Imperial College London

MEng Individual Project

Imperial College London

Department of Computing

Automatic Concept Extraction and Its Use in Explaining Video Classification

Author:
Roko Parać

Supervisors:
Dr. Alessandra Russo
Dr. Luke Dickens

Second Marker:
Dr. Robert Chatley

June 20, 2022

Abstract

For most tasks, fully neural architectures are undisputedly the best performing and most widely used models in machine learning. However, they are mostly uninterpretable, making them tricky to apply in a high-risk environment. Choosing an interpretable model, on the other hand, would yield a decrease in performance.

Recent work on concept bottleneck models [27] has managed to design a neural network model which overcomes the issue of non-interpretable models. It allows reasoning about a prediction with high-level concepts whilst achieving a competitive accuracy relative to its end-to-end counterpart. These concepts, however, need to be manually engineered and labelled to achieve such results.

In this project, we improve the concept bottleneck design by automatically mining concepts from text explanations and improving their interpretability. To do so, we design a domain-independent concept mining framework using state-of-the-art approaches from Inductive Logic Programming, Natural Language Processing and Deep Learning. We showcase that the new framework is a lot better at finding useful concepts than its predecessor, developed by Jeyakumar et al. [4], capturing twice as much information about the final labels.
In addition, we demonstrate that Inductive Logic Programming can be an effective framework for tackling sequence-to-sequence NLP tasks with few available examples, as we achieved Jaccard Index values of 0.55 and 0.85 for two challenging tasks.
Finally, to improve the concept bottleneck explainability, we designed a fully interpretable probabilistic logic-based classification framework. The framework outperforms the current end-to-end approaches for the MLB-V2E and the sudoku grid validity datasets.

Acknowledgements

I would like to express my sincere gratitude to my supervisors Dr. Luke Dickens and Dr. Alessandra Russo, for their academic guidance throughout this process. They helped me tremendously with their immense knowledge and commitment, and I am grateful to have had them as my supervisors.

My thanks and appreciations also go to my colleagues and friends, who have been a constant source of motivation and have enriched my student life.

Fundamentally, I am beyond grateful to my family. The completion of my studies could not have been possible without their unwavering support, love and encouragement.

Contents

Chapter 1

Introduction

Understanding why a machine learning model makes a particular prediction is always beneficial. It verifies that the model has learned a correct solution instead of a spurious pattern in the training data. Interpretability is extremely important when applying models in a high-risk environment, such as autonomous cars, where a cost of an error is extremely high.

Interpretable models do exist. Humans can easily interpret decision trees and logic-based models to understand why they produce a specific solution. However, they fail to reach the level of performance that end-to-end neural networks achieve.

Concept bottleneck models [27] are a novel set of models achieving comparable performance to end-to-end NNs while allowing the understanding of their prediction. They use neural networks to predict high-level human-engineered concepts before predicting the final label. As such, they allow reasoning about the final labels with intermediate concepts they have predicted.

In this project, we take the concept bottleneck idea further by mining a set of concepts directly from human-generated text explanations. In addition, we further improve the model interpretability by designing a logic-based classification pipeline from intermediate concepts to final labels.

1.1 Motivation

Natural language can be used as metadata for any sort of data. To make it machine-readable and valuable for a task at hand, one needs to extract relevant pieces of information about it consistently.

However, using textual information is a challenging task. Humans speak and write at varying levels of granularity, and a sentence may contain multiple relevant pieces of information. For instance, consider the following explanations of the same event:

```
The batter hit a fly ball but it was caught in the air by an
   outfielder.
The player struck a ball in the air. However, the ball was caught by
   a fielder before it touched the ground.
```

They convey similar semantic information in a very different syntactic manner.

The semantic parsing tasks showcase how challenging it is to deal with such a problem. These tasks aim to convert a sentence into a convenient form for a machine, such as the logical form, which they can then use for other downstream tasks. One instance of such a task is Abstract Meaning Representation Parsing [5], which converts sentences into a tree form that captures general semantic relations. The current state of the art [71] only achieves the F1 score of 0.817, allowing room for significant improvement.

The text we attempt to deal with might be even more general than what semantic approaches capture. We want to be able to use any declarative sentence stating the facts that have occurred. In addition, we want to mine concepts from text that preserve part of the semantics of the original sentence, that are syntactically structured and contain relevant information with respect to the label they are targeted to explain. The method that we envisage to develop has to be domain-agnostic in the sense that we can apply it to different texts for different classification tasks.

