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: 29

Main chapters (excluding front-matter, references and appendix): 15 pages (pp 9–23)

Main chapters word count: 467

Methodology used to generate that word count:

```
$ make wordcount
gs -q -dSAFER -sDEVICE=txtwrite -o - \
   -dFirstPage=6 -dLastPage=11 report-submission.pdf | \
egrep '[A-Za-z]{3}' | wc -w
467
```

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 16, 2024 -

Abstract

Concept Bottleneck Models (CBMs) are designed for high interpretability by predicting human-interpretable concepts from input, then predicting labels from these concepts. During run-time, 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 concepts to intervene on can have significant impacts to the accuracy of the CBM. Since experts have limited time, it is important to develop methods that can determine the best order of concepts to intervene on to maximize the performance of the model while minimizing the costs that come with interventions. Given a budget for interventions, we want to develop methods that achieve the best model performance using the limited number of interventions allowed. We investigate methods to learn a non-greedy policy that determines an optimal set of concepts to intervene on for a given budget, with the goal of maximizing the label prediction accuracy of the underlying CBM. Past methods in developing intervention policies are limited in that they only use greedy policies that attempt to maximize the accuracy of the CBM 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, while maintaining similar performance under the absence of interventions. In summary, We show that Reinforcement Learning serves as a viable approach to learning intervention policies more optimal than existing appraoches for different budgets, opening the door to 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 [3] (CBMs) and Concept Embedding Models [12] (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. 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, 13], 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 finding a non-greedy policy to determine the set of concepts to intervene on for a budget, as 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 [13]. Additionally existing approaches either attempt to model the distribution of concepts and use that to sample interventions [11], or directly learn a policy [13], 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 [8] (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, to provide intermediate rewards and 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, whereas the CBM is also trained on the sampled interventions to increase its sensitivity to interventions determined by the learnt policy.

We show that this approach can outperform existing greedy policies for different budgets, achieving better model performance after a set number of interventions, while maintaining similar performance under the absence of interventions. However, we note that this approach also has its drawbacks, including high time complexity and the need for a good surrogate model.

To sum up, this project successfully develops a novel approach in using Reinforcement Learning to learn a non-greedy intervention policy for different budgets, where we successfully show that our RL-based approach learns non-greedy policies that outperform existing greedy policies for different budgets.

Chapter 2

Background

2.1 CBM

Concept Bottleneck Models (CBMs), initially proposed by Koh et al. [3], 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 $\mathbf{x} \rightarrow \mathbf{c}$ model" and f as the "label $\mathbf{c} \rightarrow \mathbf{y}$ 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 $\mathbf{x} \rightarrow \mathbf{c}$ model to improve the performance of the $\mathbf{c} \rightarrow \mathbf{y}$ 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 [3].

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 $L_{\text{label}}(f(\hat{\mathbf{g}}(\mathbf{x})), \mathbf{y})$

