelerate the cart by applying horizontal force u_t at each timestep t to invert then stabilise the pendulum's endpoint at the goal (black cross).

5.4.2 Deep PILCO

We now describe our method—*Deep PILCO*—for data-efficient deep RL. Our method is similar to PILCO: both methods follow Algorithm 3. The main difference of Deep PILCO is its dynamics model. PILCO uses a Gaussian process which can model the dynamics' output uncertainty, but cannot scale to high dimensional observation spaces. In contrast, Deep PILCO uses a deep neural network capable of scaling to high dimensional observations spaces. Like PILCO, our policy-search algorithm alternates between fitting a dynamics model to observed transitions data, evaluating the policy using dynamics model predictions of future states and costs, and then improving the policy.

Replacing PILCO's GP with a deep network is a surprisingly complicated endeavour though, as we wish our dynamics model to maintain its probabilistic nature, capturing 1) output uncertainty, and 2) input uncertainty.

Output uncertainty

First, we require output uncertainty from our dynamics model, critical to PILCO's data-efficiency. Simple NN models cannot express output model uncertainty, and thus cannot capture our ignorance of the latent system dynamics. To solve this we use the developments presented in §3.4.2. We interpret dropout as a variational Bayesian

approximation, and use the uncertainty in the weights to induce prediction uncertainty. This amounts to the regular dropout procedure only with dropout also applied at test time, giving us output uncertainty from our dynamics model.

This approach also offers insights into the use of NNs with small data. Eq. (3.14) for example shows that the network's weight decay can be parametrised as a function of dataset size, dropout probability, and observation noise. Together with adaptive learning-rate optimisation techniques, the number of parameters requiring tuning becomes negligible.

Input uncertainty

A second difficulty with NN dynamics models is handling *input* uncertainty. To plan under dynamics uncertainty, PILCO analytically propagates state distributions through the dynamics model (step 6 in Algorithm 3, depicted in Figure 5.5). To do so, the dynamics model must pass uncertain dynamics outputs from a given time step as uncertain input into the dynamics model in the next time step. This handling of input uncertainty cannot be done analytically with NNs, as is done with Gaussian processes in PILCO.

To feed a distribution into the dynamics model, we resort to particle methods (Algorithm 4). This involves sampling a set of particles from the input distribution (step 2 in Algorithm 4), passing these particles through the BNN dynamics model (step 8 in Algorithm 4), which yields an output distribution of particles.

Algorithm 4 Step 6 of Algorithm 3: *Predict* system trajectories from $p(X_0)$ to $p(X_T)$

1: *Define* time horizon T.
2: *Initialise* set of K particles $x_0^k \sim P(X_0)$.
3: **for** $k = 1$ to K **do**
4: Sample BNN dynamics model weights W^k.
5: **end for**
6: **for** time $t = 1$ to T **do**
7: **for** each particle x_t^1 to x_t^K **do**
8: Evaluate BNN with weights W^k and input particle x_t^k, obtain output y_t^k.
9: **end for**
10: Calculate mean μ_t and standard deviation σ_t^2 of $\{y_t^1, ..., y_t^K\}$.
11: Sample set of K particles $x_{t+1}^k \sim \mathcal{N}(\mu_t, \sigma_t^2)$.
12: **end for**

This approach was attempted unsuccessfully in the past with PILCO [McHutchon, 2014]. McHutchon [2014] encountered several problems optimising the policy with particle methods, the main problem being the abundance of local optima in the optimisation surface, impeding his BFGS optimisation method. McHutchon [2014] suggested that this

might be due to the finite number of particles used and their deterministic optimisation. To avoid these issues, we randomly re-sample a new set of particles at each optimisation step, giving us an unbiased estimator for the objective (step 7 in Algorithm 3). We then use the stochastic optimisation procedure Adam [Kingma and Ba, 2014] instead of BFGS.

We found that fitting a Gaussian distribution to the output state distribution at each time step, as PILCO does, is of crucial importance (steps 10-11 in Algorithm 4). This moment matching avoids multi-modality in the dynamics model. Fitting a multi-modal distribution with a (wide) Gaussian causes the objective to average over the many high-cost states the Gaussian spans [Deisenroth et al., 2015]. By forcing a unimodal fit, the algorithm penalises policies that cause the predictive states to bifurcate, often a precursor to a loss of control. This can alternatively be seen as smoothing the gradients of the expected cost when bifurcation happens, simplifying controller optimisation. We hypothesised this to be an important modelling choice done in PILCO.

