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: 42

Main chapters (excluding front-matter, references and appendix): 27 pages (pp 9–35)

Main chapters word count: 7564

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

7564
```

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 23, 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, Dr 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 [5] (CBMs) and Concept Embedding Models [21] (CEMs) 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 [2, 22], 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 [2], or ML-based approaches that learn a greedy policy like in IntCEMs [22]. Additionally existing approaches either attempt to model the distribution of concepts and use that to

sample interventions [20], or directly learn a policy [22], 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 [15] (RL) 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 to learn a non-greedy intervention policy that can learn an intervention policy with better intervention performance than existing methods, achieving similar performance with up to 25% less interventions. At the same time the model maintains similar performance under the absence of interventions.

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

Chapter 2

Background and Related Work

2.1 CBM

Concept Bottleneck Models (CBMs), initially proposed by Koh et al. [5], 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 neurons as the number of concepts, where this layer is referred to as the "CBM Bottleneck". Henceforth g is referred to as the "concept predictor $\mathbf{x} \to \mathbf{c}$ model" and f as the "label $\mathbf{c} \to \mathbf{y}$ predictor model". Training such a model requires a dataset of inputs $\mathbf{x}$ annotated with the corresponding concepts $\mathbf{c}$ and labels $\mathbf{y}$.

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

2.1.1 Training CBMs

There are several different ways to train a CBM. If we let the concept loss L_{concept} be a loss function that measures the discrepancy between the predicted concepts $\hat{\mathbf{c}}$ and the actual concepts $\mathbf{c}$, and similarly the label loss L_{label} measuring the discrepancy between the predicted concepts $\hat{\mathbf{y}}$ and the actual concept $\mathbf{y}$, both losses as illustrated in Figure 2.1. There are the following ways to train a CBM as proposed in [5].

1. Independent: Training the two models independently by minimizing $L_{\text{concept}}(g(\mathbf{x}), \mathbf{c})$ and $L_{\text{label}}(f(\mathbf{c}), \mathbf{y})$ independently.
2. Sequential: Training the models one by one, first learning $\hat{g}$ by minimizing

   $L_{\text{concept}}(g(\mathbf{x}), \mathbf{c})$, then learning f by minimizing $L_{\text{label}}(f(\hat{\mathbf{g}}(\mathbf{x})), \mathbf{y})$

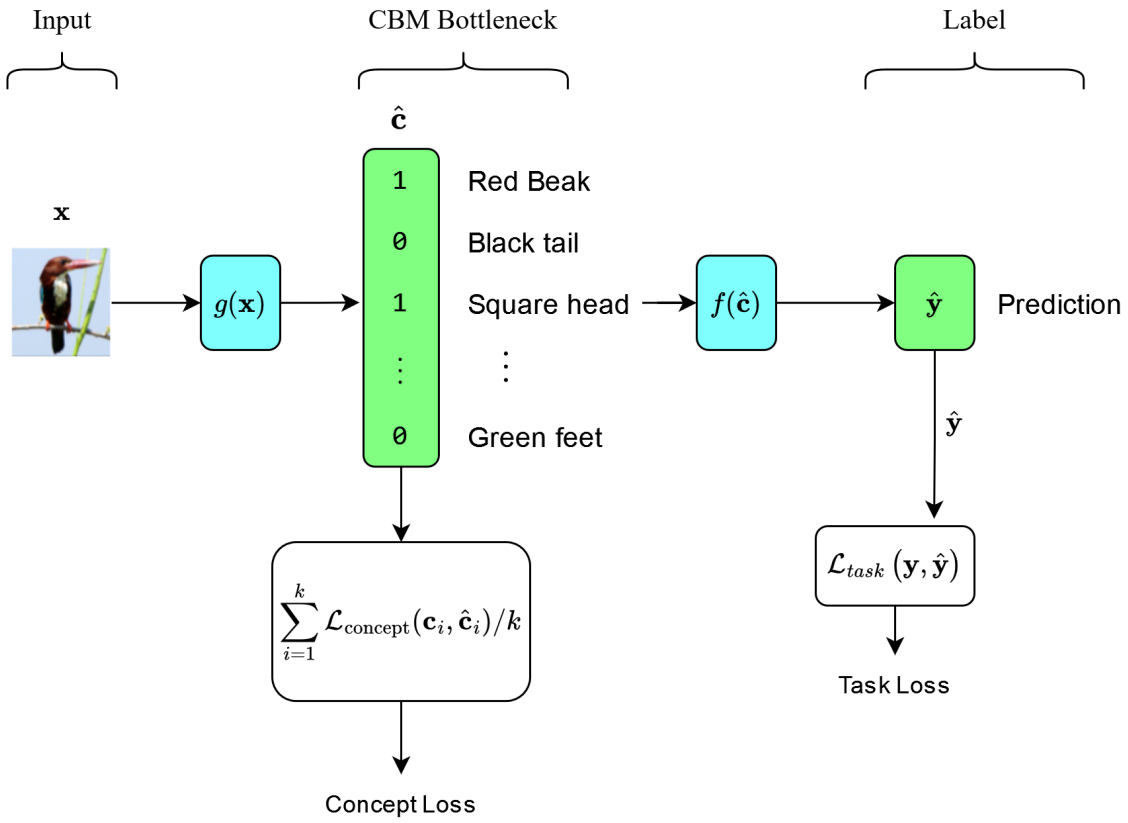

Figure 2.1: The CBM Architecture [5].

