Inference Compilation and Universal Probabilistic Programming Research Report for CS4340

Albin Jaldevik
r.a.jaldevik@student.tudelft.nl
5839408

Felix Kaubek
f.kaubek@student.tudelft.nl
5851246

December 7, 2025

1 Description of the Topic

The aim of Le et al. (2017) is to universally increase the inference efficiency of probabilistic programs, even if that means also increasing compile time as a trade-off. This is achieved by utilizing the capabilities of a neural network architecture, referred to as *artifact* in this paper, which is trained at compile time. The network was trained in such a way, that it would return the parameters of a proposal distribution for the corresponding sample statement, according to the address and instance of it. A drawback of this approach is, that exact inference is impossible, as this is just not possible with neural networks.

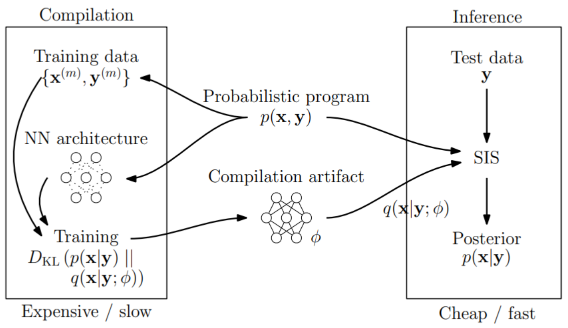
Compilation
Training data
$\{\mathbf{x}^{(m)}, \mathbf{y}^{(m)}\}$
NN architecture
Training
$D_{\mathrm{KL}}(p(\mathbf{x}|\mathbf{y}) \,||\, q(\mathbf{x}|\mathbf{y};\phi))$
Expensive / slow
Probabilistic program
$p(\mathbf{x},\mathbf{y})$
Compilation artifact
ϕ
$q(\mathbf{x}|\mathbf{y};\phi)$
Inference
Test data
$\mathbf{y}$
SIS
Posterior
$p(\mathbf{x}|\mathbf{y})$
Cheap / fast

Figure 1: Compilation and Inference Procedure [10]

One of the usual problems of using neural networks, especially in the context of probabilistic programming, is the low amount of data available. Le et al. circumvent this problem, by using an adapted version of the generative model as a way to generate limitless training data. To modify the generative model for this use case, all *observe* statements simply have to be replaced by *sample* statement (Fig 1.). This way the altered model can be used to generate as much data as wanted to train the artifact. This works, as the artifact learns which sample outcome affects the trace and how previous samples should affect the current one.
A problem the authors encountered consisted of the unequal amount of distribution parameters required per sample. To not have to restrain themselves to only using a single form of distribution the authors used a *Long-Short Term Memory*-based model (LSTM) (Fig 2.). To be able to use various forms of evidence, the authors further decided to use observe embeddings as one of the main inputs of the LSTM. This allows them to not only rely on numbers but also on visual evidence, for example, captchas.

During inference, the previously altered statements are turned back to their original observe form (Fig 1.). Then for every sample, the trained artifact is called to return a proposal. The actual inference algorithm used is sequential importance sampling with which they achieved very qualitative, but more importantly fast results. The increase in inference efficiency is drastic, cutting down to less than 100 milliseconds per captcha, from an original value of around 500ms to 8s depending on the algorithm.

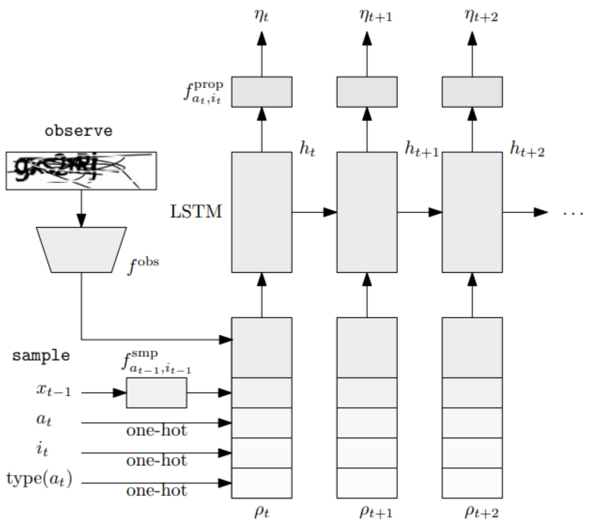

Figure 2: Architecture of the LSTM used [10].