Moreover, the set of concepts should be clearly interpretable. For example, if the method were to extract a concept *ball*, it may be unclear what it refers to in the baseball domain. Does it refer to the event *ball*, which occurs when a pitcher throws the ball outside of the strike zone, or simply the *ball* used in a baseball game?

Taking into account the properties mentioned above, one of the goals of a project is to design a mostly syntactic concept mining framework capable of extracting concept sentences from text. The mostly syntactic nature allows the method to be applicable in various domains. The only semantic part involves grouping semantically equivalent sentences using transformers, which is also domain-independent.

1.2 Objectives

This project proposes a novel method that combines techniques from natural language processing, deep learning, and logic-based learning to develop an interpretable and general high-quality classifier. It builds upon the work by Jeyakumar et al. [4], which develops a concept mining framework trained with a concept bottleneck model for the baseball classification problem. It also provides novel solutions that are applicable to other domains. The project addresses the following high-level objectives:

- Develop a general framework for mining concept sentences — The goal is to extract concept sentences from declarative sentences successfully. Such a framework should capture information at various levels of granularity and be domain-independent.
- Improve the explainability of the concept bottleneck model — The existing approach provided the explanations by choosing the top three concepts with the highest attention score of a trained model. We aim to improve it by enhancing the explanation quality and simplifying the interpretation of the explanations.
- Explore the generality of the methods. — We explore whether our proposed solutions can be translated to different domains.

1.3 Challenges

Several key challenges need to be handled as part of this project in order to achieve the above objectives:

1. *How should a concept mined from natural language be defined?* Mining a concept that is helpful and immediately extensible to other domains may be challenging. It is hard because expert knowledge cannot be used to help craft features that the subsequent architecture should use.

2. *How should a sentence be decomposed into a concept sentence?* To allow a concept to be clearly interpretable and domain independent, we represent it in the form of a sentence. However, finding a solution that consistently converts a sentence into simpler ones that convey the same piece of information is challenging. The main challenge is doing it well for a diverse set of sentences.

3. *How can a concept mining pipeline be scaled up with a large amount of data?* Using logic-based learning for mining concepts from a sentence would allow the three properties of interpretability, different level of granularity of the information, and domain independence. This is because it is logic-based and therefore naturally interpretable and the search space for solutions is declaratively definable. However, logic-based learning systems are challenging to scale.

4. *How can a logic-based learning approach be used to learn dependencies between mined concept sentences and downstream labels to improve explainability?* Most logic-based learning systems do not have mechanisms for dealing with probabilistic atoms. Each atom can either be included or not be included in a solution. On the other hand, each concept may be mined or predicted with some probability p of it being true.

1.4 Contributions

The project addresses all the challenges identified above and provides the following main contributions:

- A method for mining concept sentences from declarative sentences that is domain-independent. (Chapter 3)
- Integration of a concept mining approach into a concept bottleneck model utilising human-written label explanations as text metadata of the labels. (Chapter 4)
- A fully interpretable, logical-based learning framework for a classification task using probabilistic facts. (Chapter 5)

Chapter 2

Background

2.1 Answer Set Programming

Answer Set Programming [37] is a form of declarative programming with a Prolog-like syntax suitable for solving NP-hard search problems. However, it is based on a different computation mechanism than Prolog: stable model semantics [19]. Answer set solver is a program that generates stable models of an answer set program, which are the solutions of an answer set program. The chosen answer set solver used throughout this project is clingo [10].

This section will briefly highlight the syntax of the answer set programs and the stable model semantics.

2.1.1 Syntax

Here are the types of rules in ASP[1]:

1. *normal rule* h :- $b_1, b_2, ..., b_n, not\ n_1, not\ n_2, ..., not\ n_o$
2. *hard constraint* :- $b_1, ..., b_n, not\ n_1, not\ n_2, ..., not\ n_o$
3. *choice rule* $lb\{h_1; ..., h_m\}ub$:- $b_1, b_2, ..., b_n, not\ n_1, not\ n_2, ..., not\ n_o$

where lb, ub are integers, $h_1, ..., h_m, n_1, ..., n_o$ are atoms and $b_1, ..., b_n$ are either atoms or aggregates.