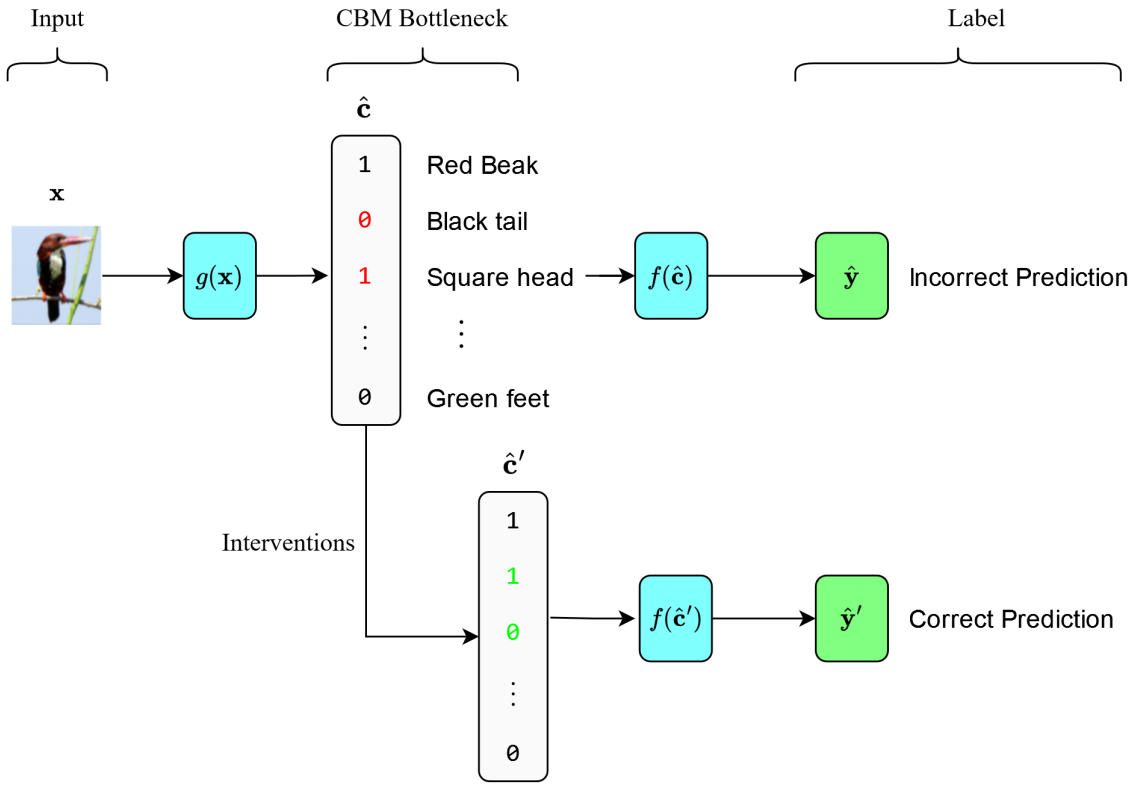

Figure 2.2: An illustration of intervening on the concepts predicted by a CBM.

3. Joint: The model is trained via a weighted sum of the losses given by
   $\lambda_{\text{concept}} L_{\text{concept}}(g(\mathbf{x}), \mathbf{c}) + \lambda_{\text{label}} L_{\text{label}}(f(g(\mathbf{x})), \mathbf{y})$
   such that both losses are minimised simultaneously.

It has been shown experimentally that while the joint models perform the best without interventions, followed by the sequential model, and the independent model performs the worst. This is because the sequential model allows the $\mathbf{c} \to \mathbf{y}$ model to learn a mapping from the concepts produced by the $\mathbf{x} \to \mathbf{c}$ model to label $\mathbf{y}$, where the concepts produced by the $\mathbf{x} \to \mathbf{c}$ model is often different from the true concepts, an underlying requirement for the independent model to perform well. Additionally, the joint model allows the $\mathbf{x} \to \mathbf{c}$ model to simultaneously learn to output a representation of concepts that allow for best performance of the $\mathbf{c} \to \mathbf{y}$ model [5].

When comparing performance under interventions, independent models outperform the two models. They are more sensitive to interventions and each successive intervention step leads to a bigger increase in performance compared to the other two, with better performances after the same number of interventions. The reason behind this is that the independent model learns a mapping from the true concepts to the label, whereas the other two learn a mapping from the predicted concepts to the label. Each intervention modifies the predicted concepts to be closer to the true concepts, which is what the $\mathbf{c} \to \mathbf{y}$ independent model is trained to do [5].

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 that learn a direct mapping from input to labels. This is because the CBM label prediction model relies only on a set of human-interpretable concepts, which limits the performance of the model as traditional models can extract information outside of these concepts that are potentially not human-interpretable [21]. This is even more apparent if the dataset does not contain a complete set of concepts that cover all features present in the input, which is very common in real-life scenarios. As such, there is a trade-off between performance and interpretability, where researchers have developed methods such as extending the CBM bottleneck with a set of unsupervised neurons to increase accuracy at a cost of decreasing interpretability [1].

