UNIVERSITY OF
CAMBRIDGE
Department of Computer
Science and Technology

Ain't Nobody Got Time For That: Budget-aware Concept Intervention Policies

Thomas Yuan

Downing College

May 2024

Submitted in partial fulfillment of the requirements for the
Computer Science Tripos, Part III

Total page count: 53

Main chapters (excluding front-matter, references and appendix): 37 pages (pp 10–46)

Main chapters word count: 10854

Methodology used to generate that word count:

- Using appropriate flags to tell texcount to only count the main chapters, then

```
$ let total=$(texcount -merge main.tex | awk '/Words in text/ {print $4}')+
$(texcount -merge main.tex | awk '/Words in headers/ {print $4}')+
$(texcount -merge main.tex | awk '/Words outside text/ {print $6}');
echo $total

10854
```

Declaration

I, Thomas Yuan of Downing College, being a candidate for the Computer Science Tripos, Part III, hereby declare that this report and the work described in it are my own work, unaided except as may be specified below, and that the report does not contain material that has already been used to any substantial extent for a comparable purpose.

Signed: Thomas Yuan

Date: May 26, 2024 -

Abstract

Concept Bottleneck Models (CBMs) are designed for high interpretability by predicting human-interpretable concepts from input, and then predicting labels from these concepts. Experts can intervene on the predicted concepts by correcting them to improve the accuracy of the predicted label. Previous studies have shown that the choice of intervention concepts can have significant impacts on the accuracy of the CBM. Since experts have limited time, it is important to develop ways to determine the best concepts to intervene on to maximize the performance of the CBM while minimizing the intervention costs. Given a budget for interventions, we want to develop methods that achieve the best model performance using the limited number of interventions. We investigate methods to learn a non-greedy policy that determines an optimal set of concepts to intervene on for a given budget, to maximize the accuracy of the CBM.

Past methods in finding intervention policies are limited in only using greedy policies that attempt to maximize the CBM's at each step. They do not incorporate the notion of intervention budgets, which we believe is an important issue to consider when using CBMs for real-life applications. We hypothesize that non-greedy policies can find more optimal concepts to intervene on for a given budget, as they maximize the final performance after a number of interventions rather than at each step. To learn a non-greedy policy, we model the problem of determining what concepts to intervene on as a Markov Decision Process and apply Reinforcement Learning by training the model to maximize the final performance of the CBM after a set number of interventions. We also use a surrogate model to model the distribution of concepts to provide intermediate rewards and auxiliary information to the Reinforcement Learning agent. Using a pre-trained surrogate model, a Reinforcement Learning agent is trained in conjunction with a CBM, which learns a policy for determining the choice of concepts to intervene on for a given budget.

We first show that non-greedy policies can outperform greedy policies for different budgets. We then show that we can learn a non-greedy policy using Reinforcement Learning which outperforms existing greedy policies for different intervention budgets, achieving similar intervention performance with up to 25% less interventions. Simultaneously the model maintains similar performance under the absence of interventions. This shows that Reinforcement Learning can learn intervention policies more optimal than existing approaches for different budgets, opening further research in non-greedy intervention policies.

Acknowledgements

This project would not have been possible without the wonderful support of my lovely supervisors Mateo Espinosa Zarlenga, Prof Mateja Jamnik and Dr. Zohreh Shams. I would also like to thank my friends and family for their support.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Regular supervised Machine Learning (ML) models learn to predict the labels of inputs. Concept Bottleneck Models (CBMs) [7]and Concept Embedding Models (CEMs) [26] are ML models designed to increase the interpretability of model predictions by decomposing a model into two submodels, splitting the original process into predicting a set of human-interpretable concepts / features present in the input, then predicting the label using these concepts. This allows us to understand the reasoning behind the predications better, mitigating some of the downsides associated with using ML models as "black-box" models, especially in areas where decisions made can be critical to human safety, such as medicine and criminal justice. During inference time, professionals can intervene on CBMs by correcting the predicted concepts, leading to more accurate predicted labels. This leads to the problem of determining the concepts to intervene on in order to achieve the best model performance.

Previous studies have shown that the choice of concepts to intervene on can have significant impacts on the accuracy of the model [3, 27], and thus it is important to find a good policy to determine what concepts we should intervene on. Due to the costs associated with performing such an intervention, we introduce budgets, which add constraints on the number of interventions allowed. This reflects real-life scenarios where there exists limits on the number of times we can query an expert to intervene on concepts.

The problem then becomes finding policies that can determine the best concepts to perform interventions on for a given budget. In particular, we investigate the research question of finding a non-greedy policy to determine the set of concepts to intervene on for a budget. We hypothesize that non-greedy policies yield more optimal solutions as they maximize the model accuracy after a number of interventions rather than at each step.

Past methods have mainly focused on greedy approaches, which includes heuristic-based approaches such as intervening on concepts that minimize the model's uncertainty [3], or ML-based approaches that learn a greedy policy like in IntCEMs [27]. Additionally existing approaches either attempt to model the distribution of concepts and use that to

sample interventions [25], or directly learn a policy [27], while limited work has been done on combining the capabilities of models from these two approaches.

To find a non-greedy policy, we investigate using Reinforcement Learning (RL) [19] to learn a policy that maximizes the model accuracy after a set number of interventions, using the accuracy of the CBM as a reward to train the RL agent. In order to guide the agent throughout the intervention process, we use surrogate models that model the distribution of concepts using a variant of normalizing flow models. These models learn a latent distribution and an invertible transformation to model the distribution of the concepts. This allows us to compute the conditional likelihood of concepts and use as an intermediate reward to the RL agent. Additionally we can sample from the underlying distribution to obtain concepts with the highest likelihood, which provides auxiliary information to the agent.

After pretraining a surrogate model to learn the distribution of concepts, we train a Reinforcement Learning agent in conjunction with a CEM, which we name RLCEM, to learn a non-greedy intervention policy, using the expected information gain from the surrogate model as a reward, and a final reward based on the accuracy of the CBM after all interventions. During training, the agent learns a policy to select the optimal concepts for interventions for different budgets, whereas the CBM is also trained on the sampled interventions to increase its sensitivity to interventions determined by the learnt policy.

To evaluate this approach, we train RLCEM on two different datasets and compare its performance to IntCEM, the state-of-the-art model for intervention performance that learns a greedy policy for interventions.

To sum up, this project has the following contributions:

- Introduce budgets into the problem setting and shown that non-greedy policies can outperform greedy policies.
- Develop and train Surrogate models by adapting AC Flow models to be used to model conditional concept likelihoods.
- Train RL Concept Embedding Models that utilize Reinforcement Learning that can learn a non-greedy intervention policy with better intervention performance than existing methods, achieving better performance with up to 25% less interventions. At the same time the model maintains similar performance under the absence of interventions.
- Standardize the RL environment of RLCEMs inheriting from the Gymnasium Interface [22] which can be used in future research in RL-based intervention policies.

However, we note that this approach also has its drawbacks, including higher time complexity, and its performance is very dependent on the nature of the task and concepts, which affects the robustness of the method.

Chapter 2

Background and Related Work

In this chapter, we explain the background knowledge for this project, including CBMs, CEMs, IntCEMs, and other existing intervention policies. All existing intervention policies adopt a greedy approach to performing interventions, whereas we believe that non-greedy intervention policies can outperform greedy intervention policies. We show how existing research cannot answer our research question of learning a non-greedy intervention policy.

2.1 CBM

Concept Bottleneck Models (CBMs), initially proposed by Koh et al. [7], are a class of models that consist of a model g that learns a mapping from input $\mathbf{x}$ to a concept vector $\mathbf{c} = g(\mathbf{x})$, where $\mathbf{c}$ is a multi-hot encoding of the concepts present in the input, and a model f that learns a mapping from such concepts vector c to the output label $\mathbf{y} = f(\mathbf{c})$, as shown in Figure 2.1. These types of model can be created by simply adding a new layer in traditional models with the same number of activations as the number of concepts, where this layer is referred to as the bottleneck. Henceforth we refer to g as the "concept predictor $\mathbf{x} \to \mathbf{c}$ model" and f as the "label predictor $\mathbf{c} \to \mathbf{y}$ model".

CBMs allow for interventions, which are using experts to correct the intermediate concepts predicted by the $\mathbf{x} \to \mathbf{c}$ model to improve the performance of the $\mathbf{c} \to \mathbf{y}$ model, which is illustrated in Figure 2.2. To train a CBM, we utilize a combination of a concept loss $\mathcal{L}_{\text{concept}}$ that measures the discrepancy between the predicted concepts $\hat{\mathbf{c}}$ and actual concepts $\mathbf{c}$, as well as a label loss $\mathcal{L}_{\text{label}}$ that measures the discrepancy between the predicted label $\hat{\mathbf{y}}$ and actual label $\mathbf{y}$.

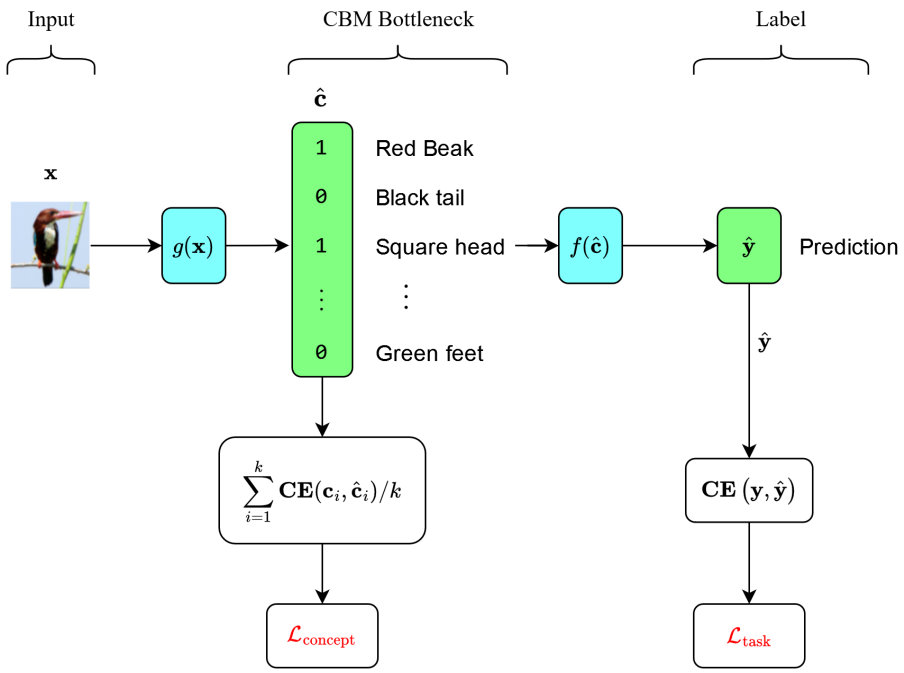

Figure 2.1: The CBM Architecture [7].

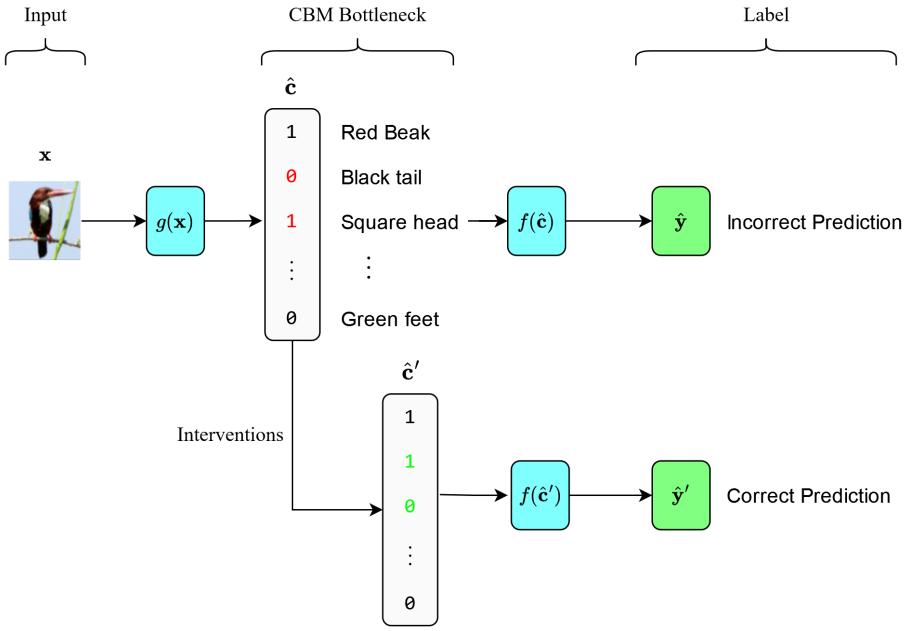

Figure 2.2: An illustration of intervening on the concepts predicted by a CBM.

2.2 CEM

While CBMs proved to be useful in achieving machine learning models with high interpretability, they do not perform as well as traditional models. This is because they heavily rely only on a set of human-interpretable concepts, which limits the performance of the model as traditional models can extract information outside these concepts [1]. This is even more apparent if the dataset does not contain a complete set of concepts that cover all features that can be used to predict the label, which is common in real-life.

To overcome this trade-off between performance and interpretability, Concept Embedding Models (CEMs) were proposed by Zarlenga et al. [26], which utilize learnable embeddings to replace the original CBM bottleneck, learning two embedding vectors for each concept: one for when the concept is and is not present.