Sampling functions from the dynamics model

Unlike PILCO, our approach allows sampling individual functions from the dynamics model and following a single function throughout an entire trial. This is because a repeated application of the BNN dynamics model above can be seen as a simple Bayesian recurrent neural network (RNN, where an input is only given at the first time step). Approximate inference in the Bayesian RNN is done by sampling function weights once for the dynamics model, and using the same weights at all time steps (steps 4 and 8 in Algorithm 4). With dropout, this is done by sampling and fixing the dropout mask for all time steps during the rollout (§3.4.2). PILCO does not consider such temporal correlation in model uncertainty between successive state transitions, which results in PILCO underestimating state uncertainty at future time steps [Deisenroth et al., 2015].

Another consequence of viewing our dynamics model as a Bayesian RNN is that the model could be easily extended to more interesting RNNs such as Bayesian LSTMs, capturing long-term dependencies between states. This is important for non-Markovian system dynamics, which can arise with observation noise for example. Here we restrict the model to Markovian system dynamics, where a simple Bayesian recurrent neural network model suffices to predict a single output state given a single input state.

5.4.3 Experiment

To compare our technique to PILCO we experimented with the cartpole swing-up task—a standard benchmark for nonlinear control. Lillicrap et al. [2015] and Gu et al. [2016]

use this benchmark as well, assessing their approach on the four-dimensional state-space task.

Figure 5.7 shows the average cost of 40 random runs (with two standard errors denoted by two shades for each plot). Deep PILCO successfully balances the pendulum in most experiment repetitions within 20-25 trials, compared to PILCO's 6-7 trials. Figure 5.8 shows the progression of deep PILCO's fitting as more data is collected.

Our model can be seen as a Bayesian approach to data efficient deep RL. We compare to recent deep RL algorithms (Lillicrap et al. [2015] and Gu et al. [2016]). Lillicrap et al. [2015] use an actor-critic model-free algorithm based on deterministic policy gradients. Gu et al. [2016] train a continuous version of model-free deep Q-learning using imagined trials generated with a learnt model. For their *low dimensional* cartpole swing-up task Lillicrap et al. [2015] require approximately 2.5×10^5 steps to achieve good results. This is equivalent to approximately 2.5×10^3 trials of data, based on Figure 2 in [Lillicrap et al., 2015] (note that [Lillicrap et al., 2015] used time horizon $T = 2$s and time discretisation $\Delta t = 0.02$s, slightly different from ours; they also normalised their reward, which does not allow us to compare to their converged reward directly). Gu et al. [2016] require approximately 400 trials for model convergence. These two results are denoted with vertical lines in Figure 5.7 (as the respective papers do not provide high resolution trial-cost plots).

Lastly, we report model run time for both Deep PILCO as well as PILCO. Deep PILCO can leverage GPU architecture, and took 5.85 hours to run for the first 40 iterations. This is with constant time complexity w.r.t. the number of trials, and linear time complexity in input dimensionality Q and output dimensionality D. PILCO (running on the CPU) took 20.7 hours for the 40 trials, and scales with $\mathcal{O}(N^2Q^2D^2)$ time complexity, with N number of trials. With more trials PILCO will become much slower to run. Consequently, PILCO is unsuited for tasks requiring a large number of trials or high-dimensional state tasks.

We surveyed research in three application areas: language, biology and medicine, and computer vision, and gave example applications in RL and active learning, lending support to our main claim of the tools developed in chapter 3 being *practical*. We next go in depth into some more theoretical questions analysing the developments of the previous chapters.