Aggregates have the following syntax $lb\#agg\{e_1; ...; e_n\}ub$ where lb, ub are ints, agg is an aggregate function (e.g. sum) and e_i is an aggregate element. Additionally, aggregate element has the form $t_1, ..., t_k : c_1, ..., c_j$ where t_i is a term and c_i is a literal.

This syntax is quite expressive. For instance, here are two ways to define a coin falling either on heads or tails.

Method 1:

```
coin(c1).
```

[1] Disjunctive rules also exist, but they are omitted as they are not used in this work.

```
heads(C) :- coin(C), not tails(C).
tails(C) :- coin(C), not heads(C).
```

Method 2:

```
coin(c1).
1 { heads(C); tails(C) } 1 :- coin(C).
```

The results of these programs, i.e. the answer sets of the programs, are {coin(c1), heads(c1)} and {coin(c1), tails(c1)}. It will be explained how they are computed in the subsequent subsection.

As seen in the example, a program can have multiple answer sets. So, an additional piece of syntax is defined, which evaluates whether some answer set is better than another. This piece of syntax is referred to as a weak constraint. It is of the form $:\sim b_1, ..., b_n.[wt@lev, t1, ..., t_m]$ where lev is an integer, b_i a literal, and t_i term.

2.1.2 Semantics

The Answer Set Programming is based on stable model semantics [19].

The answer set is defined as follows: *Given a program P and a Herbrand interpretation X, X is an* ***answer set*** *of P iff X is a minimal Herbrand model of* $RG(P)^X$ *(reduct with respect to X).*

To understand the presented definition, one needs to know a few other concepts presented in this subsection. Here we will mainly explain the definitions for those concepts intuitively rather than formally; for deeper understanding and more formal definitions, please refer to the *Answer Set Programming* book [36].

Herbrand interpretation a program is an assignment of every element of the Herbrand base of that program to either true or false. **Herbrand base** of a program P is a set of all ground atoms that can be made using constants, predicate symbols, and functions of a program P.

When a Herbrand interpretation X satisfies all the rules of a program, X is the **Herbrand model** of a program. X is also a minimal Herbrand model if there is no smaller subset X', also a Herbrand model.

Relevant grounding replaces answer set program variables with constants available. It iteratively fills up the rules by adding every rule whose elements of $body^+$(R) are heads of already included rules. The $body^+$ constraint ignores the *not* atoms. For example, take P = {p(X) :- q(X). q(a).}. The first iteration of relevant grounding would create P' = {q(a).} while after the second one P' = {q(a). p(a) :- q(a).}

The **reduct of a program** P with respect to X (P^X) is a construct used to remove the *not c* terms from a program. Computing the reduct is done in the following manner. The solver assumes the X is a solution and iterates through every *not c* within P. If *c* is not in X (assumed to be false), then the *not c* term is removed from

the rule containing it. It essentially removes the need to worry about *not c* since that term is satisfied. On the other hand, if c is in X, then the entire rule containing *not c* is removed. There is no need to consider the rule whose body is false since it cannot be satisfied.

Constructing the reduct of a **choice rule** has an additional step. It is turned into either normal rules or a constraint. Given a possible interpretation X, the choice rule $lb\{h_1; ..., h_m\}ub$:- $b_1, b_2, ..., b_n, not\ n_1, ..., not\ n_o$ is converted in a following manner:

1. if $lb \leq |\{h_1; ...; h_m\} \cap X| \leq ub$ then h_i :- $b_1, b_2, ..., b_n, not\ n_1, ..., not\ n_o$ for is created for each $i \in \{1..m\}$

2. Otherwise, a constraint of the form :- $b_1, b_2, ..., b_n, not\ n_1, not\ n_2, ..., not\ n_o$ is created.

So, to check whether X is an answer set, one needs to compute a relevant grounding of a program. Then it needs to construct a reduct of that program with respect to X. From the reduct, one can easily construct a minimal model M by starting from facts and iteratively adding heads of those rules with all body elements already in M. Finally, if X = M, then X is an answer set. Note that the head of a **constraint** is considered to be $\bot$. Hence, if the body of a constraint is satisfied, $\bot$ is added to M, so it cannot be equal to X. In this manner, the constraints eliminate possible answer sets.