To overcome this trade-off, Concept Embedding Models (CEMs) were proposed by Zarlenga et al. [21], these are CBMs that further add an additional layer of learnable embeddings before the original bottleneck, learning two embedding vectors for each concept: one for when the concept is and is not present. The architecture of CEMs is shown in Figure **??**. An intermediate scoring function ϕ_i is learnt for each concept i, and the embedding assigned to the bottleneck is an interpolation of the two embeddings based on the scoring function predicting the possibility of the concept to be present.

This architecture also allows for interventions during run-time. By simply modifying the output of the scoring function to be that of the true concept, the bottleneck can be modified similarly to CBMs and improve the performance of the model. Additionally to increase the performance of CEMs, the authors utilised observations mentioned in Section 2.1, where models trained on the true concepts are more sensitive to interventions. They proposed RandInt, a method to randomly intervene during training with $\lambda_{\text{int}} = 0.25$ probability of intervening on a concept. They show that this effectively boosts the performance of the model under interventions during test time without notable effects to the performance without interventions [21].

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 embedding structure for CEMs allow for encoding of more information in the concept representations and is more machine-interpretable, where the additional information in the bottleneck compared to scalar representations in CBMs lead to a better performing label predictor model. Additionally CEMs are still trained on the same set of human-interpretable concepts as CBM via a similar concept loss, which leads to high interpretability and good intervention performance. 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 [21].

2.3 IntCEM

Building on top of CEMs, Zarlenga et al. [22] introduced Intervention-aware CEM (IntCEM), which are 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 L_{int} and task loss L_{task} for the intervened concepts.

During training, ψ first samples intervention logits for the next concept to intervene on, then a Cross Entropys loss L_{int} 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 a Behavioural Cloning [] approach where ψ learns to clone the behaviour of a greedy optimal policy.

As mentioned in Section 2.1, training using true concept labels increases the model's sensitivity to interventions, leading to better intervention performance. IntCEM incorporates this idea by computing task loss 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.

2.4 RL

Reinforcement Learning is a machine learning subfield that focuses on training agents to make sequential decisions in an environment. Contrary to supervised learning where the goal is to minimize the discrepancy between predicted outputs and true outputs, the goal of Reinforcement Learning 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

The PPO algorithm is a policy gradient method that combines the strengths of traditional policy-based approaches and value-based approaches [9, 10]

and is one of the most popular Reinforcement Learning algorithms for learning an optimal policy due to its robustness.

2.5 Flow Models

Flow models model probability distributions by leveraging the change of variable property [17]. Given an input distribution $x \sim p_X(\mathbf{x})$, the change of variable property allows us to define deterministic invertible transformations $f : \mathbf{x} \to \mathbf{z}$, such that $\mathbf{z} = f(\mathbf{x})$ where there exists a simple invertible function f^{-1} such that $\mathbf{z} = f^{-1}(f(z))\forall z$.

Flow models define transformations using ML models with this property with learnable parameters

Each of these transformations give a normalized density, and they can be composed to create more complex invertible distributions. This property allows us to define normalizing flow models which learn a series of transformations that use a simple latent distribution to model a complex distribution, such as the distribution of concepts within a dataset for CEMs.

2.6 Related Work

2.6.1 CooP

2.6.2 Active Feature Acquisition

Chapter 3

Method

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 $\mathbf{c} \to \mathbf{y}$ label prediction model. In this chapter, we document how we propose to learn such a non-greedy intervention policy. We first formalize interventions and intervention policies, then introduce the surrogate models we use to calculate rewards, and finally, present how Reinforcement Learning is used to learn a non-greedy intervention policy.

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 $\boldsymbol{\mu}$.

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

Figure 3.1 demonstrates how an intervention is formed using a binary mask $\boldsymbol{\mu}$ from the predicted concepts $\hat{\mathbf{c}}$ and the true concepts $\mathbf{c}$.

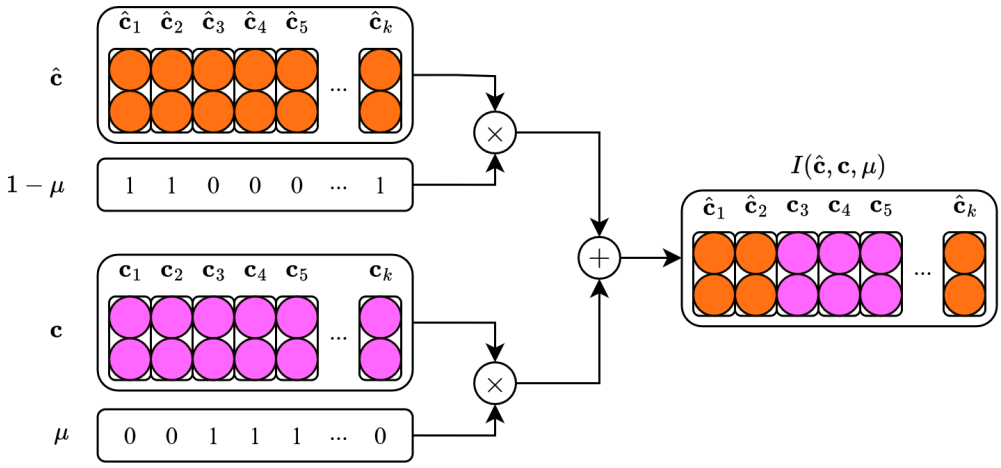

Figure 3.1: A figure of how interventions are formed using binary masks.