CEMs successfully solves the trade-off problem between performance and interpretability, allowing for similar performance to traditional models while maintaining the interpretability, along with high concept accuracy. This is because the embeddings encode more information in the concept representations, for example information related to concepts not present in the dataset. This is referred to as concept leakage, where The additional information in the CEM bottleneck compared to binary representations in CBMs lead to a better performing label predictor model. It has been shown experimentally that CEMs are able to provide better performance for concept-incomplete dataset tasks (where the concepts do not cover all features present in input), and these learnt concept embedding representations effectively represent the true concepts measured by an alignment score [26].

This architecture also allows for interventions during run-time. By simply replacing the output of the concept predictor model with the embeddings that represent the true concepts, we can correct the concept predictions and improve the performance of the model.

2.3 IntCEM

Building on top of CEMs, Zarlenga et al. [27] introduced Intervention-aware CEM (IntCEM), CEMs that are augmented with a learnable concept intervention policy model. IntCEMs' novelty lies in framing the problem of training a CEM and finding an intervention policy as a joint optimization problem by augmenting existing CEMs with a trainable intervention policy model ψ. This approach offers significant improvements in performance after interventions while maintaining similar performance without interventions. IntCEM achieves this because the intervention policy model learn a good intervention policy specific to the CEM, and the CEM also learns to be more sensitive to interventions by the model, through the introduction of an intervention loss $\mathcal{L}_{\text{intervention}}$ and task loss $\mathcal{L}_{\text{task}}$ for the intervened concepts.

During training, ψ first samples intervention logits ω for the next concept to intervene on, then a Cross-Entropy loss $\mathcal{L}_{\text{intervention}}$ is used to compute the discrepancy with the output of a greedy optimal policy, found by searching over all concept to yield the concept that leads to the highest increase in model performance when intervened. This is referred to as Behavioural Cloning [2] approach where ψ learns to mimic the behaviour of a greedy optimal policy.

Training using true concept labels increases the model's sensitivity to interventions, which

was demonstrated by Koh et al. [7], leading to better intervention performance. IntCEM incorporates this idea by computing task loss L_{task} also using the intervened concepts by ψ during training. Not only does this increase the model's sensitivity to interventions, it specifically increases the model's sensitivity to interventions sampled by ψ to further improve its intervention performance. The overall training loop of IntCEM is shown in Figure 2.3. At each intervention step, the intervention binary mask $\boldsymbol{\mu}$ is updated according to the next concept to intervene η sampled from the output of the intervention policy model ψ.

The final loss is computed as a weighted average of the three losses, given by

$$\mathcal{L} = \lambda_{\text{concept}}\mathcal{L}_{\text{concept}} + \lambda_{\text{task}}\mathcal{L}_{\text{task}} + \lambda_{\text{intervention}}\mathcal{L}_{\text{intervention}}$$

Which is then used to update both the CEM and the intervention policy model.

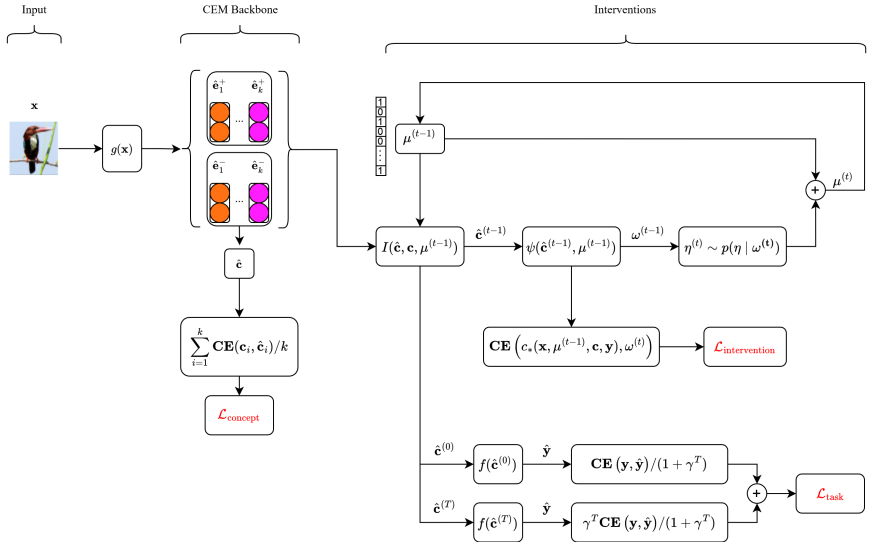

Figure 2.3: The training loop of IntCEM. $\boldsymbol{\mu}$ is the intervention binary mask, η the next concept to intervene on, and ψ the intervention policy model. The loop computes concept loss $\mathcal{L}_{\text{concept}}$, intervention loss $\mathcal{L}_{\text{intervention}}$, and task loss $\mathcal{L}_{\text{task}}$ which is used to update the IntCEM.

Despite IntCEM's impressive intervention performance, we note that it learns a greedy intervention policy. That is, its learnt policy tries to maximize the post-intervention accuracy at each step, by mimicking the behaviour of a greedy optimal policy. We propose that non-greedy intervention policies can outperform greedy intervention policies for each budget as the optimal intervention sequence can differ when the budget varies. If we were to adopt a similar approach by mimicking the behavior of a non-greedy optimal policy, at each step we would have to search through $O(k!)$ combinations of concepts to find the optimal set of concepts to intervene on. This is much larger than the $O(k)$ searches required by the optimal greedy policy, and thus this approach is infeasible due to the high time complexity. As such, we cannot adopt a similar approach to answer our research question. This prompts us to use Reinforcement Learning as an alternative approach.

2.4 RL

Reinforcement Learning [19] focuses on training agents to make sequential decisions in an environment. where the goal is to maximize the cumulative rewards received by the agent by taking actions over time. The agent has access to observations which reflect the current state of the environment and can take actions to progress to different states until termination.

In this project, we utililze Reinforcement Learning to learn a non-greedy policy to decide which concepts to intervene. We select the Proximal Policy Optimization [16] algorithm to do this, a SOTA policy-based RL algorithm that combines the strengths of traditional policy-based [15] approaches and value-based approaches [11, 12]. PPO consists of an Actor model θ and a Critic model ϕ. The Actor model learns a policy π_θ and decides on what actions to take based on the current state. The Critic model learns to estimate the value $V_\phi(s)$ of a current state, which is the expected future discounted rewards. This gives us the next state value function Q_ϕ based on rewards and the approximation from V_ϕ, which is the value of taking an action

$$Q_\phi(s, a) = r + \gamma_{RL} V_\phi(s')$$

equal to the reward r and value of the next state $V_\phi(s')$ discounted by a factor of γ_{RL}. We can then compute an advantage function $A_\phi(s, a) = Q_\phi(s, a) - V_\phi(s)$ that estimates the advantage of taking an action.

During training, we sample states s_i, actions a_i from the Agent and their corresponding rewards from the environment. This allows us to compute the next state value function, then used to compute a value loss $\mathcal{L}_{\text{value}}$ for the Critic model to minimize the discrepancy between the estimated and true values, using the following equation

$$\mathcal{L}_{\text{value}} = \frac{1}{k} \sum_{i=0}^{k} \nabla_\phi (V_\phi(s_i) - Q_\phi(s_i, a_i))$$

We can also compute a policy loss $\mathcal{L}_{\text{value}}$ for the Actor model to update it in the direction that maximizes the advantages computed using the Values estimated by the Critic model

$$\mathcal{L}_{\text{policy}} = -\frac{1}{k} \sum_{i=0}^{k} \nabla_\theta \log \pi_\theta(a_i \mid s_i) \cdot A_\phi(s_i, a_i)$$

Since the Actor model maximizes the advantages based on the Value function estimated by the Critic model, this balances between exploration and exploitation, as initially the Actor model explores states which may have low true values which are not learnt by the Critic model, and as the Critic model learns to estimate the values of states better, the

Actor model also learns a better policy. Apart from the Actor and Critic models, PPO also utilizes a clipping function to prevent the policy update to be too large or too small. Section 3.3.1 goes into how the PPO algorithm is used to train an RL agent that learns a non-greedy intervention policy.

2.5 Flow Models

Flow models model probability distributions by leveraging the change of variable property [21]. Given an input random variable X, if we can define an invertible transformation function f such that $f(X) = Z$ and $f^{-1}(Z) = X$ for another random variable Z, the change of variable property says that the probability densities of the two random variables p_X and p_Z, are related in the following way:

$$p_X(x) = \left| \det \frac{df^{-1}(x)}{dx} \right| p_Z(f^{-1}(x)) \tag{2.1}$$

using the Jacobian determinant of the inverse of f.

Flow models define transformations using ML models with this property with learnable parameters. These transformations are designed to be easily invertible and which the Jacobian determinant is simple to calculate. These transformations can be composed by sequentially applying them, and the probability distribution can also be found by sequentially applying Equation 2.1.

During training, a latent distribution is chosen which is usually one with a probability density that is simple to sample from. These transformations, which when composed can be used to transform the simple latent distribution a more complex distribution, are then used to model the complex data distribution, such as the distribution of concepts within a dataset for CEMs. This allows us to model and sample from distributions which would otherwise be difficult to sample from. Section 3.2 explains how we utilize a variation of these normalizing flow models to model the conditional distribution of concepts, which is then used to provide intermediate rewards for the RL agent to learn a non-greedy intervention policy.

2.6 Related Work

2.6.1 Expected Change in Target Prediction

Expected Change in Target Prediction (ECTP) [18] is a greedy intervention policy that selects at each step, the concept which when intervened, leads to the largest change in the probability of the currently predicted class. This is because this concept is the most "important", and mis-predicting this concept will result in the largest increase in task

error. This can be summarized as a score, for predicted concepts $\hat{\mathbf{c}}$ and $\hat{\mathbf{y}}$, the importance score of a concept c_i is given by

$$\mathbb{E}_{v \sim p_g(c_i|\mathbf{x})}[p_f(\hat{\mathbf{y}} \mid c_i = v, c_{\setminus\{i\}})] - p_f(\hat{\mathbf{y}} \mid c_i, c_{\setminus\{i\}})$$

taking over expectation of the distribution of values of c_i which takes over expectation of the distribution of values it can change to, using the probabilities computed by the concept predictor model g and label predictor model f.

ECTP is a greedy intervention policy which selects the concept with the highest importance to intervene on, and thus does not answer our research question.

2.6.2 CooP

Cooperative Prediction (CooP) [3] further builds on the idea of ECTP and utilizes uncertainty as an additional metric for cooperative prediction of concepts to intervene. CooP is a greedy intervention policy that uses a score function, selecting concepts with the highest score at each step to intervene. Its score function consists of a combination of the concept prediction uncertainty, the concept importance score and acquisition cost. The concept prediction uncertainty for a concept c_i and a given input $\mathbf{x}$ is calculated by the entropy of the distribution

$$H[p_\theta(c_i \mid x)]$$

which measures the expected information gain.

It uses a weighted sum of this information gain score, the concept importance score mentioned in Section 2.6.1, and the acquisition cost, which are the three factors believed that makes a concept optimal for interventions . For simplicity we assume that all concepts have the same acquisition cost.

However, this does not answer our research question as

1. It is a heuristic- based intervention policy, and we aim to find a learnt intervention policy which has been shown to have better performance on different datasets.

2. It is a greedy intervention policy. We think that non-greedy policies can outperform greedy ones, thus we believe there are better approaches that we aim to investigate.

As shown in Section 3.2, we also use an uncertainty-based idea to guide the Reinforcement Learning agent. We utilise the expected information gain to the target variable as intermediate rewards to the RL agent, with the goal of guiding it to perform interventions that will maximize the information gain, reducing uncertainty and learning a non-greedy intervention policy model that maximizes the post-intervention predictive accuracy. We compare the performance of our approach against CooP in Section 4.4

2.6.3 Active Feature Acquisition

Active Feature Acquisition (AFA) [9] is the problem of deciding what features to acquire from the environment, dynamically based on their expected utility and acquisition cost, in order to improve predictive performance. Compared to CBMs and interventions, AFA focuses on the trade-off between the benefit and costs of acquiring concepts, specifically cost-efficiency, whereas CBMs focus on predicting human-interpretable concepts from input. While these two focus on different field and aspects of Machine Learning, we can draw a parallel between interventions in CBMs and acquiring features in AFA. Intervening on concepts is similar to acquiring features in AFA, where this allows us to learn about the true values of them, and the two both have the same goal of maximizing predictive performance. However, our problem of interventions is more complex as we have to take into account the original prediction by the CBM and budgets for interventions. Reinforcement Learning has been applied to AFA to learn a non-greedy policy for acquiring features, and has shown impressive results in determining good feature acquisition policies. In Section 3.3 we look into how we can apply ideas of using RL in AFA to learn a non-greedy intervention policy.

In this chapter, we have explained the background for this project, as well as why existing research does not answer our research question. In the next section we look at our proposed solution and how it answers our research question.

Chapter 3

Method

In this chapter, we document how we propose to learn a non-greedy intervention policy. We first formalize interventions and intervention policies, then introduce the surrogate models we use to calculate intermediate rewards, and finally, present how Reinforcement Learning is used to learn a non-greedy intervention policy.