2.2 Inductive Logic Programming

Inductive Logic Programming [47] is a field at the intersection of machine learning and logic programming. In most cases, the goal of an inductive learning task is to learn a set of rules (a hypothesis H), which combined with the background knowledge B, can entail every positive example while not entailing any negative example. Some systems use a weakened version of that statement to allow for noisy examples, such as the newer versions of ILASP [31].

Many ILP systems have been developed over the years, such as Progol5 [48], HAIL [58], TopLog [49], and TAL [11].

The chosen systems for this project are ILASP [33] and FastLAS [34], systems that do the inductive learning of the answer sets programs. Choosing a system that learns ASP is done because the ASP environment can effectively perform non-monotonic reasoning.

The two chosen systems have a very similar syntax, with FastLAS developed as a more scalable alternative to ILASP. However, FastLAS is less general than ILASP for that reason.

We first introduce ILASP before highlighting the differences FastLAS has. To understand the ILASP system fully, a few more definitions need to be introduced.

An atom A is **bravely entailed** if by a logic program P if it is included (true) in at least one answer set of P. On the other hand, an atom A is **cautiously entailed**

if it is included (true) in all answer sets of P.

A pair $E = \langle E^{inc}, E^{exc} \rangle$ ($\langle$inclusions, exclusions$\rangle$) is a partial interpretation of sets of literals. Answer set X extends E iff set of inclusions is a subset of X ($E^{inc} \subseteq X$) and X is disjoint with exclusions ($E^{exc} \cap A = \emptyset$).

The **language(inductive) bias** M, is a a pair $\langle M_h, M_b \rangle$ which is used to construct the **search space** S_M of a learning task. The rule of the form
h :- $b_1, b_2, ..., b_n, not\ n_1, not\ n_2, ..., not\ n_o$ is in S_M iff:

1. h is compatible with M_h. This either means that h is an atom compatible with M_h, an aggregate $lb\{h_1, .., h_m\}ub$ where each atom h_i is compatible with M_h or h is empty.
2. b_i and n_i are compatible with M_b.
3. every variable appears in at least one positive body literal (the rule is safe).

Understanding what compatible refers to in the previous definition is the easiest through the means of an example.

Inductive bias example

```
#modeh(heads(var(coin))).
#modeh(tails(var(coin))).
#modeha(heads(var(coin))).
#modeha(tails(var(coin))).

#modeb(heads(var(coin))).
#modeb(tails(var(coin))).
#modeb(coin(var(coin))).
#modeb(coin(const(coin))).

#constant(coin, c1).
```

The example above is a possible definition of the inductive bias for the simple coin problem written in ILASP-compatible syntax. The `#modeh` directive specifies that an atom can occur in the head of the rule. In this case, it defines that `heads(X)` is compatible with M_h. Similarly, the `#modeha` defines that an atom can occur in the aggregate of the rule. Hence, it determines that $lb\{..., heads(X), ...\}ub$ is a head compatible to M_h. The `#modeb` syntax defines that an atom can occur in the body either as a positive or negative atom. Finally, the `#constant` defines constants that can replace const functions.

Let S_M be the search space constructed from the language bias M, B background knowledge, E^+/E^- set of positive/negative examples, T = $\langle B, S_M, E^+, E^- \rangle$ is the **learning from answer set task** (ILP_{LAS}) [30]. A hypothesis H is an inductive solution of that task iff:

1. It is a subset of search space S_M.
2. For every negative example e^-, there is no answer set A of the program $B \cap H$ which extends that example.

3. For every positive example e^+, there is an answer set A of the program $B \cap H$ which extends it.

We write $H \in ILP_{LAS}$ when H satisfies the above criteria. Notice that the positive examples are bravely entailed while the negative are cautiously entailed.

Examples in ILASP

```
#pos(id1@1,
{heads(c1)},
{tails(c1)},
{coin(c1).}
).

#neg(id1@1
{tails(c2), heads(c2)},
{},
{coin(c2).}
).
```

The shown piece of code is a simple definition of examples in ILASP. Both positive and negative take either take the same set of parameters, id@`noise_penalty`, the set of inclusions E^{inc}, the set of exclusions E^{exc}, and the context of an example (C_e). The context of an example, C_e is a set of rules and facts that are only relevant when explaining the example e [6]. It is treated as an addition to the background just for that example. Moreover, the `noise_penalty` is an optional term introduced to deal with possibly noisy examples. When it is included, ILASP may choose between covering an example and paying the `noise_penalty` for not covering the example. The optimal solution then becomes the one which minimises the sum of all uncovered example penalties and the hypothesis length [32]. This sum is also referred to as the score/penalty of the ILASP solution.