2 Literature Study

- **Amortized Rejection Sampling in Universal Probabilistic Programming** (2022). [15] With the motivation to improve the efficiency and quality of amortized inference, both Le et al. and Naderiparizi1 et al. understand the problem inherent in traditional inference methods. While Le et al. introduces LSTM-based proposals to increase inference efficiency, this paper tackles the issue of unbounded loops in importance sampling estimators, especially when those are created through user-defined rejection sampling within generative models. It focuses on the problem of ensuring finite variance in the estimates, a factor important for quality and reliability of the sample. Naderiparizi1 et al. solve this by developing a new estimator that guarantees this quality, and further addresses some efficiency problems of prior methods. This is crucial to increasing the quality and robustness of rejection sampling, which is a commonly used inference method.

- **Compiling Discrete Probabilistic Programs for Vectorized Exact Inference** (2023). [17] In comparison to the approximate inference on continuous distributions that Le et al. focus on, discrete probabilistic programs can utilize different properties which make exact inference much more readily available. Pan et al. introduce *BayesTensor*, an approach that leverages tensor algebra, allowing it to employ the efficiency of vectorized implementations via tensor processing.
Unlike previous frameworks such as Infer.NET which is not optimized for discrete distributions and therefore encounters performance bottlenecks, and Dice which utilizes binary encoding and does not support range conditioning, *BayesTensor* demonstrates improved performance and handling of Bayesian Networks. It achieves this by relying on the advancements made in dense tensor algebra frameworks, facilitated by the boom of deep learning. Consequently, *BayesTensor* does not only improve upon the efficiency of previous frameworks but also extends the capabilities of exact inference.

- **Incremental Inference for Probabilistic Programs** (2018). [2] This paper by Cusumano-Towner et al. shares a common goal with Le et al., in that they both aim to enhance the inference process of universal probabilistic programs, albeit through different methodologies. Le et al. aim to optimize the sampling speed and quality during inference. Cusumano-Towner et al.'s paper paper addresses the reduction of resampling. This is possible by implementing a *trace translator*, which transforms a trace between related probabilistic programs. It furthermore makes sure that the adapted traces align with the target distribution, achieving a higher speed up the closer the two distributions are to each other. Cusumano-Towner et al. underscore the value of this trace adaption by showcasing how close alignments with the target distribution can lead to an improvement of the convergence rate of the inference process. Both papers try to improve efficiency by leveraging machine learning for probabilistic programming inference.

- **Attention for Inference Compilation** (2019). [7] This paper by Harvey et al. is a clear extension of Le et al. It continues the work on the exact same idea of enhancing SIS via neural networks, but it improves upon it by incorporating the attention neural architecture into the framework. The attention mechanism was popularised in 2017 which might be one of the reasons it was not considered in the original paper [20]. The attention mechanism, specifically dot-product attention, allows for the dynamic weighting of variable importance over an execution trace, which overcomes the limitations of the LSTM approach in capturing dependencies over large distances.
  One drawback is that the computational complexity grows quadratically. The authors discuss strategies for combating this and empirically show that they can improve the inference for certain models using this method. The authors further propose that blending of attention mechanisms with traditional inference methods could offer substantial imrpovements to speed and accuracy of probabilistic programming.

- **Accelerating Metropolis-Hastings with Lightweight Inference Compilation** (2021). [13] This paper by Liang et al. introduces an open-source framework called Lightweight Inference Compilation (LIC). The authors further extend the ideas from Le et al. and Harvey et al. [7] by using a graph neural network modeled after the Markov blanket relationships for random variables in the generative model. LIC utilizes the graph neural network to encapsulate the Markov blanket's relational strucutre, factilitating a model to learn the dependencies among the variables more effectively. This has the added benefit of the model being able to handle novel variables during inference, even if it had never previously encountered those. This further improves the usability, as it should be less sensitive to irrelevant variables during inference. The authors further provide an empirical comparison between LIC and IC that shows how the newer model can achieve an improvement in sample efficiency with fewer parameters and less compilation time.

- **Nested Compiled Inference for Hierarchical Reinforcement Learning** (2016). [9] This workshop paper by Le et al. demonstrates how probabilistic programming can be leveraged for complex hierarchical tasks in reinforcement learning, a field where various levels of granularity are important in decision-making. It does so by showcasing how to efficiently infer policies for sub-goals, reducing the need for intensive task-specific programming. By compiling decision-making hierarchies into the inference process, it exemplifies the potential for probabilistic models to adapt quickly and effectively to new tasks.
  However, the effectiveness heavily relies on the robustness of the initial program and even further on the quality of the hierarchical structure employed, factors, which require a lot of expertise to realize successfully. Despite that its innovative approach is remarkable, the necessary increased computational power at compile time is acknowledged by the authors, and the risk of overfitting leaves room for further research.