Building on top of IntCEMs, which augment a CEM with a greedy intervention policy model and train the two simultaneously, we propose RLCEM, a novel CEM model augmented with a Reinforcement Learning agent that learns a non-greedy intervention policy. The two are trained simultaneously, similar to IntCEM, where the sampled interventions are used to increase the sensitivity of the $\mathbf{c} \rightarrow \mathbf{y}$ label prediction model.

An overview of the general structure of our RLCEM is in Figure 3.1. The RL agent learns to output interventions, which are used to train the CEM to increase sensitivity of CEM to interventions. After all interventions from the RL agent, the CEM prediction is used to calculate the final reward to train RL agent. Additionally, the surrogate model is used to calculate intermediate rewards which are used to train and guide the RL agent to make more optimal intermediate interventions. We select Arbitrary Conditional Flow (AC Flow) models [8] as our surrogate models, and use Proximal Policy Optimization [16] to train our RL agent.

3.1 Interventions

A key advantage of using CBMs is having access to run-time interventions, which is the idea of utilizing professionals to modify incorrect concept predictions to improve the performance of the model. For simplicity, we do not consider incorrect interventions, i.e. when the professionals misjudge and modify the predicted concepts to incorrect values, and assume that all interventions are correct. To formalize interventions, we define them via the following function, where the predicted concepts $\hat{\mathbf{c}}$ and the true concepts $\mathbf{c}$ are interpolated using a multi-hot encoding intervention vector $\boldsymbol{\mu}$.

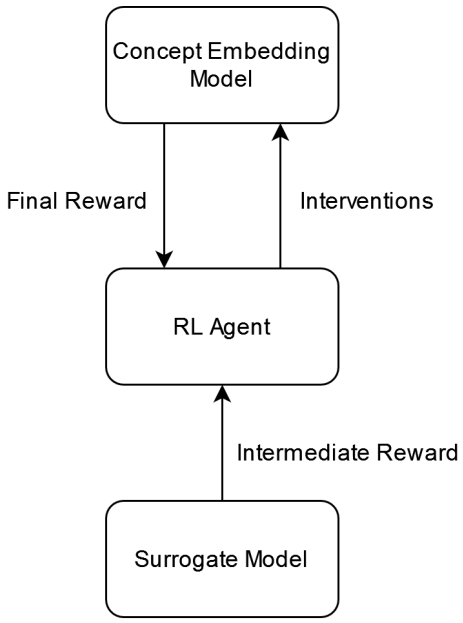

Figure 3.1: An overview of the structure of RLCEM. The RL agent samples interventions for the CEM, which is used to calculate the final reward to train the agent. The surrogate model is used to calculate intermediate rewards to train the agent.

$$I(\hat{\mathbf{c}}, \mathbf{c}, \boldsymbol{\mu}) = \boldsymbol{\mu}\, \mathbf{c} + (1 - \boldsymbol{\mu})\, \hat{\mathbf{c}} \qquad \hat{\mathbf{c}}, \mathbf{c}, \boldsymbol{\mu} \in \{0, 1\}^k$$

Figure 3.2 demonstrates how an intervention is formed using a binary mask $\boldsymbol{\mu}$ from the predicted concepts $\hat{\mathbf{c}}$ and the true concepts $\mathbf{c}$.

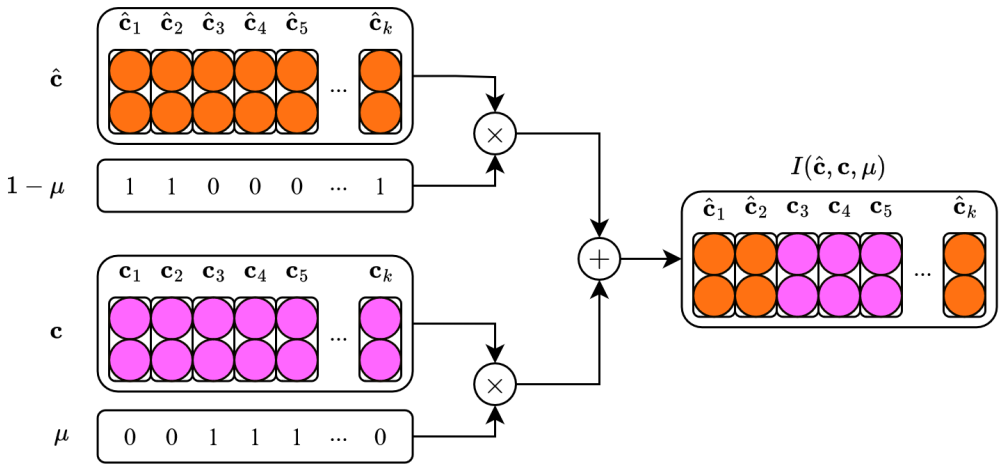

Figure 3.2: An example of how interventions are formed from the predicted concepts $\hat{\mathbf{c}}$ and true concepts $\mathbf{c}$ using binary mask $\boldsymbol{\mu}$.

3.1.1 Intervention Policies

An intervention policy $\mathcal{P}$ determines the order of concepts to intervene on with the goal of maximizing the accuracy of the $\mathbf{c} \rightarrow \mathbf{y}$ concept prediction model. A greedy intervention policy is thus a collection of functions $\mathcal{P}_i$, each of which outputs the concept to intervene

on at step i. An optimal greedy policy is the following

$$\hat{\mathcal{P}} = \bigcup_{i=1}^{k} \underset{\mathcal{P}_i}{\operatorname{argmax}}\, Acc(\hat{g}(\hat{\mathbf{c}}_{\mathcal{P}_j}), \mathbf{y})$$

$$\hat{\mathbf{c}}_{\mathcal{P}_0} = \hat{\mathbf{c}}, \hat{\mathbf{c}}_{\mathcal{P}_j} = I(\hat{\mathbf{c}}_{\mathcal{P}_{j-1}}, \mathbf{c}, \mathcal{P}_j(\hat{\mathbf{c}}_{\mathcal{P}_{j-1}}))$$

Which maximizes the accuracy at each step j sequentially for all k concepts. At each step, $\hat{\mathbf{c}}_{j-1}$ is the predicted concept after the previous $j-1$ interventions, and we aim to maximize the accuracy of the prediction made by the $\hat{g} : \mathbf{c} \to \mathbf{y}$ model.

Compared to a greedy intervention policy, a non-greedy intervention policy outputs a set of concepts to intervene on for a budget j, which we want to maximize the accuracy of the $\mathbf{c} \to \mathbf{y}$ model on. An optimal non-greedy policy maximizes the following

$$\hat{\mathcal{P}} = \underset{\mathcal{P}}{\operatorname{argmax}} \sum_{j=1}^{k} Acc(\hat{g}(\hat{\mathbf{c}}_{\mathcal{P}_j}), \mathbf{y})$$

$$\hat{\mathbf{c}}_{\mathcal{P}_j} = I(\hat{\mathbf{c}}, \mathbf{c}, \mathcal{P}(\hat{\mathbf{c}}, j))$$

Note that the notion of a budget, defined as the number of concepts the model is allowed to intervene on for simplicity, is only meaningful for non-greedy policies. Non-greedy policies aim to maximize the accuracy of the $\mathbf{c} \to \mathbf{y}$ model after using up the intervention budget, and may select different sets of intervention concepts for different budgets. Greedy policies always select the same concepts per step and thus the budget does not affect the concept selected by the policy. This is why existing intervention policies do not consider budgets, and why we consider them in our research question.

3.2 Surrogate Models

We use generative surrogate models that models the likelihoods of concepts to guide the RL model. Such a surrogate model is needed because when we are learning a non-greedy policy, its performance is measured by the accuracy of the model after all interventions in the end. When training an RL agent to learn such a policy, this corresponds to rewarding it at the final step based on the final accuracy, and training it to maximize this reward. However, previous studies have shown that using delayed rewards, where the agent is only rewarded after long episodes, can pose a challenge to its learning as the agent struggles to learn the consequences of its actions [10, 20], and thus struggles to learn to make the correct decisions. Therefore we introduce intermediate rewards to guide the RL agent to make the correct intermediate interventions that lead to the largest final accuracy.

Following Li et al. [9] in Active Feature Acquisition, we use a surrogate model to approximate the conditional likelihoods $p(\mathcal{C}_u \mid \mathcal{C}_o, \mathbf{y})$, where $\mathcal{C}_u$ is a set of un-intervened concepts,

$\mathcal{C}_o$ a set of intervened concepts, and $\mathbf{y}$ the label. In the context of an intervention, the previously intervened concepts would be $\mathcal{C}_o$, and the current concept we are intervening on would be $\mathcal{C}_u$.

We use a variant of the popular normalizing flow models [21], namely Arbitrary Conditional Flow (AC Flow) [8] models as our surrogate models. These models are flow models augmented to model arbitrary conditional likelihoods $p(\mathcal{C}_u \mid \mathcal{C}_o)$ between sets of variables $\mathcal{C}_u$ and $\mathcal{C}_o$. We extend these models to approximate the class-conditional likelihoods $p(\mathcal{C}_u \mid \mathcal{C}_o, \mathbf{y})$. To represent these sets of concept, we use concept vectors $\rfloor$ along with binary masks b and m. The mask b represents the already intervened concepts, the mask m represents the concepts we are interested in ($\mathcal{C}_u$ and $\mathcal{C}_o$). Mathematically this gives us two concept vectors

$$\mathbf{c}_o = \mathbf{c} \cdot b$$

$$\mathbf{c}_u = \mathbf{c} \cdot (1 - b) \cdot m$$

For convenience, we use $\mathbf{c}_u$ and $\mathbf{c}_o$ from now on to represent the sets of concepts $\mathcal{C}_u$ and $\mathcal{C}_o$, with $c_i \in \mathbf{c}_u, \mathbf{c}_o$ representing the individual concepts.

3.2.1 Latent Distribution

As described in Section 2.5, normalizing flow models utilize the change of variable formula to model likelihoods, which can be extended to include conditional likelihoods. AC Flow models build on Transformation Autoregressive Networks (TANs) [13], and learn to model likelihoods $p(c_0, \ldots, c_n \mid \mathbf{c}_o, \mathbf{y})$, modelling the underlying latent distribution using an autoregressive approach with Recurrent Neural Networks (RNNs) [17]. The RNN learns to model the likelihood of $p(z_0, \ldots, z_n \mid \mathbf{z}_o, \mathbf{y})$ by sequentially processing each of the variables z_i. At each step i, an RNN outputs parameters for an underlying Gaussian Mixture Model (GMM), which is a mixture of K different Gaussian distributions [14]. This allows us to compute $p(z_0, \ldots, z_i \mid \mathbf{z}_o, \mathbf{y})$, the likelihood of the current i variables, using a weighted sum of the probability density of the Gaussian distributions. Experimentally we find that setting the number of components K to be the number of classes of $\mathbf{y}$, equivalent to one Gaussian distribution per class achieves a good balance between model performance and computational efficiency. While using a GMM does not directly gives us the probability of $p(z_0, \ldots, z_n \mid \mathbf{z}_o, \mathbf{y})$, it allows us to compute the probability density which tells us the likelihood of the variables $z_0, \ldots, z_n$, which is proportional to the actual probability and also allows us to sample from the distribution.

3.2.2 Transformations

In order to transform latent likelihoods $p(z_0, \ldots, z_n \mid \mathbf{z}_o, \mathbf{y})$ to the concept likelihoods $p(c_0, \ldots, c_n \mid \mathbf{c}_o, \mathbf{y})$, we utilize a set of transformations with learnable parameters that map input concepts c_i to latent variables z_i. We follow the set of conditional transformations defined by Li et al. [8], and extend them to be label $\mathbf{y}$ specific. These transformations

$q_{\mathbf{c}_o,b,\mathbf{y}}$ are conditional on intervened concepts $\mathbf{c}_o$, binary mask b, and label $\mathbf{y}$, and are invertible so that we can obtain the liklihood and sample from the latent distribution. For all un-intervened concepts $\mathbf{c}_u$, the transformation maps these concepts to latent variables $q_{\mathbf{c}_o,b,\mathbf{y}}(\mathbf{c}_u) = \mathbf{z}_u$. We can then apply the change of variable theorem with a conditional extension, and using the Jacobian determinant tells us how the likelihood changes [21].

$$p(\mathbf{c}_u \mid \mathbf{c}_o, b, y) = \left|\det \frac{dq_{\mathbf{c}_o,b,\mathbf{y}}}{d\mathbf{c}_u}\right| p(q_{\mathbf{c}_o,b,\mathbf{y}}(\mathbf{c}_u) \mid \mathbf{c}_o, b, y)$$

In the AC Flow model, we mainly leverage linear transformations. We use Multi-Layer Perceptrons (MLP) [28] ϕ to learn a weight matrix $\mathbf{W}$ and bias vector $\mathbf{t}$

$$\mathbf{W}, \mathbf{t} = \phi(\mathbf{c}_o, b, \mathbf{y})$$

As shown in Figure 3.3, this gives us a linear transformation for all possible concepts, and by indexing

$$\mathbf{W}_u = W[1-b][1-b], \mathbf{b}_u = \mathbf{b}[1-b]$$

to select the entries corresponding to the un-intervened concepts, we obtain

$$\mathbf{z}_u = \mathbf{W}_u \mathbf{c}_u + \mathbf{b}_u$$

This transformation is straightforward to invert, and we can find the Jacobian determinant easily and apply the change of variable theorem illustrated above to get the likelihood $p(\mathbf{c}_u \mid y)$. Other variants, such as using an RNN to learn a linear transformation for each , are also included to add flexibility to the transformations such that we are able to model the input data distribution. More details on the transformations we used can be found in Appendix B.1.