FastLAS

FastLAS [34] is much more scalable with respect to hypothesis space size than ILASP, but the tasks it can solve are less general than ILASP's. Namely, the crucial reason for using both is FastLAS's inability to deal with tasks that have recursive rules [35].

However, FastLAS also allows specifying domain-specific optimisation criteria, unlike ILASP. The hypothesis H it returns is optimal with respect to a scoring function $S(H, T)$ where T is a learning task.

One can define not any scoring function. The scoring function $S(H, T)$ must be **decomposable**. It should should be possible to define it as a sum of scoring functions applied to each rule, i.e. $S(H, T) = \sum_{r \in H} S^{rule}(r, T)$.
For example, the most common scoring function, and the one used by ILASP, uses hypothesis length ($S_{len}(H, T) = |H|$). The decomposition of that function is $S^{rule}_{len}(r, T) = |r|$ since the sum of rules lengths is equal to the length of the entire hypothesis. This decomposition is written in FastLAS code using the `penatly` predicate in the following manner:

```
#bias("penalty(1, head(X)) :- in_head(X).").
#bias("penalty(1, body(X)) :- in_body(X).").
or equivalently:
#bias("penalty(L, custom)
        :- L = #count{X : in_head(X); X : in_body(X)}.").
```

To account for noisy examples, a special scoring function that FastLAS allows is $S_{pen} + S$, where S_{pen} is the sum of example penalties and S is a decomposable scoring function.

The FastLAS has a practically identical syntax as ILASP with only two slight differences. One needs to explicitly define `#modeb(not p)` if it is possible for `not p` to occur in the solution. Moreover, the constants need not be explicitly defined with `#constant`. They are instead inferred from the background.

2.3 Natural Language Processing

The critical requirement of this project is to extract syntactic concept sentences from the text corpus. This section briefly overviews the techniques applied to make the concept extraction possible.

The techniques presented in the following section are implemented by a popular library *spacy* [60] which has been utilised in this project. The language model currently used throughout the project is `en_wb_gl_lg`, the most performant CPU-optimised option.

2.3.1 Pre-processing/Post-processing Techniques

Tokenisation [26] separates the given text into smaller units named *tokens*. It is common to split English text into words as tokens, as they carry meaning and are easy to extract. The *spacy* also extracts punctuation as separate tokens.

Sentence splitting is a similar problem as tokenisation. It splits a text corpus into sentences before they are processed individually.

Part-of-speech tagging [26] attempts to determine which tag a word has in the sentence. The problem is much more complicated than the ones previously described as the part of speech can often be dependent on the meaning of the word in a sentence. For example, the sentence *I play the main character in a play* highlights how the word *play* can have two identically written words with different meanings and parts of speech. The first occurrence of *play* is a verb, and the second is a *noun*.

The spacy library uses neural networks and statistical models to determine which tokens the library should assign which POS tag [62]. The classifier which spacy uses has a very high accuracy for the English language, with accuracy between 97-98% depending on the model [61].

Truecasing [38] is a process for determining the appropriate capitalisation of a word where such information is unavailable. As outlined in [40], capitalisation is required

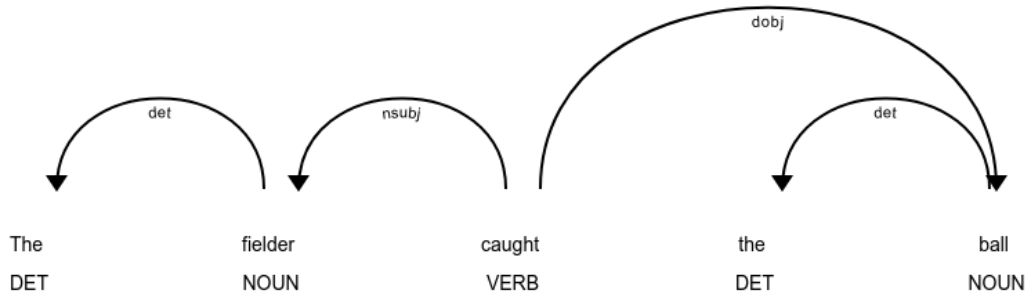
for all proper nouns and first words of sentences. The latter is simple to implement, while the former heavily relies on a part-of-speech tagger. The spacy library does provide a POS tagger that assigns Penn Treebank [46] tags to words. This tagger can capture whether a noun is proper by using *NNP* (proper noun, singular) and *NNPS* (proper noun, plural) tags.