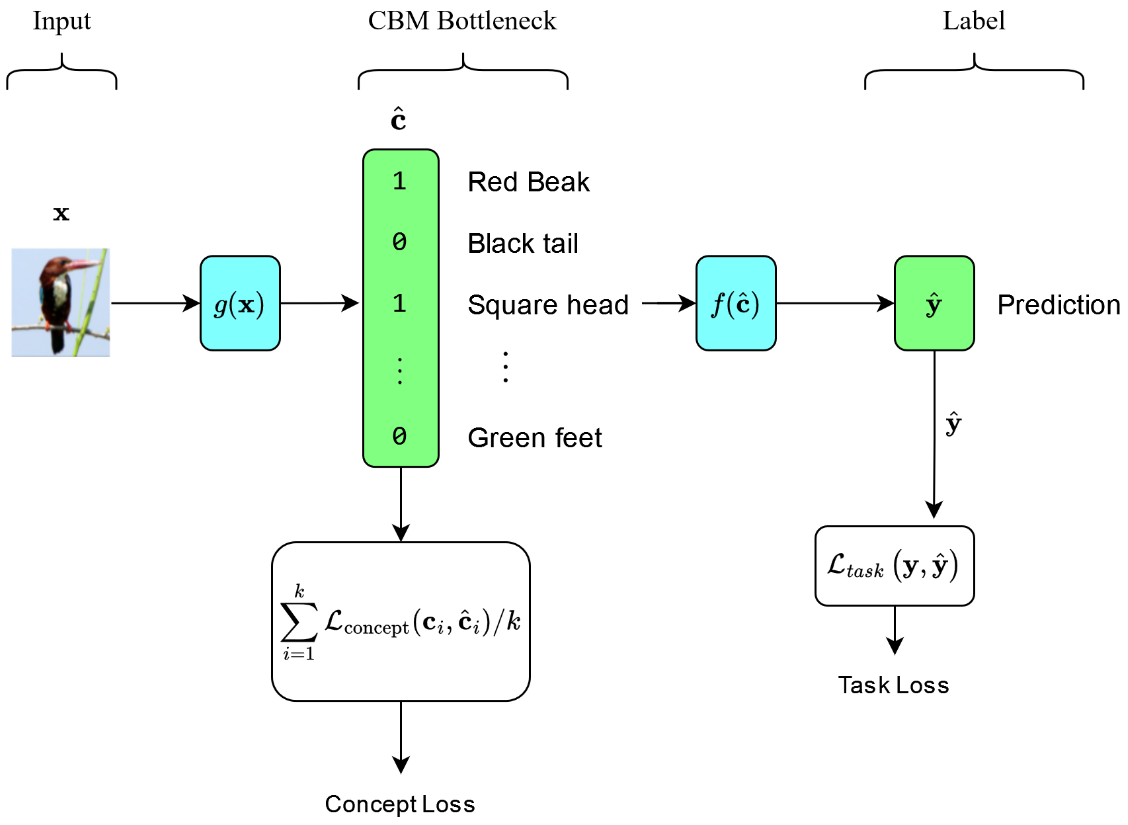

Figure 2.1: The CBM Architecture [3].

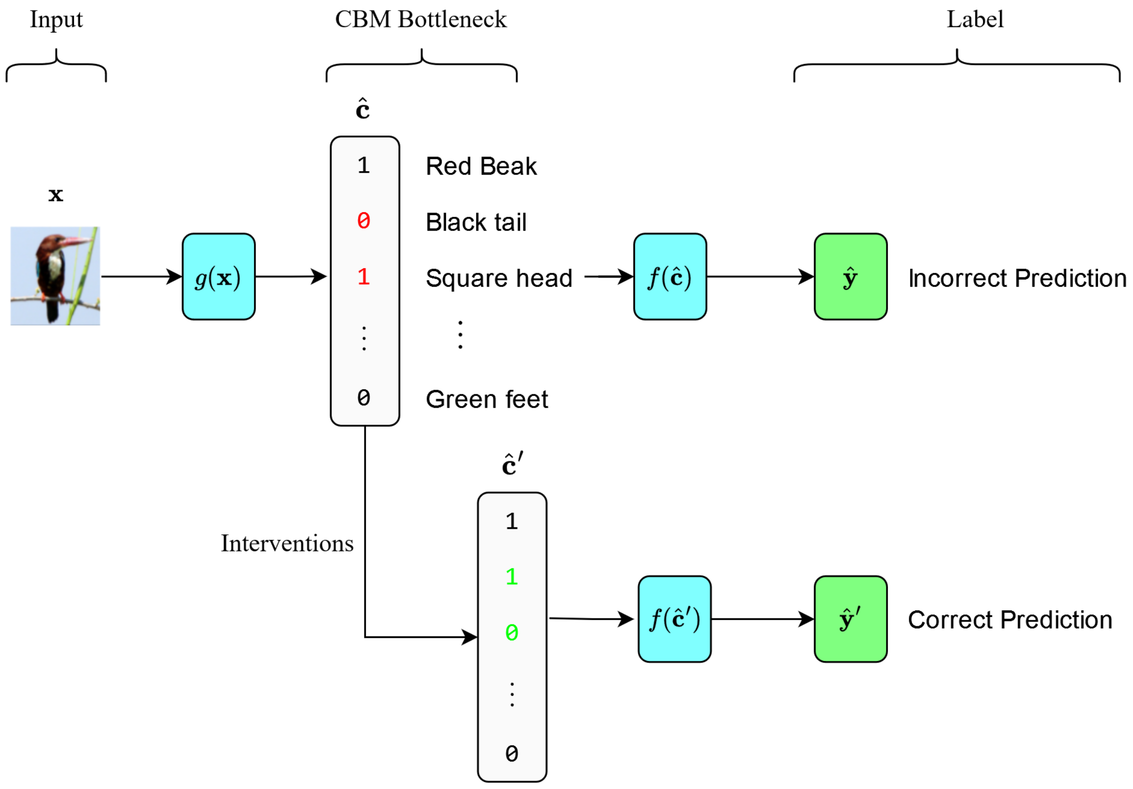

Figure 2.2: An illustration of intervening on the concepts predicted by a CBM.