These tranformations and the latent distribution are both learnt to approximate the conditional likelihoods $p(c_0, \ldots, c_n \mid \mathbf{y})$ or the likelihood of seeing a set of concepts $p(\mathbf{c}_u \mid \mathbf{y})$. By using Bayes' theorem, we note that we can additionally model the conditional likelihoods of the label and un-intervened concepts

$$p(\mathbf{y} \mid \mathbf{c}_o) = \frac{p(\mathbf{c}_o \mid \mathbf{y})P(\mathbf{y})}{\sum_{y'} p(\mathbf{c}_o \mid \mathbf{y}')P(\mathbf{y}')} \tag{3.1}$$

Which the right hand side terms can be computed using the AC Flow model, with $p(\mathbf{c}_o \mid \mathbf{y}) = p(\mathbf{c}_o \mid \emptyset, \mathbf{y})$, where we use an empty set of concepts as the condition. As we can see in Section 3.3, this likelihood is used to provide rewards to the RL agvent.

Since the transformations learnt are invertible, we can also sample from the data distribution via the underlying distribution. This is useful for the RL agent to determine interventions as this provides information on which concepts are likely to be present (or

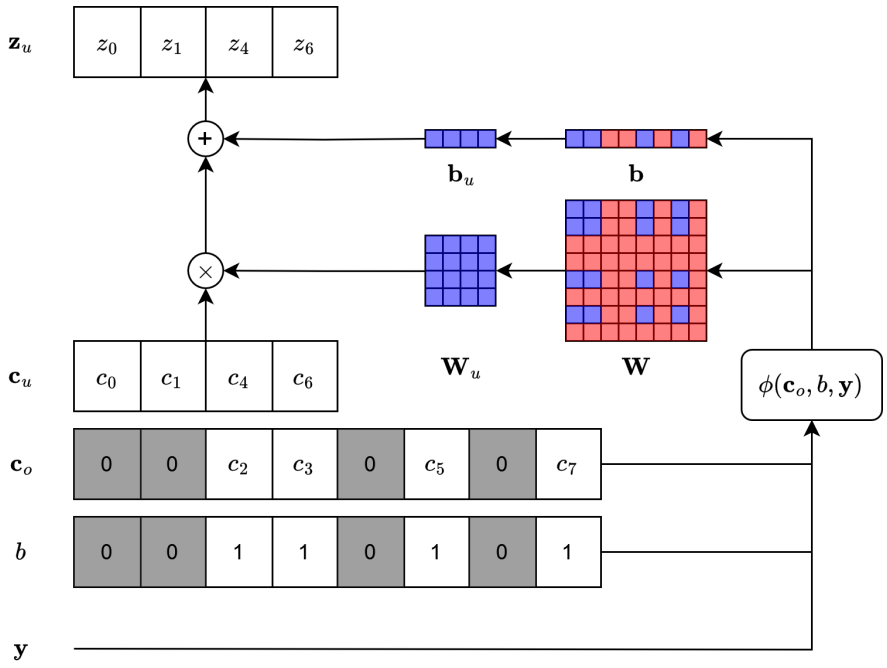

Figure 3.3: An example of linear transformation.

not present) given the currently intervened concepts.

In general, we follow the advised hyperparameters from Li et al. [8] for building the transformations used in our AC Flow model. This includes the rank of the linear matrices $\mathbf{W}$ and the hidden dimension of the latent distribution RNN.

3.2.3 Training the Surrogate Models

In this project, we adapt AC Flow models to model the likelihoods of concepts in CEMs during interventions. We follow the description of Li et al. [9] and implement corresponding AC Flow models to model the distribution of concepts.

We train the AC Flow models using a combination of two losses, a negative log likelihood loss and a mean-squared error (MSE) loss . Since these models output likelihoods directly, we can directly maximize the likelihood, or equivalently minimizing the negative log likelihood. Additionally, a cross entropy loss is incorporated to help the model learn the conditional likelihood with respect to the label y. Given $\mathbf{c}_u, \mathbf{c}_o, y$, we compute the class with the highest likelihood $\hat{y} = \text{argmax}_y \, p(\mathbf{c}_u, \mathbf{c}_o \mid y)$ and compute a cross entropy loss $CE(\hat{y}, y)$ such that the model learns to output higher likelihoods for concepts that belong to the correct class. The loss thus becomes

$$Loss = -\log p(\mathbf{c}_u \mid \mathbf{c}_o, \mathbf{y}) + CE(\underset{\mathbf{y}}{\text{argmax}}\, p(\mathbf{c}_u, \mathbf{c}_o \mid \mathbf{y}), \mathbf{y})$$

Additionally, we add a penalty term to the loss to prevent the model from outputting large values of likelihoods in general. To penalize large values, similar to an L2 regularization loss, we add the square of the logits $\log p(\mathbf{c}_u, \mathbf{c}_o \mid \mathbf{y})$ and $\log p(\mathbf{c}_o \mid \mathbf{y})$ to the loss. This

penalty term is further discussed in Section 4.3. The final loss for training the AC Flow model is

$$\begin{aligned} Loss = &- \log p(\mathbf{c}_u \mid \mathbf{c}_o, \mathbf{y}) + CE(\underset{\mathbf{y}}{\operatorname{argmax}}\, p(\mathbf{c}_u, \mathbf{c}_o \mid \mathbf{y}), \mathbf{y}) \\ &+ \lambda_{l2} \left(\log^2 p(\mathbf{c}_u, \mathbf{c}_o \mid \mathbf{y}) + \log^2 p(\mathbf{c}_o \mid \mathbf{y}) \right) \end{aligned}$$

3.3 RLCEM

Instead of using the original CBMs as our base models, we adopt CEMs as they have been shown to have better performance with and without interventions compared to CBMs [26]. Additionally, CEMs are more robust to concept-incompleteness, which is when the concepts present in the dataset annotations do not contain all possible concepts that can be used to, an important issue in real-life datasets. This is because CEMs learn to output intermediate concept embeddings rather than binary values [26].

We construct RLCEMs by augmenting CEMs with an RL agent as described below, which is our proposed solution to our main research question.

3.3.1 Reinforcement Learning

We model the problem of finding a non-greedy intervention policy as a Reinforcement Learning problem. As mentioned in Section 2.4, Reinforcement Learning is used to find non-greedy solutions to problems by design as it models the long-term effects of its actions, and aims to maximize the overall reward gain.

In order to formulate the problem as a Reinforcement Learning problem, we model the problem of deciding the concepts to intervene on as a Markov Decision Problem [23].

States comprise of the information available at each step. This contains the remaining budget and state of the CEM, including its bottleneck and predicted concepts. This also contains the output of the surrogate model, including the sampled values for the un-intervened concepts $\bar{\mathbf{c}_u} \sim p(\mathbf{c}_u \mid \mathbf{c}_o, \mathbf{y})$, $\bar{\mathbf{c}_u} \sim p(\mathbf{c}_u \mid \mathbf{c}_o)$ from the class-conditional and marginalized likelihoods as described in Section 3.2.

Actions correspond to performing interventions on the un-intervened concepts. For simplicity, we group concepts into groups, and each action intervenes on all concepts within a group.

Rewards are the increase in information, calculated using entropy, to the target y [9] given by

$$\text{Reward} = H(y \mid \mathbf{c}_o) - \mathbb{E}_{p(\mathbf{c}_u \mid \mathbf{c}_o)}[H(y \mid x_u, x_o)]$$

This is because we want to encourage the RL agent to make interventions that provide more information to the target variable, or reduce uncertainty, an idea that has shown to

work well in CooP [3]. Simplifying gives us

$$\text{Reward} = H(\mathbf{c}_u \mid \mathbf{c}_o) - \mathbb{E}_{p(\mathbf{y}|\mathbf{c}_o)}[H(\mathbf{c}_u \mid y, \mathbf{c}_o)]$$

which can directly be calculated from the output of the surrogate model using Equation 3.1. The final reward when the sequence terminates should reflect how well the prediction post-intervention is. It is calculated as negative of the Cross Entropy Loss, where a higher value corresponds to a lower discrepancy between the predicted label and the true value. As this requires knowledge of the true label, this reward is only computed during training to train the RL agent. This is also the reason why Reinforcement Learning is unable to learn a greedy intervention policy as this would require knowledge of the ground truth label at each step to compute the intermediate rewards. Li et al. [9] also show that using this intermediate reward will not affect the optimality of the learnt policy. Using these two rewards results in an RL agent that learns to balance between making interventions that increase the information to the target variable, which should lead to better final accuracy, and other types of interventions that also result in a higher final accuracy.

Termination is when the remaining budget is insufficient for more interventions.

To model budgets and acquisition costs, we can either

1. Incorporate the acquisition costs for each concept into the reward by subtracting the corresponding cost. The RL agent learns to balance automatically intervening on concepts that are more costly versus the potential extra reward gained by the increase in information and accuracy. The RL agent then determines when to terminate if the cost of intervening on new concepts outweight the potential gain.

2. Or leave it out of the reward, such that the reward only contains the information gain in each step. We manually step to the termination step if and only if the the remaining budget does not allow for future interventions.

The first approach involves balancing the weights between the intervention costs and reward, and does not adopt to the idea of budgets very well, and we select the second approach. To further simplify, we assume that all concept groups have the same cost to intervene on. Thus we set the cost of each intervention to be 1, and the budget becomes the number of possible interventions.

3.3.2 Training the agent

In Active Feature Acquisition as described in Section 2.6.3, Li et al. [9] combine Reinforcement Learning with AC Flow models to find the optimal features to acquire from the environment. We adopt a similar approach to train the RL agent.

We first pre-train an AC Flow model that learns arbitrary conditional likelihoods about the underlying concepts $p(\mathbf{c}_u \mid \mathbf{c}_o, y)$. Then, A Reinforcement Learning agent is trained to

maximize the reward given in Section 3.3.1. At each step, the agent is given the current intervened concepts $\mathbf{c}_o$, and then the agent samples the next concepts to intervene $\mathbf{c}_u$, where the agent is rewarded based on the expected information gain to the target variable. At each step, the agent has access to the sampled un-intervened concepts $\hat{\mathbf{c}_u}$ from the AC Flow model. This allows the RL agent to learn to intervene on concepts that are more likely incorrectly predicted.

Compared to Active Feature Acquisition, the problem setting is a lot more complex as rather than simply acquiring features from the environment, we are trying to determine which concepts are more likely to be incorrectly predicted by the $\mathbf{x} \rightarrow \mathbf{c}$ model, and which concepts are more likely to, when corrected, guide the model towards the correct prediction $\mathbf{y}$. Additionally the goal is to train one RL agent to be able to determine which concepts to intervene on for different budgets, which adds another layer of complexity as we require one unified model for the different tasks with different budgets.

3.3.3 Reinforcement Learning Algorithm

As mentioned in Section 2.4, the state-of-the-art RL algorithm for learning a policy is Proximal Policy Optimization (PPO) [16], which we use to train a RL agent that learns a non-greedy intervention policy. PPO utilizes a Critic model to estimate the value of a particular state, which is the discounted expected future rewards. Then an Actor model learns a policy for taking actions that lead to states with higher values as estimated by the Critic model.

During training, a state and its true value is computed and compared to the estimated value by the Critic, and a value loss is computed to minimize the discrepancy between these two values. Then a policy loss is computed by the discrepancy between actions selected for states and the estimated change in value by taking that action. Ultimately, the Critic model should be able to estimate the value of a state which reflects its future label prediction accuracy after all interventions, and the Actor model should be able to estimate the optimal policy, which is the interventions that lead to the highest label prediction accuracy after all interventions.

3.3.4 Combining RL with CEM

Zarlenga et al. [27] showed that combining learning an intervention policy and learning a CEM as a joint optimization problem achieved the best intervention performance. Not only do we learn an intervention policy specific to a CEM and task, the CEM also learns to be more sensitive to interventions, achieving higher label prediction accuracy when concepts are intervened. Thus we also combine the training of the RL agent and the CEM as a joint optimization problem, and train both simultaneously in one training loop. As shown in Figure 3.4, the training loop of the RLCEM is as follows:

1. The $\mathbf{x} \rightarrow \mathbf{c}$ model first computes the predicted concepts $\mathbf{c}$. We then compute a

concept loss $\mathcal{L}_{\text{concept}}$ with the true concepts. We also compute a task loss $\mathcal{L}_{\text{task}}$ with the true labels which is then back-propagated through the $\mathbf{c} \rightarrow \mathbf{y}$ model.

2. Since we want to train the RL agent $\mathcal{A}$ to be able to learn the concepts to intervene for different budgets, thus during training for each mini-batch, we sample n_{rollout} different budgets.