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 $\hat{g} : \mathbf{c} \rightarrow \mathbf{y}$ 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 $\mathbf{c} \rightarrow \mathbf{y}$ 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 $\mathbf{c} \rightarrow \mathbf{y}$ 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 probabilities 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 [8, 16], 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. [7] in Active Feature Acquisition, we use a surrogate model to approximate the conditional probabilities $p(\mathcal{C}_u \mid \mathcal{C}_o, \mathbf{y})$, 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 [17], namely Arbitrary Conditional Flow [6] (AC Flow) models as our surrogate models. These models are flow models augmented to model arbitrary conditional probabilities $p(\mathcal{C}_u \mid \mathcal{C}_o)$ 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 probabilities, which can be extended to include conditional probabilities. AC Flow models build on Transformation Autoregressive Networks [11] (TANs), and learn to model likelihoods $p(c_0, \ldots, c_n \mid \mathbf{y})$, modelling the underlying latent distribution using an

autoregressive approach with Recurrent Neural Networks (RNNs) [14]. The RNN learns to model the likelihood of $p(z_0, \ldots, z_n \mid \mathbf{y})$ 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 [12]. This allows us to compute $p(z_0, \ldots, z_i \mid \mathbf{y})$, the likelihood of the current i variables, using a weighted sum of the probability density of the K 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 $p(z_0, \ldots, z_n \mid \mathbf{y})$, 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 $p(z_0, \ldots, z_n \mid \mathbf{y})$ to the concept likelihoods $p(c_0, \ldots, c_n \mid \mathbf{y})$, 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. [6], 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 $q_{\mathbf{c}_o,b}(\mathbf{c}_u) = \mathbf{z}_u$. We can then apply the change of variable theorem with a conditional extension, and using the Jacobian determinant tells us how the likelihood changes [17]

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

In the AC Flow model, we mainly leverage linear transformations. We use Multi-Layer Perceptrons [23] (MLP) 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 **??**, 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

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

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 probability

$p(\mathbf{c}_u \mid y)$. 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 and how to find their Jacobian determinant can be found at [11].

These tranformations and the latent distribution are both learnt to model the conditional likelihoods $p(\mathbf{c}_0, \ldots, \mathbf{c}_n \mid \mathbf{y})$. This can then be used to compute the arbitrary conditional likelihood $p(\mathbf{c}_u \mid \mathbf{c}_o, \mathbf{y})$ which is the likelihood of seeing a set of unobserved concepts $\mathbf{c}_u$ given a set of observed concepts $\mathbf{c}_o$ and a class $\mathbf{y}$. By using Bayes' theorem, we note that

$$p(\mathbf{c}_u \mid \mathbf{x}_o, \mathbf{y}) = \frac{p(\mathbf{c}_u, \mathbf{c}_o \mid \mathbf{y})}{p(\mathbf{c}_o \mid \mathbf{y})}$$

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 pass in an empty set of concepts as the condition. Due to the invertible property of the learnt transformations, a model learnt this way also allows us to sample from the underlying probability distribution then applying the transformations, which gives us samples from the input distribution. This is useful for the RL agent to determine interventions as this provides information on which concepts are likely to be present (or not present) given the currently intervened concepts. In general, we follow the advised hyperparameters from Li et al. [6] for building the transformations used in our AC Flow model. This includes the rank of the linear matrices W and the hidden dimension of the latent distribution RNN. However, we notice that while they use $n \times k$ components in the Gaussian Mixture Model where n is the number of concepts and k is the number of classes, we can use n components without any noticeable performance drop. The main power of how the AC Flow model is able to model the data distributions accurately is via sufficient transformations, and thus we do not change them.

3.2.3 Training the Surrogate Models

In this project, we adapt AC Flow models to model the probabilities of concepts in CEMs during interventions. We follow the description of Li et al. [7] 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

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

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 a L2 regularization loss, we use the square of the logits for $\log p(\mathbf{c}_u, \mathbf{c}_o \mid \mathbf{y})$ and $\log p(\mathbf{c}_o \mid \mathbf{y})$. The effects and reasoning of this term is further discussed in Section **??**. 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 CBMs as our base models, as CEMs have been shown to have better performance with and without interventions compared to CBMs, and its robustness to concept-incompleteness, which is when the concepts present in the dataset annotations do not contain all possible concepts, we utilise CEMs as our base model.

RLCEM is a CEM augmented with an RL agent, and is the main focus of this project. We construct RLCEMs according to the description below, which is then compared against the current best performing model for interventions IntCEM. The two models are evaluated on the datasets described in 4.1 and we report the intervention performance.

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 [18].