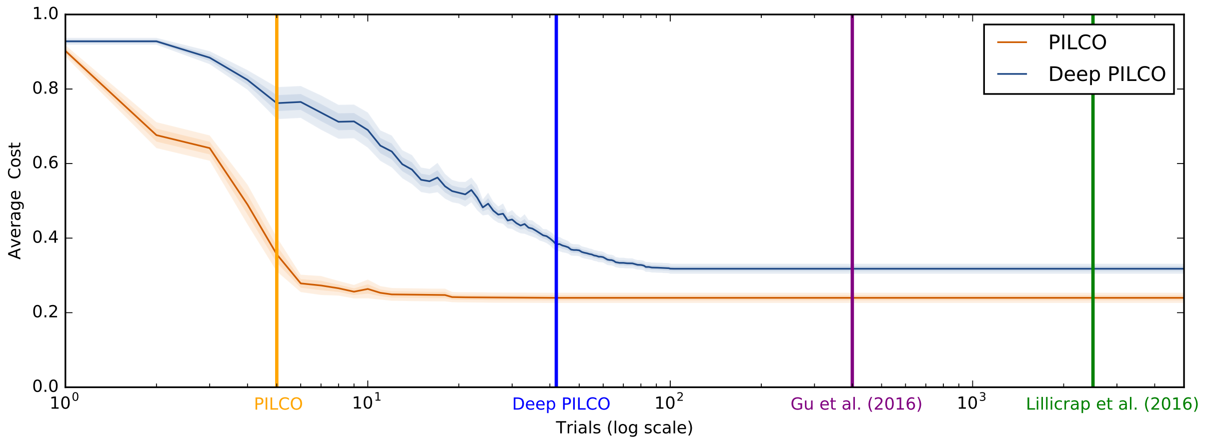
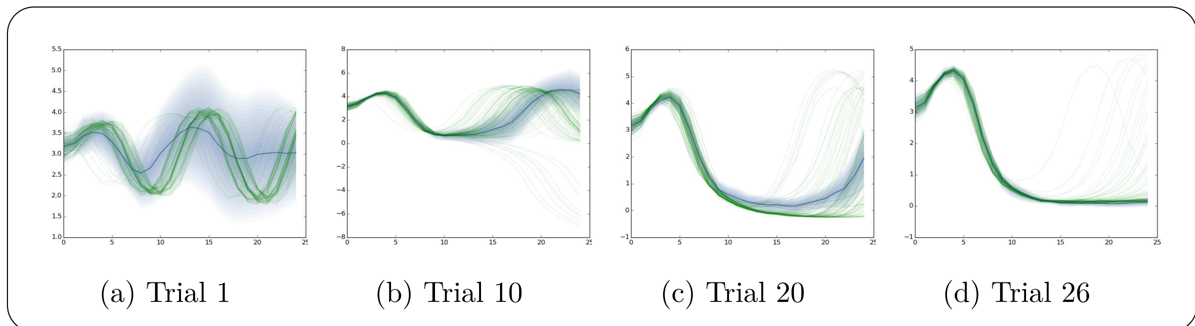


Fig. 5.7 Average cost per trial with standard error (on *log scale*) for PILCO and Deep PILCO on the cartpole swing-up task, averaging 40 experiment repetitions for both models. Vertical lines show estimates of number of trials required for model convergence (successfully balancing the pendulum with most experiment repetitions) for PILCO (yellow, 5 trials), Deep PILCO (blue, 42 trials), Gu et al. [2016] (purple, $\sim$ 400 trials) and Lillicrap et al. [2015] (green, $\sim 2,500$ trials).



(a) Trial 1 (b) Trial 10 (c) Trial 20 (d) Trial 26

Fig. 5.8 **Progression of model fitting and controller optimisation as more trials of data are collected.** Each x-axis is timestep t, and each y-axis is the pendulum angle θ in radians (see Figure 5.6). The goal is to swing the pendulum up such that $\text{mod}(\theta, 2\pi) \approx 0$. The green lines are samples from the ground truth dynamics. The blue distribution is our Gaussian-fitted predictive distribution of states at each timestep. **(a)** After the first trial the model fit (blue) does not yet have enough data to accurately capture the true dynamics (green). Thus the policy performs poorly: the pendulum remains downwards swinging between 2π and 4π. **(b)** After 10 trials, the model fit (blue) predicts very well for the fist 13 time steps before separating from the true rollouts (green). The controller has stabilised the pendulum at 0π for about 10 time steps (1 second). **(c)** After 20 trials the model fit and policy are slightly improved. **(d)** From trial 26 onward, the dynamics model successfully captured the true dynamics and the policy successfully stabilises the pendulum upright at 0π radians most trials.