   For each sampled budget T, in each step t the RL environment $\mathcal{E}$ provides the states $\mathbf{obs}^t$, the remaining budget $T - t$, and the previous reward r^{t-1} to the RL agent. The agent then samples actions $\hat{a}^{(t)}$, which is then used to update the intervention mask $\boldsymbol{\mu}^{t+1}$ and the predicted concepts $\hat{\mathbf{c}}^{t+1}$ in the next step.

3. After all interventions are performed, the states $\mathbf{obs}$, actions $\hat{a}$, the corresponding rewards r, are used to calculate the true values of states. This is then compared against the values estimated by the Critic model $\hat{V}(\mathbf{obs})$ to compute a value loss $\mathcal{L}_{\text{value}}$ for the Critic model. Then the estimated values $\hat{V}(\mathbf{obs})$ and the estimated next state values $\hat{Q}(\mathbf{obs}, \hat{a}, r)$ are used to compute the advantage function $\hat{A}$ which tells us the advantage of taking actions. These advantages are then used to compute a policy loss $\mathcal{L}_{\text{policy}}$ to update the Actor model. These two losses combine to give us the intervention loss $\mathcal{L}_{\text{intervention}}$.

4. Lastly, we compute task losses for both concepts before intervention and after intervention, discounted by γ^T, similar to IntCEM.

The structure of the training loop is similar to IntCEM. We compute a concept loss and a task loss without interventions. Then we sample interventions according to our intervention policy model, and compute task losses with respect to the intervened concepts to increase the CEM's sensitivty to interventions. Then we update our intervention policy model so that it makes better interventions which lead to a higher accuracy. Similar to IntCEM, final loss for training the model is given by

$$\mathcal{L} = \lambda_{\text{concept}}\mathcal{L}_{\text{concept}} + \lambda_{\text{task}}\mathcal{L}_{\text{task}} + \lambda_{\text{intervention}}\mathcal{L}_{\text{intervention}}$$

3.3.5 Limitations

A major concern is the Time Complexity associated with learning such an RLCEM. As mentioned in 3.3.1, in order to ensure that the RL agent learns a policy for a variety of different budgets, for k concepts and n concept groups, we sample $O(n)$ different policies for each mini-batch during training, set to $n/2$ in practice. For each budget, the RL agent needs to sample $O(n)$ actions and compute the corresponding rewards used for training. Compared to a greedy intervention policy which is trained sequentially over n possible interventions, a non-greedy intervention policy learnt using RL requires more complexity of $O(n^2)$ compared to $O(n)$. Additionally, since the underlying surrogate model utilises a sequential RNN to model the conditional distribution which has time complexity propor-

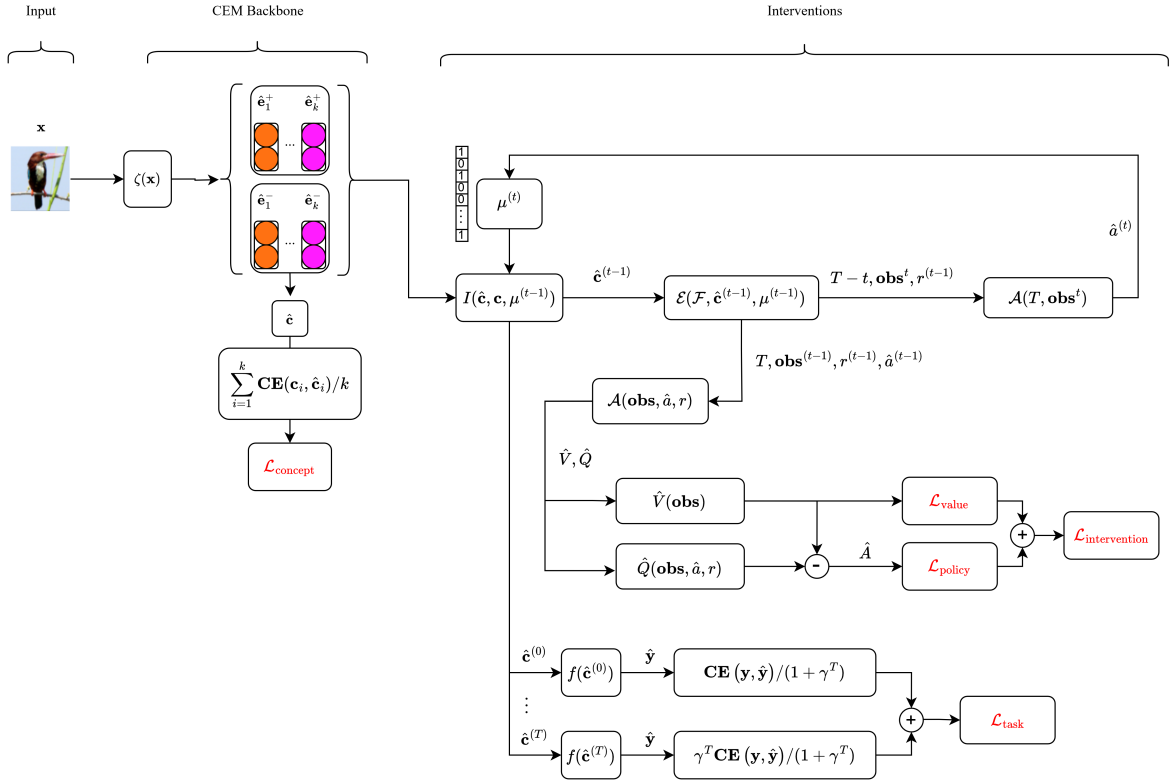

Figure 3.4: The training loop of RLCEM. T is the sampled budget, $\mathcal{E}$ is the RL environment, $\mathcal{A}$ the RL agent, and $\hat{V}, \hat{Q}$ the estimated value functions from the agent. The loop computes concept loss $\mathcal{L}_{\text{concept}}$, intervention loss $\mathcal{L}_{\text{intervention}}$, and task loss $\mathcal{L}_{\text{task}}$ which is used to update the RLCEM.

tional to the number of concepts $O(k)$, the final time complexity can reach up to $O(n^2k)$ compared to $O(n)$ of the current greedy intervention policy methods which adds a lot of cost to training. This limitation and its impacts are further discussed in Section 4.5.

In this chapter we presented a new method that utilises Reinforcement Learning to learn a non-greedy intervention policy, along with how we propose to train a generative AC Flow model to provide intermediate rewards to the RL agent during training and testing to guide its intervention decisions. This serves as a direct solution to our research question of developing a method to find non-greedy intervention policies. In the next chapter, we evaluate the performance of the proposed RLCEM against current methods on how they perform with and without interventions.

Chapter 4

Evaluation

In this chapter, we evaluate the performance of our RLCEM, and compare it against existing greedy intervention policies. RLCEM is a CEM augmented with an RL agent to learn a non-greedy intervention policy, and trained according to Section 3.3. We compare RLCEM to our baseline, including two of the known best-performing [27] greedy intervention policies IntCEM and CooP. IntCEM learns a greedy intervention policy by mimicking the behaviour of an optimal greedy policy, and CooP is a heuristic-based greedy intervention policy that selects concepts to intervene based on its associated uncertainty and importance score. As mentioned in Section 2.6.2, CooP uses a weighted average of three scores, and thus we perform a grid search for the weight of these three weights, and select the combination of weights with the best performance. We also compare RLCEM to Random, a policy that selects random concepts to intervene on.

All experiments in this chapter are conducted across three trials, where we report the mean and standard deviation. This helps make our results more statistically significant and reduces the errors that arise from randomness in Machine Learning experiments.

4.1 Models and Datasets

To evaluate the performance of our RLCEM model, we follow Zarlenga et al. [27] and select the datasets MNIST-ADD and CUB for our experiments. We use the same CEM sub-models for RLCEM and the baseline IntCEM to ensure the results reflect a fair comparison of the learnt intervention policy, which we follow from Zarlenga et al. [27]. For both datasets, 20% of the training dataset is selected as a validation dataset to monitor the performance of the model during training. This is used for hyperparameter selection and early stopping to prevent over-fitting. A summary of the two datasets used can be found in Table 4.1.

4.1.1 MNIST-ADD

MNIST-ADD is a dataset created from the MNIST [4] dataset. The MNIST dataset consists of hand-written digits from 0 to 9, which are black-and-white images with sizes $1 \times 28 \times 28$. MNIST-ADD samples 12 images from the MNIST dataset as input $\mathbf{x}$, with concepts corresponding to the values of each of the input images. To model concept-incompleteness in real life datasets [26], which is when the concept annotations do not contain all concepts relevant to predicting the label, we only select the concepts corresponding to 8 of the input images, ensuring that the same conceps are selected across all samples. This produces 54 concepts which are then grouped into 8 mutually-exclusive concept groups corresponding to images. The label for each MNIST-ADD sample is a binary label $\mathbf{y}$ corresponding to whether or not the sum of the input 12 images is greater than half of the possible maximum value. The MNIST-ADD dataset consists of 10,000 training samples and 10,000 test samples, formed by choosing each of the 12 input images randomly from the MNIST training and testing datasets respectively.

For this task, we use a ResNet-18 [5] backbone for the $\mathbf{x} \rightarrow \mathbf{c}$ model with its output linear layer modified to 54 activations, corresponding to the 54 concepts. The ResNet-18 backbone consists of 18 residual layers, each containing two convolutional layers with batch normalization and non-linear activation functions, and is a popular backbone for image-related tasks. The $\mathbf{c} \rightarrow \mathbf{y}$ model is an MLP with {128, 128} hidden layer activations, and forwards the 54 predicted concepts to predict the binary label. As it is a binary task, we use AUC instead of accuracy to measure the performance as it is less sensitive to class imbalances and classification thresholds.

4.1.2 CUB

CUB [24] is a real-life concept-annotated dataset that identifies birds. The input $\mathbf{x}$ consist of $3 \times 299 \times 299$ coloured images of birds, and output $\mathbf{y}$ is a label corresponding to the species of the bird. There are 112 concepts representing features of birds such as their colour, shape, etc., which are then grouped into 28 concept groups according to Zarlenga et al [27]. We further group these into 7 different concept groups for easy visualization and a better comparison with MNIST-ADD. This also allows us to use the same RL models across datasets as the size of the action space is simiar. There are 5,994 training samples and 5,794 testing samples.

For this dataset, we use a ResNet-34 [5] backbone for the $\mathbf{x} \rightarrow \mathbf{c}$ model with its output linear layer modified to 112 activations, corresponding to the 112 concepts. The ResNet-34 uses the same layers as ResNet-18 with more layers deigned to process larger and more complicated images, which we select due to the larger and more complex images in the CUB dataset. The $\mathbf{c} \rightarrow \mathbf{y}$ model is an MLP with {128, 128} as the hidden layer activations, and forwards the 112 predicted concepts to predict the label out of 200 classes.

Dataset	MNIST-ADD	CUB
Training Samples	10,000	5,994
Testing Samples	10,000	5,794
Input Size	[12,28,28]	[3,299,299]
Concepts (k)	54	112
Concept Groups (n)	8	7
Output classes	2	200

Table 4.1: The datasets and tasks used in this project.

More details on the datasets used can be found at Appendix A.

4.1.3 Reinforcement Learning agent Model

We adopt the same MLP for both the Actor and Critic model in the RL agent, using the same MLP structure as that in the IntCEM intervention policy model, but increase the number of activations in the hidden layers from {128, 128, 64, 64} to {512, 512, 256, 256} as our task of learning a non-greedy policy is more complex. The RL agent needs to learn $O(n^2)$ sub-tasks instead of $O(n)$, as there are $O(n)$ steps per budget for $O(n)$ different budgets. The same model is used for all datasets as the number of concept groups for intervention, which corresponds to the action space, is similar.

4.2 Non-greedy policies

Before evaluating the performance of RLCEM, it is important to test whether non-greedy policies can outperform greedy policies. Therefore we conduct an ablation study on the MNIST-ADD dataset, comparing two policies: GreedyOptimal and TrueOptimal. These are both the optimal policies that have access to the true label $\mathbf{y}$ during test, which allows them to search and select the concepts where intervening on these concepts yields a prediction $\hat{\mathbf{y}} = \hat{g}(\hat{\mathbf{c}})$ with the highest accuracy.

While GreedyOptimal searches for the best concept group out of all remaining un-intervened groups to select at each step, TrueOptimal searches through all possible combinations of concept groups to intervene at each budget, yielding a time complexity of $O(n!)$ rather than $O(n)$. This is also why while IntCEM can learn a greedy intervention policy model directly by training it to mimic the behaviour of a GreedyOptimal policy, it is infeasible to train a non-greedy intervention policy model to mimic the behaviour of a TrueOptimal policy.

GreedyOptimal and TrueOptimal represent the theoretical upper bound intervention performance that greedy and non-greedy policies can achieve respectively, and we compare

the two to see if the optimal non-greedy intervention policy can outperform the optimal greedy intervention policy. We train an IntCEM according to Zarlenga et al. [27], and evaluate the intervention performance when using GreedyOptimal and TrueOptimal.