- States are the observations, including all the information that is 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 most likely un-intervened concepts $\bar{\mathbf{c}_u} \sim p(\mathbf{c}_u \mid \mathbf{c}_o, \mathbf{y})$ $\bar{\mathbf{c}_u} \sim p(\mathbf{c}_u \mid \mathbf{c}_o)$, sampled from both the class-specific likelihoods and the marginalized likelihood as described in Section 3.2.
- An action corresponds to intervening on a concept out of all possible concepts. For

simplicity, we split concepts into concept groups, and each action intervenes on all concepts within a concept group. This is further discussed in Section 4.1.

- The rewards of each step is given by the increase in information to the target variable y, $H(y \mid x_o) - \mathbb{E}[H(y \mid x_u, x_o)]$. This information is directly obtained from the AC Flow surrogate model. When the entire intervention sequence finishes, or when we reach the termination state where the budget is insufficient for more interventions, the final reward is the negative of the Cross Entropy Loss of the predicted label with respect to the true label, where a higher value corresponds to a lower discrepancy between the prediction and the true value. This corresponds with the goal of maximizing accuracy after a set number of interventions. As this requires knowledge of the true label, this reward is only computed during training for computing the loss and is not accessible to the model during testing. 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 reward.

- The sequence reaches the termination state when the entire budget has been consumed.

1. Incorporate the acquisition costs for each concept into the reward by subtracting the corresponding cost. Thus 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. Only utilize the information gain in each step to determine

We end up choosing the second approach. After utilizing the first approach, we found it difficult to balance the weight of costs versus the other rewards, and it increases the hyperparameters we have to tune, and allowing the RL agent to decide when to terminate the intervention process does not really work for intervening with budgets. This is because

To simplify the problem, 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 Comparison with Active Feature Acquisition

In Active Feature Acquisition as described in Section **??**, Li et al. [7] combine Reinforcement Learning with AC Flow models to give impressive results on finding the optimal concepts to acquire from the environment. We adopt a similar approach to how the RL agent itself is trained.

We first pre-train an AC Flow model that learns arbitrary conditional distributions about

the underlying concepts $p(\mathbf{c}_u \mid \mathbf{c}_o, y)$. Then, A Reinforcement Learning agent is trained to learn the order of concepts to acquire in order to maximize the accuracy of a label predictor model. At each step, the agent is given the current set of acquired features x_o, and the agent samples the next feature to acquire x_u, where the agent is rewarded based on the expected information gain to the target variable y, $H(y \mid x_o) - \mathbb{E}[H(y \mid x_u, x_o)]$. This can be simplified as $H(x_u \mid x_o) - \mathbb{E}[H(x_u \mid x_o, y)]$, and can directly be estimated by the AC Flow model by computing the conditional probability densities $p(x_u \mid x_o, y)$, and $p(x_u \mid x_o)$ by marginalization. Li et al. [7] also show that using this intermediate reward will not affect the optimality of the learnt policy.

We pre-train these models on the concept-annotated dataset, and then use the frozen AC Flow model to train our RL agent. In particular, during step i of the intervention process, the set of unintervened concepts correspond to the unobserved features, vice versa, and we use the conditional probability density of intervened concepts as rewards to the RL agent similar to above. Additionally, at each step with intervened concepts $\mathbf{c}_o$, the agent has access to the sampled concepts $\mathbf{c}_u$ from the AC Flow model. This includes $\mathbf{c}_u$ sampled from $p(\mathbf{c}_u \mid \mathbf{c}_o, y)$ and $p(\mathbf{c}_u \mid \mathbf{c}_o)$, which provides information on the concepts with the highest likelihood of being present given the intervened concepts, both with and without the label y. This allows the RL agent to learn to intervene on concepts that are more likely to be predicted incorrectly, leading to improvements in accuracy of the predicted labels.

Compared to Active Feature Acquisition, the problem setting is a lot more complex due to the fact that rather than simply acquiring features from the environment, we are trying to determine which concepts are more likely to be incorrectly predicted by the $\mathbf{x} \to \mathbf{c}$ model, as well as 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 [13] (PPO), 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 is then used to learn a policy that take actions that lead to states with maximum value estimated by the Critic model. This model effectively balances the tradeoff between exploration and exploitation present in Reinforcement Learning problems, as initially the Actor model explores states which may have low true value which is not learnt by the Critic model yet, and as the Critic model learns to estimate the value of states better,

the Actor model also learns a better policy. 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. It also utilizes a clipping function to prevent the policy update to be too large or too small. 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. [22] 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 depicted in Figure **??**, the joint RLCEM is trained as follows:

1. The $\mathbf{x} \rightarrow \mathbf{c}$ model first computes the predicted concepts $\mathbf{c}$. We then compute a concept loss L_{concept} with the true concepts. We also compute a task loss L_{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 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, the RL agent samples actions, which we then use to update the intervened concepts and calculate the corresponding rewards using the AC Flow model and true label according to Section **??**.

3. For each budget T, the intervened concepts after consuming the budget $\hat{\mathbf{c}}$, as well as the prediction $\hat{\mathbf{y}}$ using $\hat{\mathbf{c}}$, is then used to compute a task loss L_{task} discounted by γ^T, similar to IntCEM.