- **Using probabilistic programs as proposals** (2018). [3] This paper by Cusumano-Towner et al. builds upon the well-known Monte Carlo inference algorithm by letting users write specialized proposal programs to enhance sample efficiency. This makes it possible to leverage the domain expertise of users and can use domain-specific heuristics, to vastly enhance the sample efficiency, as well as explore the probable space effectively. The additional utilization of neural networks further increases the sampling efficiency. This allows the program to be written by non-experts, while the domain knowledge of said experts can still be incorporated in a meaningful way.
  While the inclusion of expert knowledge is well made and formulated, it heavily relies on this expert to be able to model a proposal distribution. This is not a trivial feat and could heavily influence the usability of this advancement in real-world applications. However the authors

argue with the correct tools and a bit of training, this problem can be mitigated. Both this paper, as well as Le et al., focus on previously under-investigated and utilized ways of increasing sampling efficiency.

- **Towards Verified Stochastic Variational Inference for Probabilistic Programs** (2019). [11] This paper by Lee et al. also works on utilizing the power of neural networks to improve the inference of probabilistic programs but focuses on a different goal. It aims to increase the reliability of stochastic variational inference (SVI), which is a fundamental algorithm used in deep probabilistic programming. They do so by proposing rules that can be automated by static program analysis to verify assumptions that lie at the base of SVI. If said assumptions are not met, the reliability of stochastic variational inference is strongly reduced, either through an undefined objective function or loss of convergence guarantee
The strengths of this paper are its usage of analysis and further implementation of automated verification rules, that vastly improve the reliability of SVI's. Its weakness however is the fact that only so many explicit models have been considered, which might hinder its generalisability.

- **Probabilistic Neural Programmed Networks for Scene Generation** (2018). [5] This paper by Deng et al. contributes to the field of probabilistic programming by proposing a novel framework for text-to-scene image generation, addressing the challenge of creating scene images with complicated semantics. Their work *PNP-Net*, encompasses a variational auto-encoder that composes scenes, models the object in the scene, and further renders the scene. This expands the usability of probabilistic programming to the generation of visual scenes and demonstrates its capabilities of encoding semantics.
The paper's strength is its ability to handle complex semantics, not only generating objects but also capturing their interaction and relationships among each other, even if some of them have not been fully learned beforehand. They further display an advanced use of variational auto-encoders. Its, for neural networks common, weakness is its dependence on data and further scalability is not commented on. Both this paper as well as Le et al. showcase the potential of utilizing probabilistic programming as a data generator in combination with a neural network. While Le et al. utilize it as a way to generate data to train an LSTM, Deng et al. utilize it to generate relations between objects.

3 Relation to Other Topics in the Course

- **Automated Variational Inference in Probabilistic Programming:** This paper by Wingate et al. introduces a method to simplify variational inference in probabilistic models, by automating the transformation of probabilistic programs into inference algorithms. This approach reduces the expertise and time required to apply Bayesian inference in complex models. Both Le et al. and this paper highlight innovations in increasing the efficiency of inference in probabilistic programming, though they focus on different techniques: one on automation through variational inference, one by leveraging neural networks to speed up the process. [21]

- **Deep Amortized Inference for Probabilistic Programs:** These authors try to optimize inference, by introducing a *guide program*, which is a trained adjustment for sampling statements. The aim is to reduce the amount of rejected traces that will be generated throughout inference. In comparison to Le et al., it tries to add to various inference methods, while Le et al. tries to replace existing methods. [19]

- **Divide, Conquer, and Combine: a New Inference Strategy for Probabilistic Programs with Stochastic Support:** This paper by Zhou et al. incorporates the well known principle of divide and conquer by splitting a universal probabilistic program up into multiple straight line programs. These improve the speed by being able to divide the program precompilation into different execution paths and compiling them with a *fixed* support. It shares the same goal as Le et al., improving inference efficiency of universal probabilistic programs, but focuses on algorithmic improvements rather than neural networks. [23]

- **Rethinking Variational Inference for Probabilistic Programs with Stochastic Support** Building upon Zhou et al. [23] this paper also focuses on dividing programs into SLP's and improving efficiency this way. Furthermore, the goal is to facilitate stochastic support. Both this papers and Le et al. aim to improve efficiency, but have different strategies to do so, this paper furthermore extends the space of possible models. [18]

- **SMCP3: Sequential Monte Carlo with probabilistic program proposals:** This paper by Lew, Matheos et al. introduces *SMCP3* an improved version of Sequential Monte Carlo (SMC) inference. It allows for far more complex proposals than SMC, and delivers more refined samples and particle weights. The motivation is very different to Le et al., the focus is more on extending SMC and increasing the complexity allowed by the algorithm to further increase the modeling space. The increased efficiency is more of a byproduct than a goal. [12]