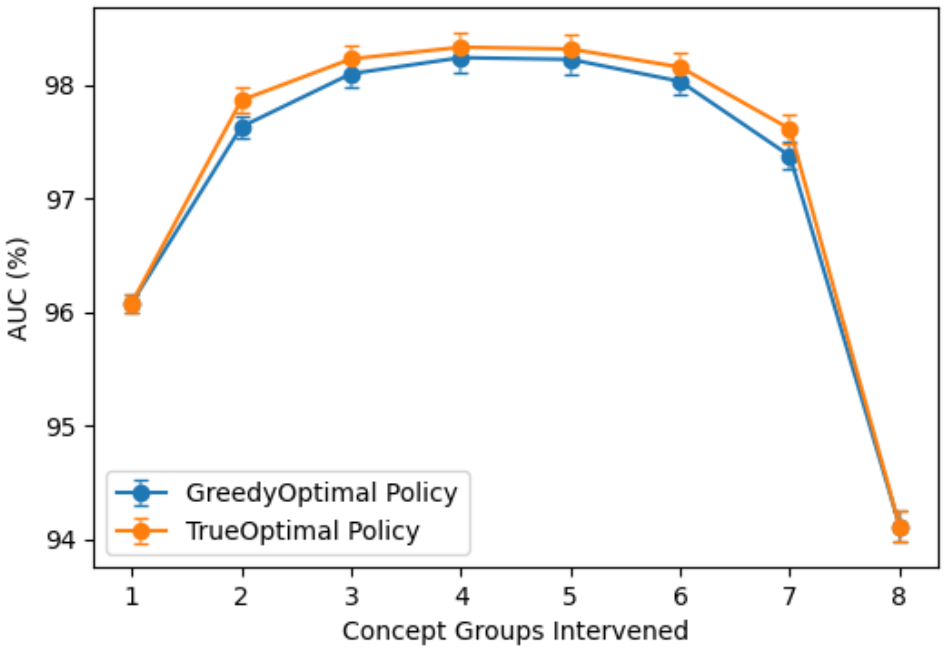

Figure 4.1: Test Intervention AUC (%) of GreedyOptimal and TrueOptimal on an IntCEM trained on MNIST-ADD. GreedyOptimal and TrueOptimal are the greedy and non-greedy optimal intervention policies respectively.

Figure 4.1 shows the test performance of GreedyOptimal and TrueOptimal on an IntCEM for the MNIST-ADD task. We see that TrueOptimal outperforms GreedyOptimal for all intervention groups, except when only 1 group is intervened and when 8 groups are intervened since the non-greedy and greedy optimal policies makes the same interventions for these two scenarios. The reason the performance when using fewer interventions (e.g. when 4 groups are intervened) is higher than when more groups are intervened (e.g. when all groups are intervened) is because the intervention policies have access to the true label, and can cherry-pick interventions that result in the prediction to align more with the true label. This is normal for a policy that has access to the true labels, and as we can see in Section 4.4, this behaviour is not observed in learnt policies such as IntCEM or RLCEM. Thus we have shown that non-greedy policies can outperform greedy policies as they have a higher upper-bound, and we move on to learning and evaluating non-greedy policies.

4.3 Surrogate Models

We then look into the training of the surrogate models, which are used to model the conditional likelihoods $p(\mathbf{c}_u \mid \mathbf{c}_o, \mathbf{y})$ and used to provide intermediate rewards to the RL agent. As mentioned in Section 3.2, we train the AC Flow model using a negative log

likelihood loss and a cross entropy loss, using random samples of $\mathbf{c}_o$ and $\mathbf{c}_u$ that are subsets of the overall concepts. The trained AC Flow model will be frozen and used in the RLCEM model to provide intermediate rewards to the RL agent.

To evaluate the performance of a trained AC Flow model, we look at its negative log likelihood and its accuracy on a validation set. This is the accuracy of predicting the label $\hat{\mathbf{y}}$ based on the class with the highest likelihood for concepts in the dataset, given by

$$\hat{\mathbf{y}} = \text{argmax}_{\mathbf{y}}\, p(\mathbf{y} \mid \mathbf{c}_o)$$

whereas the negative log likelihood (NLL) is the negative log likelihood of $p(\mathbf{c}_u \mid \mathbf{c}_o, \mathbf{y})$ for concepts $\mathbf{c}_u$ and $\mathbf{c}_o$ present in the validation dataset.

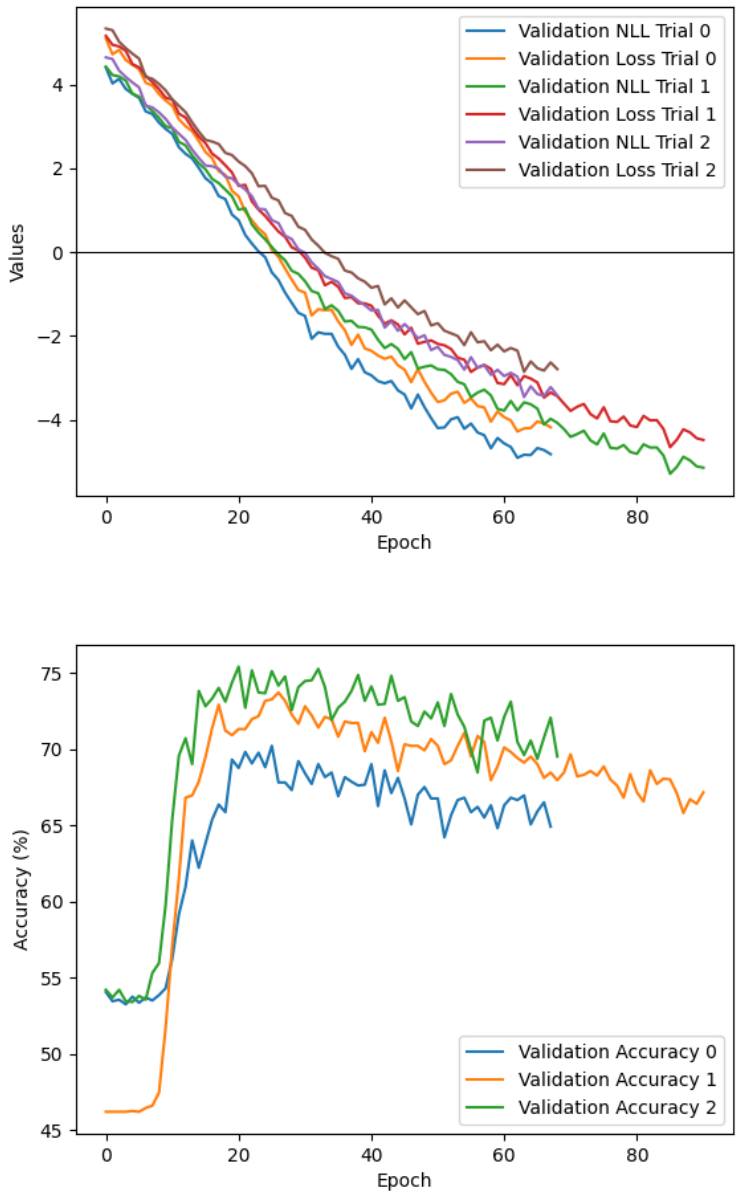

Figure 4.2: Performance on Validation Set when training an AC Flow model on the MNIST-ADD dataset across 3 trials. Top: Validation NLL and loss. Bottom: Validation Accuracy.

Figure 4.2 shows the validation results when training an AC Flow model on the MNIST-ADD dataset, using a learning-rate of 1e-6. A large issue that we observe is that the validation loss keeps decreasing and there is no limit to stop it. This is because the model learns to output the log likelihoods of $p(\mathbf{c}_u \mid \mathbf{c}_o, \mathbf{y})$, and the underlying model is a Gaussian Mixture Model which learns to output a probability density at each value. While these probability densities tells us about the likelihoods, they are not probability values within $[0, 1]$. Thus the log likelihood can fall outside of $[-\infty, 0]$, translating to a negative NLL and negative los. The model learns to output larger and larger values for the more probable likelihoods, and the NLL keeps decreasing below 0, as there is no upper limit to the values the likelihoods can take. At the same time the validation accuracy stops improving much quicker and then deteriorates, indicating that the model is over-fitting long before validation loss starts to converge. As a result, we use the validation accuracy as an early stopping metric to prevent over-fitting. This helps ensure that the model learns about the underlying distribution and the log likelihood values accurately reflect the likelihood of concepts, which is shown from the validation accuracy, while preventing the output log likelihood values from growing endlessly, which as we can see in Section 4.4, leads to other issues.

4.4 Evaluating RLCEM

We implement RLCEMs according to Section 3.3, implementing the Reinforcement Learning environment using the open-source Gymnasium [22] interface. This allows our RLCEM to be integrated with the Gymnasium RL ecosystem, and our implementation to be available to the research community for further research in RL-based intervention policies.

We train RLCEMs and evaluate them on the datasets MNIST-ADD and CUB, comparing them against three other intervention policies: IntCEM, CooP, and Random. While IntCEM is evaluated on the CEM learnt together with the IntCEM policy, we evaluate CooP and Random on the CEM learnt with the RL based policy. This is to ensure a fair comparison with the RLCEM policy.

4.4.1 Training Hyperparameters

The hyperparameters used for training IntCEM and an RLCEM are similar, consisting of the weights of the losses in the overall loss equation

$$\mathcal{L} = \lambda_{\text{concept}}\mathcal{L}_{\text{concept}} + \lambda_{\text{task}}\mathcal{L}_{\text{task}} + \lambda_{\text{intervention}}\mathcal{L}_{\text{intervention}}$$

The task loss discount factor for interventions γ, and other common hyperparameters for Machine Learning such as learning rate and batch size. We follow the recommended hyperparameters for IntCEM from Zarlenga et al. [27]. Due to similarities between IntCEM

and RLCEM, we re-use the same loss weights except $\lambda_{\text{intervention}}$, the weight of the loss used to train the intervention policy model. For RLCEM, we search over $\lambda_{\text{intervention}} \in [0.2, 1, 5]$, and select based on validation intervention performance. We train all models using the Adam [6] optimizer, with learning rates from [1e-3, 1e-4, 1e-5], and batch size to fit the hardware we use. We train for a maximum of 300 epochs, and stop training if the validation loss stops improving over 15 epochs to prevent over-fitting. A summary of the hyperparameters used is shown in Table 4.2.

Hyperparameter	IntCEM		RLCEM	
Task	MNIST-ADD	CUB	MNIST-ADD	CUB
Learning Rate	1e-5	1e-4	1e-5	1e-4
λ_{concept}	5		5	
λ_{task}	1		1	
$\lambda_{\text{intervention}}$	5		1	
γ	1.1		1.1	
Policy Model Activations	{128, 128, 64, 64}		{512, 512, 256, 256}	

Table 4.2: Training Hyperparameters used to train IntCEM and RLCEM for the two tasks MNIST-ADD and CUB.

4.4.2 MNIST-ADD

After training the AC Flow model on the MNIST-ADD dataset as illustrated in the previous section, we train RLCEM by using the pretrained AC Flow model to provide intermediate rewards equal to the increase in information to the target as mentioned in Section 3.3. The test intervention performance of RLCEM against existing intervention policies is shown in Figure 4.3. We see that barring the first and last intervention performance, where all non-greedy and greedy intervention policies perform similarly, RLCEM outperforms all of IntCEM, CooP and Random, yielding a higher AUC under the same number of interventions. When no interventions are performed, RLCEM has a similar performance to IntCEM, showing that the intervention performance increase does not come at the expense of un-intervened performance. This is expected as the base CEMs are trained in a similar fashion. Additionally, IntCEM outperforms Random at first but the performance of IntCEM degrades as interventions increase, also supports the hypothesis that greedy intervention policies may learn sub-optimal policies for different budgets, especially for larger budgets. RLCEM combines the strategy of directly learning a policy from IntCEM, and the uncertainty-based strategy of CooP, and learns an intervention policy that dynamically balances between them based on the budget.

We also perform a numerical analysis of the intervention performance, comparing the intervention performance across quartiles of interventions performed. Table 4.3 shows the

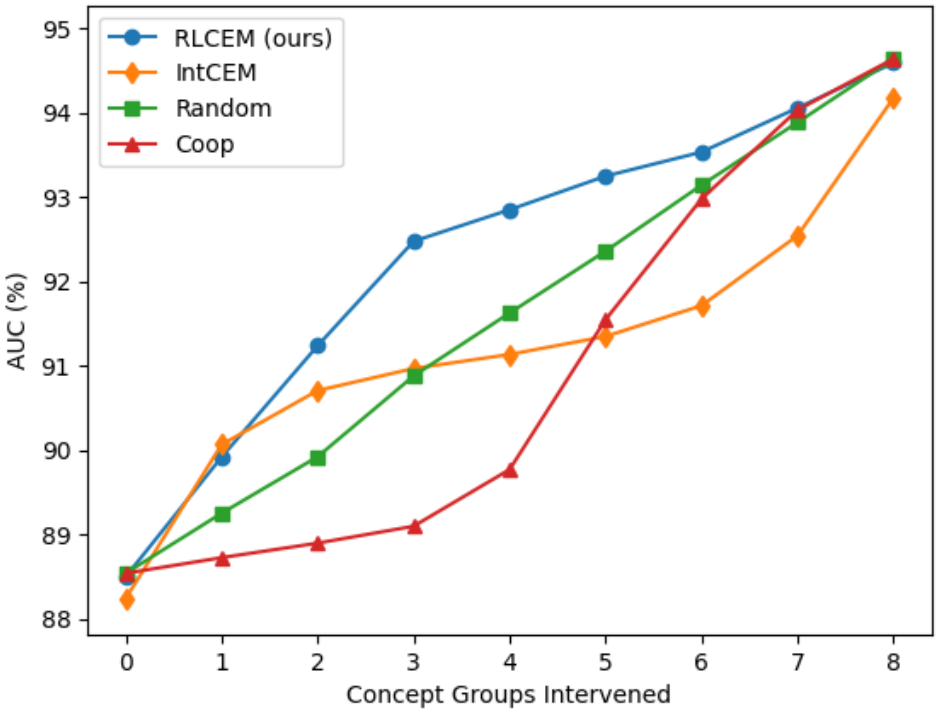

Figure 4.3: Intervention AUC (%) of RLCEM intervention policy compared to existing intervention policies on MNIST-ADD.

test intervention AUC of the different intervention policies, across quartiles of interventions. We see that RLCEM outperforms all other intervention policies, reaching similar or better intervention performance with 25% less groups intervened.