4. Additionally, we compute the true value of each state that we pass to the RL agent using the discounted future rewards. We compute a value loss with respect to the Critic model's estimate of the value of states. Additionally we compute a policy loss for the Actor model's policy by comparing the value of states before and after the sampled actions.

The overall idea 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 a task loss using the intervened concepts to increase the CEM's sensitivty to interventions. Then we train our intervention policy model so that it makes interventions

that lead to a higher accuracy.

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 proportional 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 **??**.

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. All experiments in this chapter are conducted across multiple trials, where we report the mean and standard deviation. This helps make our results statistically significant and reduces the errors that arise from randomness, which is unavoidable in Machine Learning experiments. We compare our RLCEM which is a CEM augmented with an RL agent to learn a non-greedy intervention policy, to our baseline, which is IntCEM, and a number of other heuristic-based intervention policies, including CooP that selects the concept which minimizes uncertainty, and Random that selects random concepts to intervene on.

4.1 Models and Datasets

To evaluate the performance of our RLCEM model, we follow Zarlenga et al. [22] and select the datasets MNIST-ADD and CUB for our experiments. Since these datasets have different input image sizes and different number of possible classes, the $\mathbf{x} \rightarrow \mathbf{c}$ and $\mathbf{c} \rightarrow \mathbf{y}$ models are slightly different, which we directly select the same models as Zarlenga et al. [22] for a fair comparison. These CEM sub-models are kept the same for RLCEM and the baseline IntCEM to ensure the results reflect a fair comparison of the learnt intervention policy. Additionally for most of our ablation studies, we conduct it on MNIST-ADD due to its straightforwardness. For both datasets, 20% of the training dataset is selected as a validation dataset to monitor the performance of the model during training. The validation dataset is not used to update the model, but is rather used for hyperparameter selection and early stopping to prevent over-fitting.

4.1.1 MNIST-ADD

MNIST-ADD is a synthetic dataset created from the MNIST [3] dataset, a dataset of handwritten digits from 0 to 9. The MNIST dataset contains black-and-white samples with an input size of $1 \times 28 \times 28$, and as shown in Table 4.1, MNIST-ADD sample selects

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 datasetss, which is when the concept annotations in the dataset may not contain all possible concepts relevant, we select only 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, and 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. To generate the MNIST-ADD dataset, we generate 10,000 training samples and 10,000 test samples by choosing each of the 12 input images randomly from the MNIST training and testing datasets respectively.

For this dataset, we use a ResNet-18 [4] backbone for the $\mathbf{x} \rightarrow \mathbf{y}$ 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 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} as the hidden layers, and forwards the 54 predicted concepts to predict the binary label.

4.1.2 CUB

CUB [19] is a real-life dataset that identifies bird attributes. As shown in Table 4.1, input $\mathbf{x}$ consist of $3 \times 299 \times 299$ images of birds, and output $\mathbf{y}$ is a label representing the bird species. We construct 112 annotation concepts using identifying features of birds such as their colour, shape, etc., then group it into 28 concept groups according to Zarlenga et al [22]. We further group these into 7 different concept groups for easy visualization and a better comparison with MNIST-AFA. 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, which we fully utilise during training and testing.

For this dataset, we use a ResNet-34 [4] 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 layers, and forwards the 112 predicted concepts to predict the label out of 200 classes.

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

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	100

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

4.1.3 Evaluating Performance

4.1.4 Reinforcement Learning Agent Model

We adopt the same MLP for both the Actor and Critic model in the RL agent. We use the same number of layers as the intervention policy MLP used in IntCEM, but increase the number of neurons in the hidden layers to {512, 512, 256, 256} from {128, 128, 64, 64} as our problem is more complex, requiring learning $O(n^2)$ tasks instead of $O(n)$ ($O(n)$ 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. We find that increasing the number of hidden layer neurons in the MLP used by IntCEM to {512, 512, 256, 256} does not have any noticeable improvements to the intervention performance on a validation set.

4.2 Non-greedy policies

Before evaluating the performance of RLCEM, it is important to test whether or not 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, i.e. the policies that select the concepts to intervene on that lead to the highest accuracy. During testing, both of these policies have access to the true label $\mathbf{y}$, which allows them to find the concepts to intervene on such that the output $\hat{\mathbf{y}} = \hat{g}(\hat{\mathbf{c}})$ has 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 train a greedy intervention policy model directly by training it to mimic the behaviour of a Greedy Optimal policy, it is infeasible to train a non-greedy intervention policy model to mimic the behaviour of a True Optimal

policy.

GreedyOptimal and TrueOptimal represent the theoretical upper bound intervention performance that greedy and non-greedy policies can achieve, 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. [22], and evaluate the intervention performance when using GreedyOptimal and TrueOptimal.