- **Efficient Probabilistic Inference in the Quest for Physics Beyond the Standard Model:** Both this and our paper focus on enhancing inference efficiency by incorporating deep recurrent neural networks to parameterize distributions. However, the approach is very different. This paper introduces a cross-platform probabilistic execution protocol that directly interacts with existing large-scale simulators. The motivation behind both papers could also not be any different. Le et al. aim to increase inference efficiency, while Byden et al. aim to model physical events at an unprecedented scale. [1]

- **DeepProbLog: Neural Probabilistic Logic Programming** Both this and Le et al. focus on integrating deep learning with probabilistic programming. While Le et al. aim to improve inference in universal programs, across languages and without strong restrictions. DeepProbLog focuses on expanding ProbLog, which itself is an extension of Prolog, and aims to increase the inference speed of logical reasoning in combination with probabilistic inference. [14]

- **DeepStochLog: Neural Stochastic Logic Programming** This paper extends the work of DeepProbLog and aims to improve upon its scalability and efficiency limiations by introducing neural grammar rules. it further enables specific inference and learning algorithms, which speed up both the training as well as inference part. Like DeepProbLog and Le et al. it focuses on improved inference efficiency by utilizing the capabilities of machine learning. [22]

- **Denoising Diffusion Probabilistic Models** This paper, as well as Le et al. partly, focus on the capabilities of probabilistic programms as a generative tool. Both utilize machine learning, but Le et al. take advantage of it to train their artifact, while for Ho et al. the generative part is the goal. The paper aims to generate new pictures out of noise, by training a denoising function on datasets. [8]

4 Experimental Reproduction

We started by going through the Gen documentation and tutorials. There are great examples of how to perform different types of inference algorithms as well as how to interact with neural networks in Julia. We did not find an existing implementation of our paper online but there are some related implementations (such as the lightweight inference one in Python [13]).

We decided to focus on understanding the complied inference algorithm by implementing a simplified version from scratch in Gen. The main simplification we decided on was to focus on a specific probabilistic program instead of supporting general programs. The sample principles still hold but this allowed us to work with something more tangible when we know what the variables represent. The problem we decided to focus on is the classic Alarm/Burglar discrete Bayesian network model depicted in Figure 3 and Listing 1. One of the main benefits of using this model is that it is simple enough to analytically analyze the posterior.

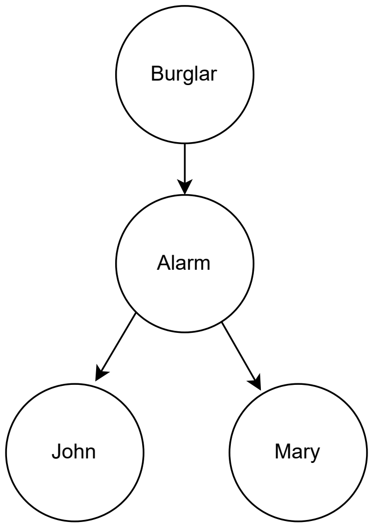

Figure 3: Burglar Probabilistic model.

```
const P_ALARM_GIVEN_BURGLAR = 0.9
const P_ALARM_GIVEN_NO_BURGLAR = 0.15
const P_JOHN_GIVEN_ALARM = 0.6
const P_JOHN_GIVEN_NO_ALARM = 0.3
const P_MARY_GIVEN_ALARM = 0.7
const P_MARY_GIVEN_NO_ALARM = 0.1

@gen function burglar_model(chance)
burglar ~ bernoulli(chance)

if burglar
alarm ~ bernoulli(P_ALARM_GIVEN_BURGLAR)
else
alarm ~ bernoulli(
    P_ALARM_GIVEN_NO_BURGLAR)

if alarm
john ~ bernoulli(P_JOHN_GIVEN_ALARM)
mary ~ bernoulli(P_MARY_GIVEN_ALARM)
else
john ~ bernoulli(P_JOHN_GIVEN_NO_ALARM)
mary ~ bernoulli(P_MARY_GIVEN_NO_ALARM)
```

Listing 1: Probabilistic model in Julia psudocode.

We implemented two different versions of the complied inference algorithm to conduct experiments with. The full code is submitted in a separate file. In the first, simple version, we use a standard feedforward network to learn the complete proposal distribution at once. The neural network has one hidden layer with 25 neurons with Relu activation function. The final layer is a Sigmoid function forcing the output to be between 0 and 1 since it is used as a parameter to the Bernoulli sampling function. We train the model on sampled data and it learns the conditional probabilities for Burglar and Alarm given John and Mary. We calculate the analytical conditional probabilities for $P(B|J, M)$ (Burglar) and $P(A|J, M)$ (Alarm) and compare them to the output of the trained neural network for this case. The results in Table 1 and 2 show that our network successfully learns the analytical posteriors from training. We then use this network as the proposal distribution in the Metropolis-Hastings Algorithm.