2.3.2 Parsing

Dependency Parsing

Dependency Parsing is a form of parsing that attempts to determine words' dependency within a sentence [26].

Figure 2.1: Dependency graph of *The fielder caught the ball.*



The resulting structure obtained is a dependency parse tree, which is acyclic and rooted because it is a tree. An example of a dependency parse tree can be seen in 2.1. This structure enables determining which purpose a word serves in a sentence. To illustrate why determining what purpose a word serves in a sentence may be helpful, here is a predicate/subject refresher: *The subject is a person/object who/what the sentence is about, while the predicate tells what the subject is or what it is doing* [63]. From this definition, it is clear that one can easily extract an actor in a sentence if they know what it is doing. The dependencies produced by the *spacy* dependency parser are more precise as they follow Universal Dependencies. For instance, those dependencies distinguish between nominal and clausal subjects. Please refer to the Universal Dependencies to see a complete list of the dependencies available if needed.

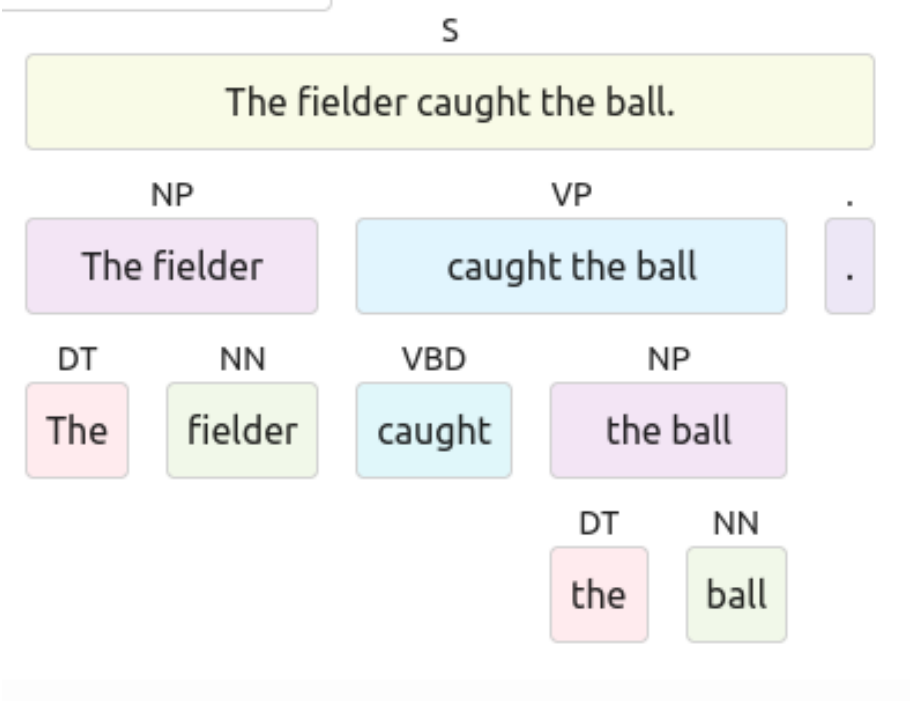
To evaluate the performance of a dependency parser, one uses labelled attachment score (LAS) and unlabelled attachment score (UAS) accuracy [26]. The former validates how well a word is assigned to its head and whether a correct dependency relation is assigned. The latter score checks whether the head is assigned correctly. Spacy's dependency parsing system slightly trails behind the state-of-the-art approaches at the moment, with its UAS and LAS being at 95.1% and 93.7%, respectively. Mniri et al. [46] developed the current state-of-the-art model, with UAS/LAS scores at 97.4% and 96.3%.