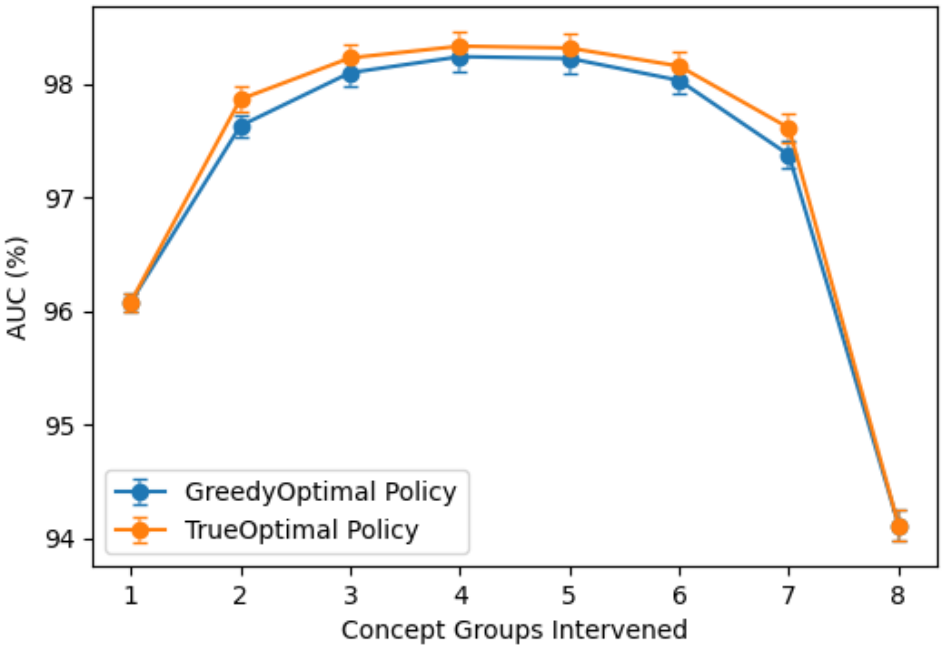

Figure 4.1: Intervention AUC on test set of using GreedyOptimal and TrueOptimal on an IntCEM trained on MNIST-ADD.

Figure 4.1 shows the test performance of GreedyOptimal and TrueOptimal on an IntCEM trained on MNIST-ADD. 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 less interventions (eg. when 4 groups are intervened) is higher than when more groups are intervened (eg. 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.

4.3 Surrogate Models

We then look into the training of the surrogate models, which are used to model the conditional likelihoods $p(\mathbf{c}_u \mid \mathbf{c}_o, y\mathbf{y})$ 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. 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. The accuracy is its accuracy in predicting the label $\hat{\mathbf{y}}$ which is given by

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

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. We stop training if the validation loss does not improve over several epochs, a common tactic used to avoid over-fitting.

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 and early stopping if validation loss does not improve over 5 epochs. A large issue that we observe is that the validation loss keeps decreasing and there seems to be no limit on when it stops. 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, at every point tells us the likelihood of that point, it does not output an absolute probability with values in $[0, 1]$. Therefore the log likelihood can fall outside of $[-\infty, 0]$, and contain positive values, which is translated to a negative NLL and negative loss. As the model keeps training, the model learns to output larger and larger values for the more probable likelihoods, and the NLL keeps decreasing to below 0, and the model does not appear to be able to stop as there is no upper limit to the values it can output (and no lower limit to the NLL and hence loss). At the same time we can see that the validation accuracy stops improving after around 20 epochs and deteriorates, indicating that the model is over-fitting long before validation loss starts to converge. As a result, we use the validation accuracy as the 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.

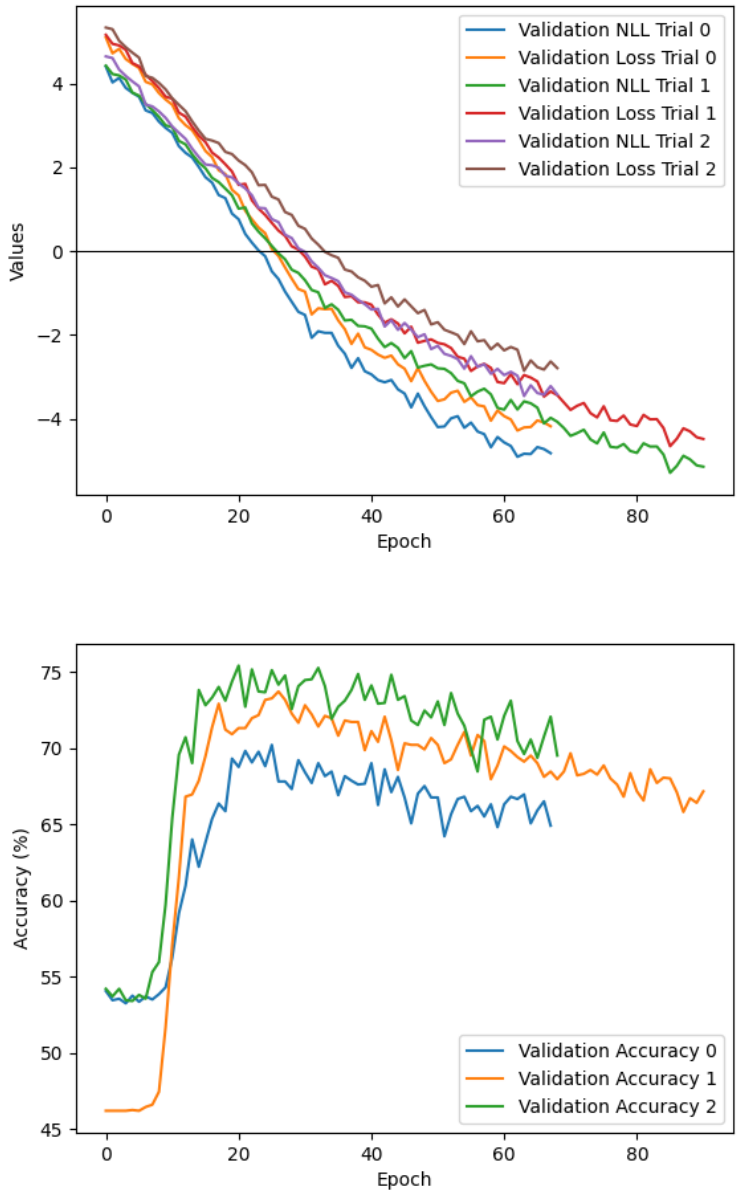

Figure 4.2: Performance on Validation Set when training an AC Flow model on the MNIST-ADD dataset across 3 trials. Top: Validation NLL and loss. Bottom: Validation Accuracy.