Table 1: Probabilities for $P(B|J, M)$ from NN. NN is the model output and A the analytical value.

	Mary = 0	Mary = 1
John = 0	NN: 0.03, A: 0.03	NN: 0.22, A: 0.22
John = 1	NN: 0.07, A: 0.08	NN: 0.32, A: 0.32

Table 2: Probabilities for $P(A|J, M)$ from NN. NN is the model output and A the analytical value.

	Mary = 0	Mary = 1
John = 0	NN: 0.05, A: 0.05	NN: 0.55, A: 0.54
John = 1	NN: 0.16, A: 0.16	NN: 0.80, A: 0.80

We also implemented the recurrent architecture using a Long-Short Term Memory (LSTM) and sequential importance sampling. The LSTM has the great benefit that it can also incorporate the previous samples into every individual parameter proposal. We use an LSTM architecture with hidden dimension 10 and the outputs are passed through an additional neural network and Sigmoid function (proposal layer) to produce the final parameters. Once again, we investigate some outputs from the model to see what it learned. When we sample the first variable (Burglar) we do not have any previous sample outputs so we only condition on the observations (John/Mary), the results are shown in Table 3. When we sample the Alarm we also pass what we sampled for the Burglar as well as the LSTM hidden state which allows the model to condition on the outcome of Burglar. The learned conditional probabilities generated from the LSTM are shown in Table 4. We did not calculate these analytically but it is apparent that the model learns that if Burglar is true then there is a higher probability of Alarm.

Table 3: Probabilities for $P(B|J,M)$ from LSTM. NN is the model output and A the analytical value.

	Mary = 0	Mary = 1
John = 0	NN: 0.04, A: 0.03	NN: 0.21, A: 0.22
John = 1	NN: 0.07, A: 0.08	NN: 0.33, A: 0.32

Table 4: Probabilities for $P(A|B,J,M)$ from LSTM.

		Mary = 0	Mary = 1
John = 0	B = 0	0.03	0.43
	B = 1	0.65	0.95
John = 1	B = 0	0.1	0.72
	B = 1	0.86	0.97

To investigate the inference performance of our implementation we constructed an experiment. We perform inference where the performance metric is the accuracy of our prediction on Burglar and Alarm (both must be correct to score) given if John and Mary called. The only thing we are changing is the proposal distribution passed to Metropolis-Hastings. In Figure 4 we compare 5 different proposals. Block is the default proposal in Gen, and we also included the untrained versions of the simple feed-forward network (NN) as well as the LSTM. To better understand how the proposal affects inference we experimented with different (all relatively low) Metropolis-Hastings sampling budgets.

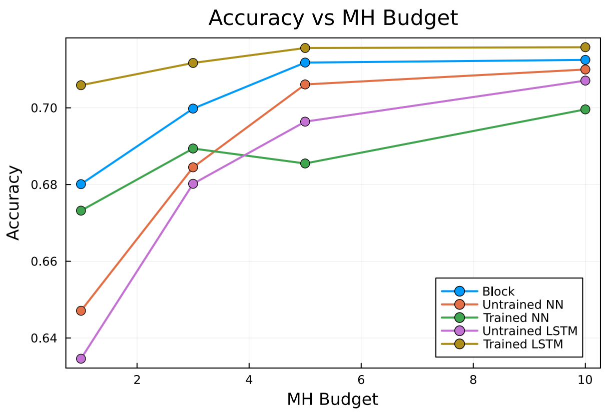

Figure 4: Estimated inference accuracy for different proposal distributions and budgets for 10 000 traces.

The results in Figure 4 show that our trained LSTM architecture outperforms the other proposals. We can also observe that the choice of proposal distribution has a larger impact the lower the inference budget. As expected, we also generally see that a higher budget improves inference. One slightly surprising result is how poorly the feedforward (NN) proposal performed, especially for higher budgets but this could also be an outlier. We conclude that the recurrent compiled inference network (LSTM) seems to be the way to go.

5 Research Proposal

Ideas for follow-up projects based on Le et al:

- **Other Inference Procedure**. One idea could be to transfer the idea of compiled inference to another inference procedure than SIS. There are numerous to choose from and we hypothesize that the idea could also improve performance for other methods.
- **Other Neural Architectures**. An easy extension is to change the neural architecture from LSTM to Attention/Transformer but this has already been done by the authors themselves.

- **Meta Learning**. In the current framework, the authors had to hand-pick multiple parts of the architecture to fit their specific problem. For instance, the observe embedding for the captchas is fully custom. What if we can automatically tune these necessary components to adapt to different programs via meta-learning?