Concept Groups Intervened	RLCEM (Ours)	IntCEM	CooP	Random
0%	**88.5** $\pm$ 0.2	88.24 $\pm$ 0.09	**88.5** $\pm$ 0.2	**88.5** $\pm$ 0.2
25%	**91.2** $\pm$ 0.2	90.7 $\pm$ 0.3	88.9 $\pm$ 0.2	89.9 $\pm$ 0.1
50%	**92.9** $\pm$ 0.3	91.1 $\pm$ 0.3	89.8 $\pm$ 0.5	91.6 $\pm$ 0.1
75%	**93.5** $\pm$ 0.3	91.7 $\pm$ 0.3	93.0 $\pm$ 0.4	93.1 $\pm$ 0.2
100%	**94.6** $\pm$ 0.2	94.2 $\pm$ 0.3	**94.6** $\pm$ 0.2	**94.6** $\pm$ 0.2

Table 4.3: Intervention AUC (%) of RLCEM intervention policy compared to existing intervention policies on MNIST-ADD, across quartiles of interventions performed. Higher is better. We highlight the best performing policy in each row and values within 1 standard deviation.

We have successfully shown that RLCEM is able to find a better policy than CooP and Random, since all these policies were tested on the same CEMs. However despite RLCEM outperforming IntCEM in intervention performance, to analyze the RLCEM and IntCEM policies we need to take into account the sensitivities to interventions, as these two policies are tested on their respective trained CEMs, and the performance difference

could be attributed to the CEM rather than the policy. To test the sensitivity of RLCEM and IntCEM to interventions, we compare applying Random interventions to the two trained CEMs.

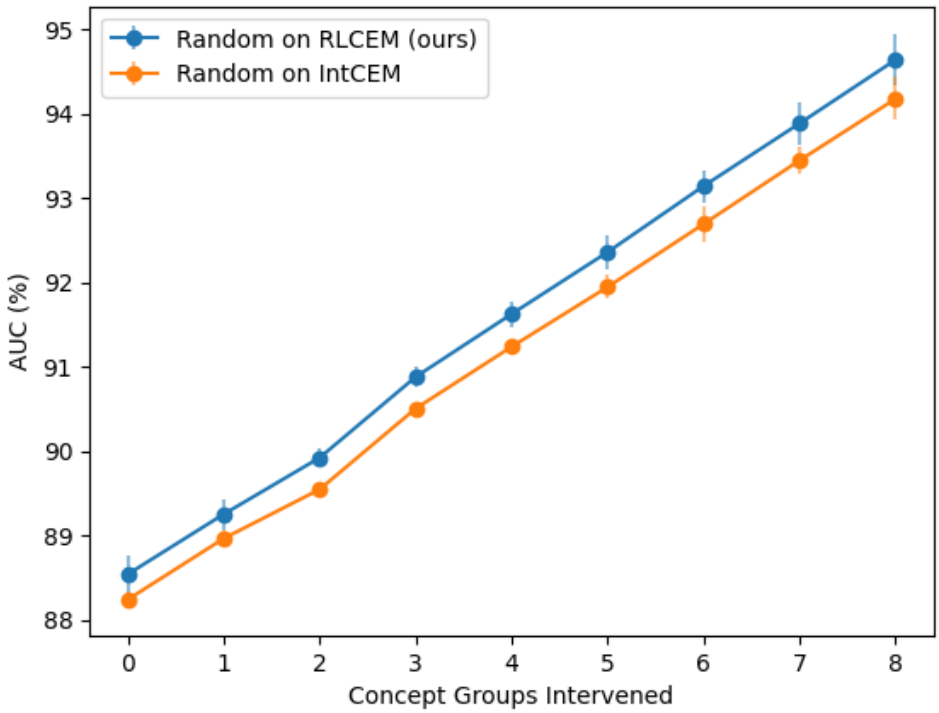

Figure 4.4: Intervention AUC (%) of Random Interventions on RLCEM and IntCEM.

Figure 4.4 shows the intervention performance of Random on RLCEM and IntCEM. We see that the two have very similar intervention performances, both exhibiting a linear increase in AUC as the number of interventions increase. This shows that they have very similar sensitivities to intervention, which allows us to draw the conclusion that RLCEM learns a more optimal intervention policy than IntCEM for different budgets.

4.4.3 CUB

We evaluate the performance of RLCEM against existing greedy intervention policies on CUB. However, an important issue that we faced was loss exponentially increasing, in particular the value loss for the Critic model of the RL agent, as seen in Figure 4.5 where the y-axis is logarithmic.

The reason for this is the outputs of the AC Flow model are too large. In Section 4.3 we mentioned measures we took to prevent the likelihood values outputted by the AC Flow model from being too large. However, CUB has a large number of concepts and the relationships between concepts is more straightforward, where features can be grouped by whether they appear on the same bird species. This is not the case in MNIST-ADD as the distribution of concepts corresponding to different digits are independent of each other.

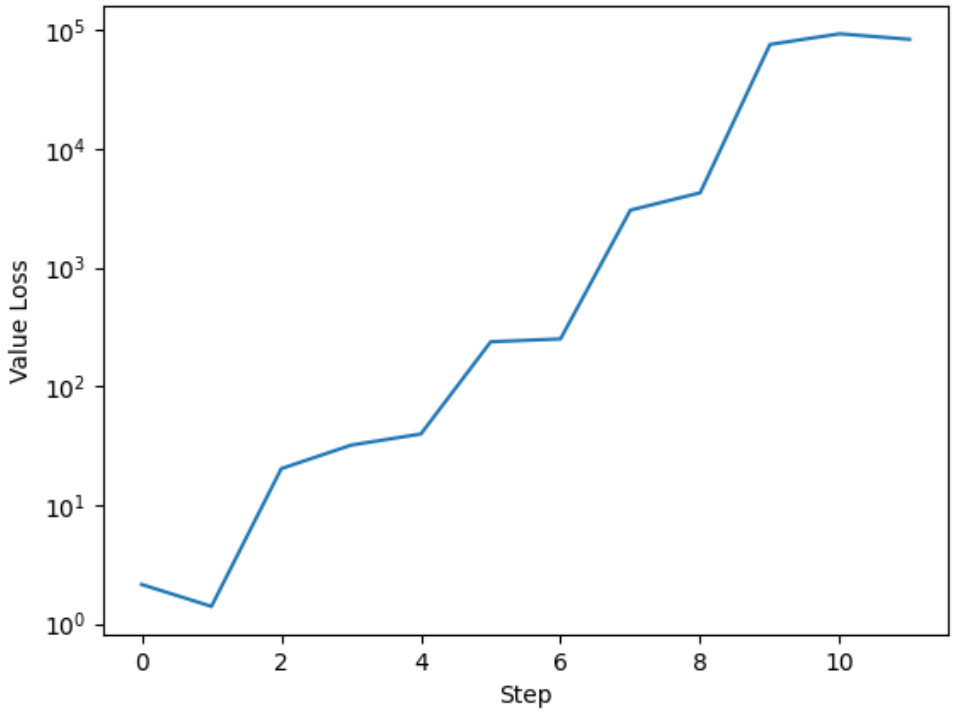

Figure 4.5: Value Loss of RLCEM when trained on CUB, y-axis is logarithmic scale. The value loss increases exponentially with steps, and becomes very large, which is undesirable.

The model for CUB quickly converges and the outputted log likelihoods for concepts increase rapidly which leads to an exponential increase in outputted likelihoods. This means that the rewards for the RL agent are also very large, which result in enormous value losses. Not only does this damage the model's ability to learn as the large loss values and updates cause instability and numerical issues in training, it also means that the intermediate rewards will overpower the final reward, rendering it useless in training the model. This means that the agent only learns to maximize expected information gain rather than also maximizing the post-intervention performance, which is our main goal. Since the issue is with the output of the AC Flow model which is used as rewards, common methods to prevent loss spiking and exploding gradients, such as clipping the gradient or losses, did not alleviate the problem. Normalizing the outputs is also infeasible as that would require normalizing over all concept and label combinations in $p(\mathbf{c}_u \mid \mathbf{c}_o, \mathbf{y})$, which is a massive space.

In order to solve this problem, we add a penalty to the loss term of the AC Flow model to penalize large log likelihoods. As mentioned in Section 3.2.3, we add an L2 penalty

$$\lambda_{l2} \left(\log^2 p(\mathbf{c}_u, \mathbf{c}_o \mid \mathbf{y}) + \log^2 p(\mathbf{c}_o \mid \mathbf{y}) \right)$$

to the loss, and we try to set λ_{l2} as high as possible to penalize large values without affecting the performance of the model. Experimentally we find that setting $\lambda_{l2} = 0.5$ results

in this term almost completely offsetting the training effects of the NLL loss, resulting in poor validation performance. Thus we set $\lambda_{l2} = 0.4$, which does not significantly impact the validation accuracy of the trained AC Flow model. The performance of the AC Flow model with and without the L2 Penalty is shown in Appendix D.

After adding the penalty to the loss term of the AC Flow model, we train new AC Flow models and use them to train RLCEMs on the CUB task. We then evaluate them against the same existing greedy intervention policies. Figure 4.6 shows the performance of RLCEM against existing greedy intervention policies. We note that while RLCEM still has better performance than CooP and Random, it performs similarly and even more poorly in some cases than IntCEM.

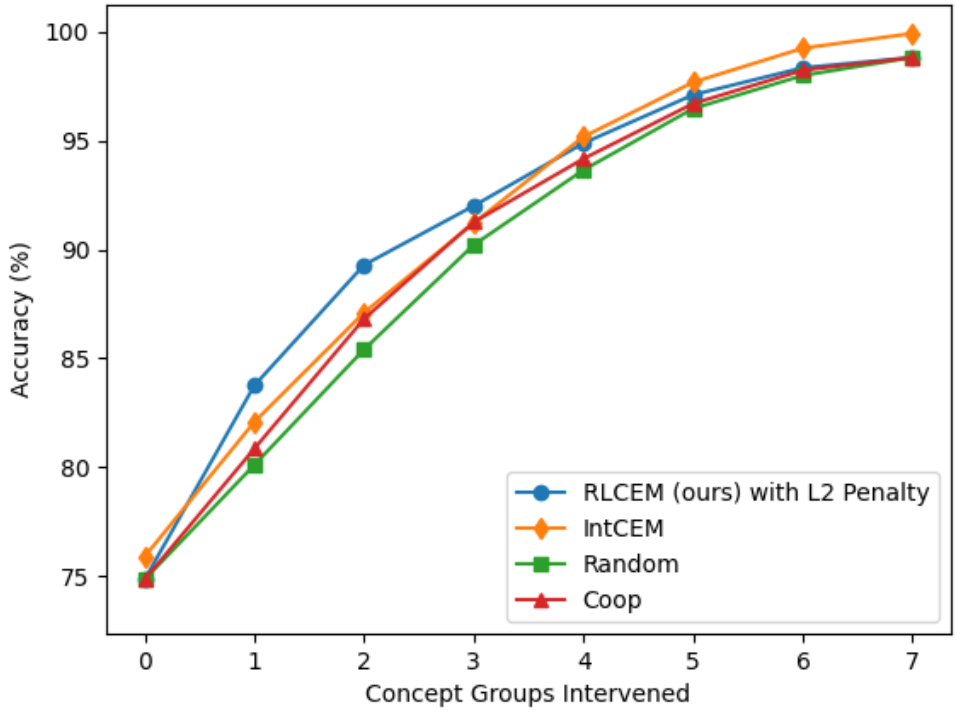

Figure 4.6: Intervention Accuracy (%) of RLCEM intervention policy trained with L2 Penalty, compared to existing intervention policies on CUB.

This is further shown in Table 4.4, where we tabulate the intervention accuracies of RLCEM compared to other greedy policies across quartiles of interventions performed. For each quartile, the accuracy is calculated by linearly interpolating the two points around the quartile boundary. For example, to calculate the 50% quartiles, we take the mean of the intervention accuracy with 3 and 4 groups intervened. From the results we can see that RLCEM only outperforms IntCEM and the other policies at one quartile, and that the RLCEM results have very high standard deviation.