4.4 RLCEM Performance

4.4.1 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.

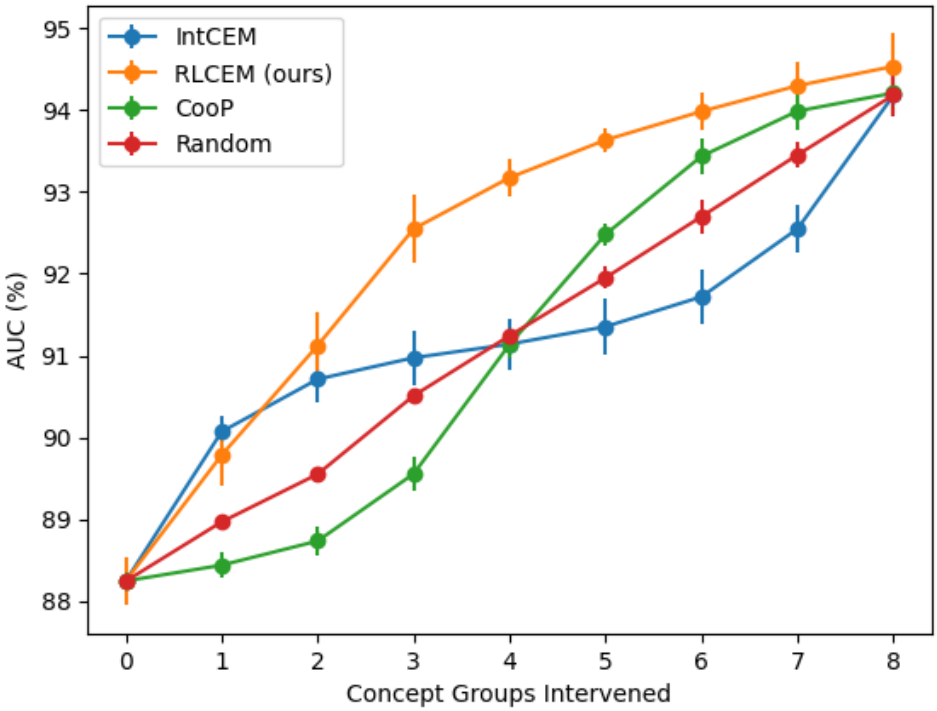

Figure 4.3: Intervention AUC (%) of RLCEM intervention policy compared to existing intervention policies.

4.4.2 CUB

4.5 Limitations

Concept Groups Intervened	RLCEM (Ours)	IntCEM	CooP	Random
0%	**88.2** $\pm$ 0.3	**88.24** $\pm$ 0.09	**88.24** $\pm$ 0.09	**88.24** $\pm$ 0.09
25%	**91.1** $\pm$ 0.4	**90.7** $\pm$ 0.3	88.7 $\pm$ 0.2	89.55 $\pm$ 0.08
50%	**93.2** $\pm$ 0.2	91.1 $\pm$ 0.3	91.1 $\pm$ 0.2	91.24 $\pm$ 0.07
75%	**94.0** $\pm$ 0.2	91.7 $\pm$ 0.3	93.4 $\pm$ 0.2	92.7 $\pm$ 0.2
100%	**94.5** $\pm$ 0.4	**94.2** $\pm$ 0.3	**94.2** $\pm$ 0.3	**94.2** $\pm$ 0.3

Table 4.2: Intervention AUC (%) of RLCEM intervention policy compared to existing intervention policies, higher is better. We highlight the best performing policy in each row and values within 1 standard deviation.

Chapter 5

Summary and conclusions

5.1 Future Work

Bibliography

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

[2] 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.

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

[4] 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.

[5] 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.

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

[7] 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.

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

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

[10] 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.

[11] 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.

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

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

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

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

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

[17] 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.

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

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

[20] 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.

[21] 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.

[22] 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.

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

Appendix A

Dataset Details

Appendix B

Surrogate Model Details

Appendix C

Hardware Specifications

Appendix D

Training Graphs