We consider the meta-learning proposal the most interesting and will investigate it further below.

5.1 Problem Description

One problem in the approach of Le et al. is that it requires both domain knowledge and an understanding of neural architectures to handcraft certain parts of the inference algorithm for each specific probabilistic program. For example, the authors present a custom convolutional neural network structure used in the observe embedding of the captcha images. Coming up with new architectures for every task is a tedious process and this issue is also highlighted by the authors in the original paper as something worth exploring improvements to. It is not only the observe embedding that might benefit from fine-tuning but also other aspects such as the dimensionality of the LSTM.

5.2 Experimental Motivation

To further investigate the impact of hyperparameters and the need for meta-learning we used our code implementation to conduct some experiments. We focus on investigating how the hidden dimension of the LSTM artifact affects the performance of the inference algorithm.

Figure 5 shows the evolution of the training loss for the LSTMs proposals for various hidden dimensions. The loss is the negative log-likelihood and we can clearly see that the hidden dimension has an effect on it. The architecture with only one hidden dimension does not seem to be able to converge to the same loss as the models with additional parameters.

Figure 6 shows the accuracy in the Burglar experiment for the trained LSTM proposals with different hidden dimensions for MH budget 1. It is difficult to conclude exactly what effect the hidden dimension has on the actual inference performance but it is clear that it has some impact. These two experiments therefore support the claim that there could be benefits of tuning the hyperparameters of the artifact via meta-learning.

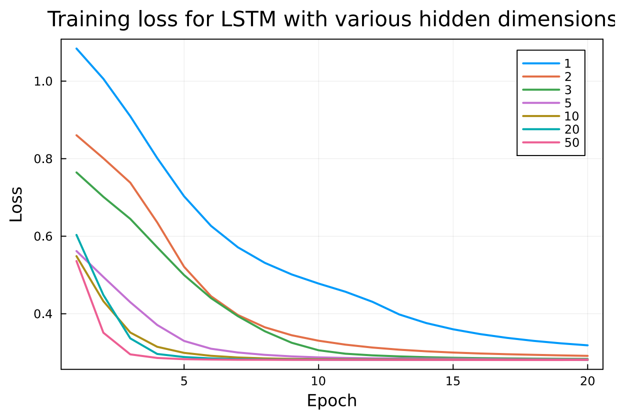

Figure 5: Training Loss (negative log-likelihood) evolution for different LSTM hidden dimensions.

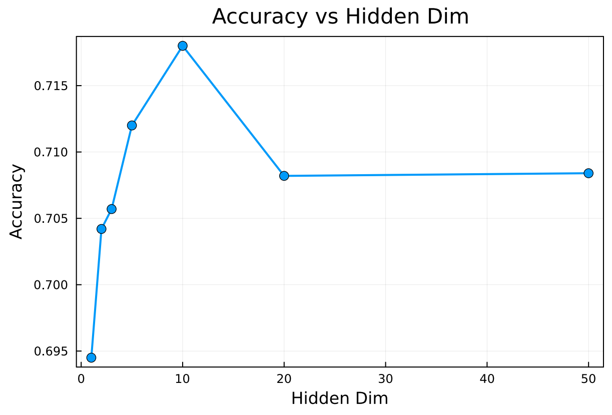

Figure 6: Accuracy on the Burglar experiment from section 4 for different LSTM hidden dimensions with MH budget 1.

5.3 Technical Approach

The meta-learning problem can be thought of as learning how to learn. What we essentially want to do is come up with an architecture that can tackle new, previously unseen problems without starting from scratch and relying on human judgment. The main parts of the complied inference algorithm we would like to meta-learn is

1. The observe embedding.
2. The sample embedding.
3. The proposal layers.
4. The hidden dimension of the LSTM.
5. The number of stacked LSTMs.

Our proposed technical solution is to increase the number of free parameters in the model and then use a meta-learning technique like MAML (Model-Agnostic Meta-Learning) [6] to enable efficient learning for new, unseen tasks. To clarify, we will exemplify using the observe embedding. In the original paper, the authors used a custom convolutional neural network (CNN) for the observed embedding. CNNs are traditionally great for vision tasks but since have fewer parameters than a fully connected layer there are certain things they can't learn. For example, CNNs are known for being translation invariant meaning that they have a hard time taking the position of objects into account. Since we want to construct an artifact that can learn any function we will therefore use fully connected layers instead of CNNs. The universal approximation theorem states that with enough width and certain activation functions, feedforward networks should be able to learn any continuous function [4]. It should therefore be possible for a fully connected network to learn just as complex functions as a CNN. The same principle of increasing the number of free parameters would apply to the other parts of the architecture that we would like to meta-learn, for example, using an extra large hidden LSTM dimension.