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 [3].

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 [3].

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 [12]. 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. [12], 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 [12].

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 [12].

2.3 IntCEM

Building on top of CEMs, Zarlenga et al. [13] 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

2.5 Flow Models

Flow models model probability distributions by leveraging the change of variable property [9]. Given an input distribution $x \sim p_X(\mathbf{x})$, the change of variable property allows us to define deterministic invertible transformations $f : \mathbf{x} \rightarrow \mathbf{z}$, 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.

Chapter 3

Related Work

3.1 CooP

3.2 Active Feature Acquisition

Chapter 4

Method

4.1 Intervention Policies

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. Thus an intervention can be defined as 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 $\boldsymbol{\mu}$.

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

To formalize an intervention policy, we define an intervention policy $\mathcal{P}$ to be a policy, either learnt or heuristic-based, that determines the order of concepts to intervene on with the goal of maximizing the accuracy of the $\mathbf{c} \to \mathbf{y}$ 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 $\hat{g} : \mathbf{c} \to mathbfy$ model on the intervened concepts $\hat{\mathbf{c}}_{\mathcal{P}_j}$ and the label $\mathbf{y}$.

4.1.1 Non-greedy Intervention Policies

Compared to a greedy intervention policy, a non-greedy intervention policy outputs a set of concepts to intervene on for a given budget j, which we want to maximize the accuracy of the label predictor 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 important 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.

4.2 Surrogate Models

We use surrogate models to model the probabilities of concepts which are used to guide the RL model. Following Li et al. [5] which uses Reinforcement Learning in the similar problem of Active Feature Acquisition, we use a surrogate model to model the conditional probabilities $p(\mathbf{x}_u \mid \mathbf{x}_o, \mathbf{y})$, where $\mathbf{x}_u$ is a set of unacquired concepts, $\mathbf{x}_o$ is a set of acquired concepts, and $\mathbf{y}$ is the label. We select a variant of the popular normalizing flow models [9], namely Arbitrary Conditional Flow (AC Flow) [4] models as our surrogate models, which are flow models augmented with the power to model arbitrary conditional probabilities. While these AC Flow models are used to model the likelihoods $p(\mathbf{x}_u \mid \mathbf{x}_o)$, they can easily be extended to model the likelihoods $p(\mathbf{x}_u \mid \mathbf{x}_o, \mathbf{y})$.

4.2.1 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 are built on top of Transformation Autoregressive Networks [6] (TANs), and learn to model the probabilities for an arbitrary set of variables $p_X(\mathbf{x}_0, \ldots, \mathbf{x}_n \mid \mathbf{y})$, modelling the underlying latent distribution using an autoregressive approach with Recurrent Neural Networks (RNNs) [7]. The RNN learns to model the likelihood of $p_Z(\mathbf{z}_0, \ldots, \mathbf{z}_n \mid \mathbf{y})$ by sequentially processing each of the variables $\mathbf{z}_i$. At each step, the output of the RNN $\mathbf{h}_i$ is passed through a learnable linear layer to get parameters for an underlying Gaussian Mixture Model (GMM), a mixture of multiple different Gaussian distributions . which allows us to compute $p(\mathbf{z}_0, \ldots, \mathbf{z}_n \mid \mathbf{y})$ using a weighted sum of the probability density of

K Gaussian distributions. Experimentally we find that setting the number of components K to be the number of classes y can take, or using one Gaussian distribution for each class achieves a good balance between model performance and computational efficiency. This is discussed further in Section **??**. While using a GMM does not directly gives us the probability $p_Z(\mathbf{z}_0, \ldots, \mathbf{z}_n \mid \mathbf{y})$, it allows us to compute the probability density of the distribution, giving us the conditional likelihood of the set of variables $\mathbf{z}_0, \ldots, \mathbf{z}_n$, which provides valuable information to the RL agent, and also allows us to sample from the distribution.