Therefore we cannot conclude that RLCEM is better than the greedy intervention policies. Penalizing large outputs in the AC Flow model results in more similar output values for

Concept Groups Intervened	RLCEM (Ours)	IntCEM	CooP	Random
0%	**74** $\pm$ 1	**75** $\pm$ 1	**74** $\pm$ 1	**74** $\pm$ 1
25%	**87** $\pm$ 2	**85** $\pm$ 1	**85** $\pm$ 3	84 $\pm$ 2
50%	**93** $\pm$ 2	**93** $\pm$ 1	**92** $\pm$ 3	**91** $\pm$ 3
75%	97 $\pm$ 2	**98.1** $\pm$ 0.7	97 $\pm$ 2	96 $\pm$ 2
100%	98.8 $\pm$ 0.8	**99.9** $\pm$ 0.1	98.8 $\pm$ 0.8	98.8 $\pm$ 0.8

Table 4.4: Intervention Accuracy (%) of RLCEM intervention policy trained with L2 Penalty, compared to existing intervention policies on CUB. We display the results across quartiles of interventions performed. Higher is better. We highlight the best performing policy in each row and values within 1 standard deviation.

the likelihood of different concepts, and the approximated information gain is no longer accurate and cannot act as a useful reward to guide the RL agent to make the optimal interventions, and the RL agent struggles to learn a good intervention policy. The large standard deviation also shows that this method is not robust and the performance can greatly differ.

4.4.4 Back to MNIST-ADD

To demonstrate that the L2 Penalty negatively affects the RL agent, we investigate the performance of RLCEM on MNIST-ADD with and without this L2 penalty. As shown in Figure 4.7, after adding a L2 Penalty to the AC Flow model, while still slightly better than Random, the intervention performance of RLCEM significantly decreases. This confirms that adding the L2 Penalty to the AC Flow model worsens the intervention performance of RLCEM, resulting in poor performing AC Flow models. This leads to the inconsistent and poorer intervention performances.

4.5 Limitations

While we have shown that RLCEM can learn a non-greedy intervention policy that outperforms existing greedy intervention policies, it also has its drawbacks. Most notably the largest drawback is its inconsistent performances which are dependent on the output of the AC Flow model. When we add an L2 Penalty to train the AC Flow model, it then struggles to learn to output accurate conditional likelihood. This then affects the RL agent, which struggles to learn the optimal interventions with a

Another limitation of RLCEM is its time complexity. Table 4.5 shows the per-epoch training times of RLCEM. As mentioned in Section 3.3.5 RLCEM has a training time complexity of $O(n^2k)$ compared to IntCEM's $O(n)$, which makes it not scalable for tasks

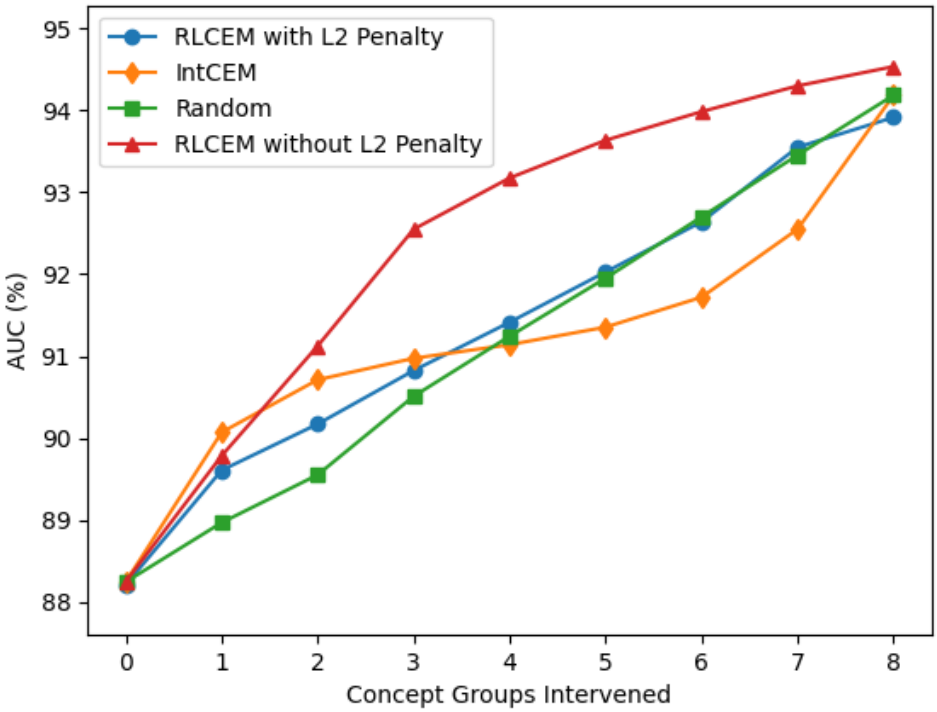

Figure 4.7: Intervention AUC (%) of RLCEM trained with and without an L2 Penalty (for the used AC Flow model). We also include IntCEM and Random for reference.

with large concepts. This is reflected by the much higher per-epoch training time of RLCEM compared to IntCEM, especially fo CUB where one epoch of RLCEM takes more than ten times longer than IntCEM, as CUB has a large number of concepts . This number further increases if we want to increase the number of budgets sampled for training for each mini-batch.

Task	RLCEM	IntCEM
MNIST-ADD	77.8 ± 0.8	13.96 ± 0.04
CUB	610 ± 30	44 ± 5

Table 4.5: Per epoch Training Time in seconds of RLCEM vs IntCEM.

Lastly, RLCEM is also impacted by concept leakage present in CEMs. As mentioned in Section 2.2, the concept embeddings used by CEM result in leakage of information representing other concepts within the concept embeddings. This weakens the performance of the AC Flow models are trained purely on the concept annotations, and concept leakage makes it so the approximated distribution of concepts do not accurately represent the distribution of concepts embeddings.

In this chapter, we have successfully evaluated the performance of RLCEM against existing greedy intervention policies and random. We have showns that RLCEM outperforms the SOTA greedy intervention policies on MNIST-ADD, achieving similar or better in-

tervention performance with 25% less interventions performed. However, this method is not robust as seen from its failures in learning a good non-greedy intervention policy from the CUB task. The method is highly dependent on the quality of outputs from the pre-trained AC Flow model, which in turn can vary from task to task.

Chapter 5

Summary and conclusions

To summarize, we have presented a new CEM-based model, which we name RLCEM, that utilizes Arbitrary Conditional Flow models and Reinforcement Learning to learn a non-greedy intervention policy. A variant of AC Flow model is used to learn the class-specific conditional distribution between concepts, which is then used to train a Reinforcement Learning agent that learns a non-greedy intervention policy. The AC Flow model provides intermediate rewards to the agent based on the expected information gain to the target variable, and the post-intervention Cross-Entropy loss is used as a final reward to train the agent to make more optimal interventions that lead to higher label prediction accuracies. The sampled interventions are also used to update the CEM to increase its sensitivity to interventions, further boosting the performance of the learnt intervention policy.

We have also evaluated RLCEM against existing greedy intervention policies IntCEM, CooP, and a policy where we randomly select concepts to intervene. We show that RLCEM was able to outperform these greedy intervention policies, learning a more optimal non-greedy intervention policy. RLCEM achieves similar or better performance than the other policies while requiring 25% less interventions, showing that it is able to learn a non-greedy policy that outperforms existing greedy policies in intervention performance. At the same time, we show that RLCEM achieves similar un-intervened performance to existing models.

However, we also show that RLCEM does not learn a more optimal intervention policy in certain tasks due to sub-optimal AC Flow models. We show that the performance of RLCEM is highly dependent on the trained AC Flow model, which performance can vary. Hence, this method is not very robust. Additionally, we show that RLCEM also suffers from other problems like high time complexity and concept leakage.

5.1 Future Work

Due to the limitations of RLCEM, there is room for future work to find solutions and improvements to these limitations. For example, using a different approach to model the distribution of concepts, such as a different latent distribution model for the AC Flow model, or using a completely different surrogate models. Further research can investigate these different solutions to improve the robustness of the overall RLCEM by improving the robustness of the surrogate model itself.

Currently the AC Flow model uses an RNN based approach to model the conditional distribution of concepts, which takes $O(d)$ steps per output and is one of the reasons for the high time complexity of our RLCEM approach. Future work can look into using more efficient models to improve the scalability of the AC Flow model, for example transformer-based architectures that have replaced RNNs. However, the current RNN-based approach uses a sequential approach for sampling, which is not straightforward to translated to a transformer-based architecture, and appropriate changes will have to be made.

Lastly, more research can be done on the effects of concept leakage on modelling the distribution of concepts in CEMs. Methods to combat the negative effects of concept leakage, such as dynamically updating the AC Flow model based on the distribution of concept embeddings may be worth exploring.

Bibliography

[1] *Promises and Pitfalls of Black-Box Concept Learning Models*, volume 1, 2021.

[2] Michael Bain and Claude Sammut. A framework for behavioural cloning. In *Machine Intelligence 15*, 1995.

[3] Kushal Chauhan, Rishabh Tiwari, Jan Freyberg, Pradeep Shenoy, and Krishnamurthy Dvijotham. Interactive concept bottleneck models. *Proceedings of the AAAI Conference on Artificial Intelligence*, 37:5948–5955, 06 2023.

[4] Li Deng. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

[5] Kaiming He, X. Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2015.

[6] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[7] Pang Wei Koh, Thao Nguyen, Yew Siang Tang, Stephen Mussmann, Emma Pierson, Been Kim, and Percy Liang. Concept bottleneck models. 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 5338–5348. PMLR, 13–18 Jul 2020.

[8] Yang Li, Shoaib Akbar, and Junier B. Oliva. Flow models for arbitrary conditional likelihoods, 2020.

[9] P. Melville, M. Saar-Tsechansky, F. Provost, and R. Mooney. Active feature-value acquisition for classifier induction. In *Fourth IEEE International Conference on Data Mining (ICDM'04)*, pages 483–486, 2004.

[10] Marvin Minsky. Steps toward artificial intelligence. *Proceedings of the IRE*, 49(1):8–30, 1961.

[11] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. 12 2013.

[12] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin A. Riedmiller, Andreas Kirkeby Fidjeland, Georg Ostrovski, Stig Petersen, Charlie Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518:529–533, 2015.

[13] Junier Oliva, Avinava Dubey, Manzil Zaheer, Barnabas Poczos, Ruslan Salakhutdinov, Eric Xing, and Jeff Schneider. Transformation autoregressive networks. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 3898–3907. PMLR, 10–15 Jul 2018.

[14] Douglas A. Reynolds. Gaussian mixture models. In *Encyclopedia of Biometrics*, 2018.

[15] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 1889–1897, Lille, France, 07–09 Jul 2015. PMLR.

[16] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *ArXiv*, abs/1707.06347, 2017.

[17] Alex Sherstinsky. Fundamentals of recurrent neural network (rnn) and long short-term memory (lstm) network. *Physica D: Nonlinear Phenomena*, 404:132306, 2020.

[18] Sungbin Shin, Yohan Jo, Sungsoo Ahn, and Namhoon Lee. A closer look at the intervention procedure of concept bottleneck models. In *Workshop on Trustworthy and Socially Responsible Machine Learning, NeurIPS 2022*, 2022.

[19] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. A Bradford Book, Cambridge, MA, USA, 2018.

[20] Richard Stuart Sutton. *Temporal credit assignment in reinforcement learning*. PhD thesis, 1984. AAI8410337.

[21] Esteban G. Tabak and Eric Vanden-Eijnden. Density estimation by dual ascent of the log-likelihood. *Communications in Mathematical Sciences*, 8(1):217 – 233, 2010.

[22] Mark Towers, Jordan K. Terry, Ariel Kwiatkowski, John U. Balis, Gianluca de Cola, Tristan Deleu, Manuel Goulão, Andreas Kallinteris, Arjun KG, Markus Krimmel, Rodrigo Perez-Vicente, Andrea Pierré, Sander Schulhoff, Jun Jet Tai, Andrew Tan Jin Shen, and Omar G. Younis. Gymnasium, March 2023.

[23] Martijn van Otterlo and Marco Wiering. *Reinforcement Learning and Markov Decision Processes*, pages 3–42. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[24] Catherine Wah, Steve Branson, Peter Welinder, Pietro Perona, and Serge Belongie. *The Caltech-UCSD Birds-200-2011 Dataset*. Jul 2011.

[25] Xinyue Xu, Yi Qin, Lu Mi, Hao Wang, and Xiaomeng Li. Energy-based concept bottleneck models: Unifying prediction, concept intervention, and probabilistic interpretations. In *International Conference on Learning Representations*, 2024.

[26] Mateo Espinosa Zarlenga, Pietro Barbiero, Gabriele Ciravegna, Giuseppe Marra, Francesco Giannini, Michelangelo Diligenti, Zohreh Shams, Frederic Precioso, Stefano Melacci, Adrian Weller, Pietro Lio, and Mateja Jamnik. Concept embedding models. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems*, 2022.

[27] Mateo Espinosa Zarlenga, Katherine M. Collins, Krishnamurthy Dj Dvijotham, Adrian Weller, Zohreh Shams, and Mateja Jamnik. Learning to receive help: Intervention-aware concept embedding models. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

[28] Andreas Zell. Simulation neuronaler netze. 1994.

Appendix A

Dataset Details

Appendix B

Surrogate Model Details

B.1 Transformations

B.2 Hyperparameters

Appendix C

Hardware Specifications

Appendix D

L2 Penalty