The proposed solution would massively increase the number of free parameters in the neural network. This will traditionally cause two major issues. The first issue is that increasing the number of parameters in a neural architecture can impede the model from generalizing and instead overfitting the training data. We hypothesize that we can salvage this major drawback in this case by simply generating additional training data. Since we, in this framework, create the training data ourselves by sampling from the probabilistic model we can theoretically generate an infinite amount of training data which should prevent the model from overfitting. If this does not solve the issue completely we could deploy other common regularization techniques such as dropout layers or freezing parts of the parameters.

The second major issue is that training a larger model will most likely be extremely computationally and time-consuming. To counteract this issue, we propose using Model-Agnostic Meta-Learning (MAML) [6]. The idea in MAML is that we try to find a set of initial model parameters θ that makes it easy to fine-tune the model for various tasks. For example, if we would primarily work with image evidence then the model might learn that an observe embedding similar to a CNN would be a good starting point for the model parameters. This should counteract the increased training time and make the artifact quickly adapt to new tasks.

To utilize MAML, we need to decide on a set of tasks to train the model with. The tasks should be representative of what the model might be used for in practice. The tasks could for example be various image-based tasks such as captcha in different resolutions as well as non-vision models. Technically a task would be fully defined by a probabilistic model which we could generate multiple of automatically. Since we are training the model end to end we could utilize the same loss function as in the original paper (negative log-likelihood of the proposal distribution).

We refer to the original MAML paper [6] for the exact algorithm implementation details but the broad technical idea is essentially to perform stochastic gradient by sampling tasks in every update step to learn a set of common parameters as close to the optimal parameters for each individual task as possible. See

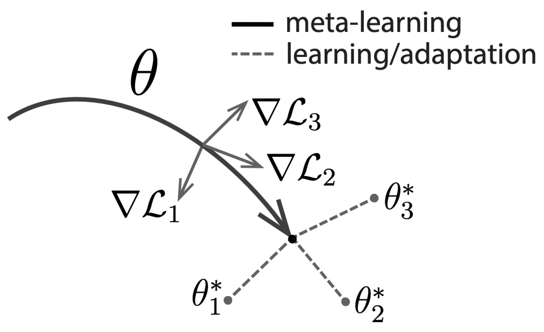

Figure 7: Ilustration of the MAML algorithm [6].

Figure 7 for an illustration from the paper.

5.4 Experiments

1. Reproduce the experiments in the original paper and compare their embeddings to the meta-learned ones. There can be three different outcomes. Either the meta-learning outperforms their custom embeddings or the meta-learning finds the same embeddings which is also an interesting result. The least satisfying result would be if the meta-learning performs worse than the author's embeddings but this is of course a possibility. We might then have to draw the conclusion that domain knowledge still beats this approach but that there could still be value in the meta-learning approach for users lacking knowledge of how to design custom embeddings.

2. Construct a hypothetical scenario to show that the embeddings need adjustments to different problems. For example, take the original captcha problem and scale the resolution of the images by a factor 2, 4, 8, etc. The original embeddings that the authors use will probably work well for their captcha resolution but we hypothesize that the convolutions won't work as well for adjusted resolutions, even if the images are semantically similar. The best scenario would be if our meta-learning approach manages to adjust the hyperparameters to work as well for the new resolutions (and outperform the original embedding).

3. It would also be interesting to test which parameters the meta-learning algorithm works best with. Meaning, that we can give it control over various elements like dimensions of the LSTM, or the way embeddings work and also take that control away, and then see which version works best. This would furthermore not only test the capabilities of the meta-learning algorithm but also give insight into the importance of various parameters and show how impactful changes truly are. This would be classified as ablation studies.

5.5 Potential Issues

The meta-learning approach might face some issues. Primarily, the computational resources required to train the model might be overwhelming depending on how broad the range of tasks (probabilistic models) one wants it to adapt to. The number of parameters would probably be very high which would require both a lot of memory and time to train. One potential solution is to use a more modern, simplified learning procedure such as Reptile [16]. The algorithm utilizes a first-order approximation of approximation of MAML to reduce the computational resources needed.

Another potential solution could be to not have one model for every task but instead have a couple of standard models for each broad task category. For example, there could be one model for all image-based tasks, one for videos, and one for tabular data. This would allow us to reduce the number of trainable parameters for each individual model since we could, for instance, assume that CNNs can be used for all image-based tasks.

References

[1] Atilim Gunes Baydin, Lukas Heinrich, Wahid Bhimji, Bradley Gram-Hansen, Gilles Louppe, Lei Shao, Prabhat, Kyle Cranmer, and Frank D. Wood. Efficient probabilistic inference in the quest for physics beyond the standard model. *CoRR*, abs/1807.07706, 2018.