4.2.2 Transformations

In order to transform $p(\mathbf{z}_0, \ldots, \mathbf{z}_n \mid y)$ to $p(\mathbf{x}_0, \ldots, \mathbf{x}_n \mid \mathbf{y})$, we utilize a set of transformations with learnable parameters. We follow the set of transformations defined by Li et al. [4] and implement the corresponding transformations. These transformations learn a transformation $q_{\mathbf{x}_o}$ that is conditional on observed features $\mathbf{x}_o$

This is combined with the transformations used in normalizing flows to model the probability density of $p(\mathbf{x}_0, \ldots, \mathbf{x}_n)$. AC Flow models build on top of this and model the conditional likelihoods $p(\mathbf{x}_0, \ldots, \mathbf{x}_n \mid \mathbf{y})$. This can then be used to compute the arbitrary conditional likelihood $p(\mathbf{x}_u \mid \mathbf{x}_o, \mathbf{y})$ which is the likelihood of seeing a set of unobserved features $\mathbf{x}_u$ given a set of observed features $\mathbf{x}_o$ and a class $\mathbf{y}$. By using Bayes' theorem, we note that

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

Which the right hand side terms can be computed using the AC Flow model. Due to the invertible property of the learnt transformations, a model learnt this way also allows us to sample from the data distribution by sampling from the underlying probability distribution then applying the transformations, which is useful for determining interventions as this provides information on which concepts are likely to be present (or not present) given the currently intervened concepts.

4.2.3 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. [5] 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, an MSE loss is incorporated to train the model to learn to sample the most probable unobserved features x_u given x_o, y to provide to the RL agent as auxiliary information. For x_o, y we obtain $\mathbb{E}_{p_X(x_u \mid x_o)}[x_u]$ by computing the expected latent features $\bar{z} = \mathbb{E}_{p_Z(z|x_o)}[z]$, and then inverting the conditional transformations to obtain the expected unobserved features $q_{x_o}^{-1}(\bar{z})$, then compute the MSE loss with the true unobserved features. The loss thus becomes

$$Loss = -\log p(x_u \mid x_o, y) + ||q_{x_o}^{-1}(\bar{z} - x_u)||^2$$

to train the model to learn about the likelihood distribution of concepts among classes, we add a cross entropy loss between the predicted label $\hat{y} = \text{argmax}_y\, p(x_u, x_o \mid y)$ and the true label. This helps ensure the model learn to scale the likelihood of concepts according to the label, and helps the model learn the probabilities $p(x_u \mid x_o, y)$.

4.3 RLCEM

4.3.1 Reinforcement Learning

We model the problem of finding a non-greedy intervention policy as a Reinforcement Learning problem. As mentioned in Section [**?**], 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 of finding an optimal non-greedy intervention policy as a Reinforcement Learning problem, we model an intervention trajectory as a Markov Decision Problem [10] according to the following definition:

- States are the observations of the model, including all the information that the is available at each step. This includes the state of the CEM, including its bottleneck and predicted concepts, the output of the surrogate model, including $p(x_u \mid x_o, y)$, $p(x_r \mid x_u, x_o, y)$, $p(x_r \mid x_u, x_o)$ as described in Section 4.2.

4.3.2 Active Feature Acquisition

In Active Feature Acquisition as described in Section 3.2, Li et al. [5] combine Reinforcement Learning with AC Flow models to give impressive results on finding the optimal features to acquire from the environment. This is done first by pre-training an AC Flow model that learns arbitrary conditional distributions about the underlying features $p(x_u \mid x_o, y)$. Then, A Reinforcement Learning agent is trained to learn the order of features 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. [5] 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 x_o, the agent has access to the sampled concepts x_u from the AC Flow model. This includes x_u sampled from $p(x_u \mid x_o, y)$ and $p(x_u \mid x_o)$, 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.

As such corresponding adjustments need to be made, which are illustrated in Sections**??** and **??**.

4.4 Models and Datasets

We follow Zarlenga et al. [13] and use the datasets MNIST-ADD, CUB, and CelebA for our experiments.

4.4.1 MNIST-ADD

4.4.2 CUB

4.4.3 CelebA

Chapter 5

Evaluation

5.1 Non-greedy policies

5.2 Surrogate Models

5.3 RLCEM Performance

5.4 Limitations

Chapter 6

Summary and conclusions

6.1 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] 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.

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

[5] 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.

[6] 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.

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

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

[9] 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.

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

[11] 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.

[12] 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.

[13] 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.

Appendix A

Dataset Details

Appendix B

Surrogate Model Details

Appendix C

Hardware Specifications

Appendix D

Training Graphs