[2] Marco Cusumano-Towner, Benjamin Bichsel, Timon Gehr, Martin Vechev, and Vikash K.

Mansinghka. Incremental inference for probabilistic programs. *SIGPLAN Not.*, 53(4):571–585, jun 2018.

[3] Marco F. Cusumano-Towner and Vikash K. Mansinghka. Using probabilistic programs as proposals. *CoRR*, abs/1801.03612, 2018.

[4] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

[5] Zhiwei Deng, Jiacheng Chen, YIFANG FU, and Greg Mori. Probabilistic neural programmed networks for scene generation. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[6] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International conference on machine learning*, pages 1126–1135. PMLR, 2017.

[7] William Harvey, Andreas Munk, Atılım Güneş Baydin, Alexander Bergholm, and Frank Wood. Attention for inference compilation. *arXiv preprint arXiv:1910.11961*, 2019.

[8] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *CoRR*, abs/2006.11239, 2020.

[9] Tuan Anh Le, Atılım Günes Baydin, and Frank Wood. Nested compiled inference for hierarchical reinforcement learning. In *NIPS Workshop on Bayesian Deep Learning*, 2016.

[10] Tuan Anh Le, Atilim Gunes Baydin, and Frank Wood. Inference compilation and universal probabilistic programming, 2017.

[11] Wonyeol Lee, Hangyeol Yu, Xavier Rival, and Hongseok Yang. Towards verified stochastic variational inference for probabilistic programs. *Proc. ACM Program. Lang.*, 4(POPL), dec 2019.

[12] Alexander K. Lew, George Matheos, Tan Zhi-Xuan, Matin Ghavamizadeh, Nishad Gothoskar, Stuart Russell, and Vikash K. Mansinghka. Smcp3: Sequential monte carlo with probabilistic program proposals. In Francisco Ruiz, Jennifer Dy, and Jan-Willem van de Meent, editors, *Proceedings of The 26th International Conference on Artificial Intelligence and Statistics*, volume 206 of *Proceedings of Machine Learning Research*, pages 7061–7088. PMLR, 25–27 Apr 2023.

[13] Feynman Liang, Nimar Arora, Nazanin Tehrani, Yucen Li, Michael Tingley, and Erik Meijer. Accelerating metropolis-hastings with lightweight inference compilation. In *International Conference on Artificial Intelligence and Statistics*, pages 181–189. PMLR, 2021.

[14] Robin Manhaeve, Sebastijan Dumancic, Angelika Kimmig, Thomas Demeester, and Luc De Raedt. Deepproblog: Neural probabilistic logic programming. *CoRR*, abs/1805.10872, 2018.

[15] Saeid Naderiparizi, Adam Scibior, Andreas Munk, Mehrdad Ghadiri, Atilim Gunes Baydin, Bradley J. Gram-Hansen, Christian A. Schroeder De Witt, Robert Zinkov, Philip Torr, Tom Rainforth, Yee Whye Teh, and Frank Wood. Amortized rejection sampling in universal probabilistic programming. In Gustau Camps-Valls, Francisco J. R. Ruiz, and Isabel Valera, editors, *Proceedings of The 25th International Conference on Artificial Intelligence and Statistics*, volume 151 of *Proceedings of Machine Learning Research*, pages 8392–8412. PMLR, 28–30 Mar 2022.

[16] Alex Nichol, Joshua Achiam, and John Schulman. On first-order meta-learning algorithms, 2018.

[17] Jingwen Pan and Amir Shaikhha. Compiling discrete probabilistic programs for vectorized exact inference. In *Proceedings of the 32nd ACM SIGPLAN International Conference on Compiler Construction*, CC 2023, page 13–24, New

York, NY, USA, 2023. Association for Computing Machinery.

[18] Tim Reichelt, Luke Ong, and Tom Rainforth. Rethinking variational inference for probabilistic programs with stochastic support, 2023.

[19] Daniel Ritchie, Paul Horsfall, and Noah D. Goodman. Deep amortized inference for probabilistic programs, 2016.

[20] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[21] David Wingate and Theophane Weber. Automated variational inference in probabilistic programming, 2013.

[22] Thomas Winters, Giuseppe Marra, Robin Manhaeve, and Luc De Raedt. Deepstochlog: Neural stochastic logic programming. *CoRR*, abs/2106.12574, 2021.

[23] Yuan Zhou, Hongseok Yang, Yee Whye Teh, and Tom Rainforth. Divide, conquer, and combine: a new inference strategy for probabilistic programs with stochastic support. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 11534–11545. PMLR, 13–18 Jul 2020.