Constituency Parsing

Constituency Parsing is a form of syntactic parsing which assigns a structure to a sentence created by using a context-free grammar [26]. This approach aims to group words into constituents, a group of words that behave as a single unit. For example, one may group a sequence of words surrounding a noun into a noun phrase such as *the Imperial students*. Words grouped into constituents can be used as an intermediate form for semantic analysis and checking whether a sentence is grammatically correct. In the context of this project, it was utilised by the inherited work presented in 4.1.

The resulting structure produced by constituency parsing is a parse tree. An example of a tree parsed by a constituency parser can be seen in figure 2.2.

Figure 2.2: Parse tree of *The fielder caught the ball.* as generated by spacy Berkley Neural Parser [61].



Chapter 3

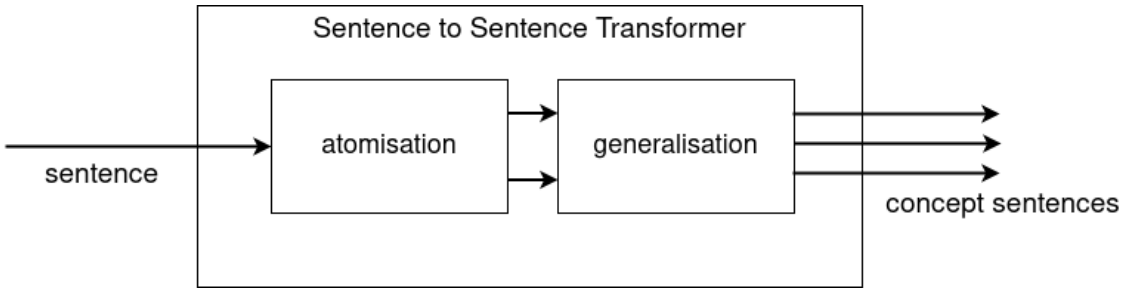
Solving NLP Tasks Logically

Logical representations have an important place in the Natural Language Processing field. Semantic parsing [26] is a famous task that aims to convert a natural language input that captures the meaning of that input. After all, having a sentence in a logical form allows reasoning about it to form new conclusions.

However, in this chapter, the logic is used in a slightly unconventional manner for NLP. It is an intermediate representation for a sequence-to-sequence problem where it captures only the syntax of the text rather than its meaning. The current state-of-the-art approaches for sequence-to-sequence problems, such as language translation, all use transformers. Nevertheless, seq2seq transformers would perform poorly with available datasets for the two problems tackled in this chapter. The datasets consisting of 100 examples are too small for a model with billions of parameters to be able to generalise. Transformer fine-tuning is usually done on tens of thousands of examples, such as the IMDB Movie Reviews dataset [41].
Logic-based learning systems, on the other hand, can generalise well from few examples [28], making them suitable for the problems presented in this chapter.

In this chapter, we demonstrate how to solve two seq2seq NLP tasks using logical approaches, which constitute a crucial part of our concept bottleneck model (Chapter 4). They are combined into a sentence-to-sentence transformer (note the difference between the standard transformer), shown diagrammatically in the figure 3.1.

Figure 3.1: Diagrammatic representation of the sentenece2sentence transformer



3.1 Sequence2Sequence Tasks

The two tasks that have been tackled using logic-based learning approach are **sentence atomisation** and **generalisation**.

The former converts a declarative sentence into one or more **atomic sentences**, while the latter converts an **atomic sentence** into one or more **concept sentences**.

Atomic Sentence: A sentence that cannot be decomposed into multiple syntactically well-formed sentences. We consider a sentence to be syntactically well-formed if it follows the grammar rules of the English language. For example, the sentence *The batter swung and missed* can be split into sentences *The batter swung* and *The batter missed*
Note that this is equivalent to the definition of the atomic formula used in logic [24]. However, the sentence structure considered in this project is often much more complex than the one considered in logic.
An **atomic sentence** should only contain simple predicates, eliminating compound and complete predicates from a sentence.

Concept Sentence: A syntactic generalisation of an atomic sentence, which satisfies the following three conditions:

1. It is a syntactically well-formed sentence in its own right.
2. True if the atomic sentence is.
3. Obtained only through syntactic manipulation of an atomic sentence, a result of modifying the syntax tree of the sentence.

Concept sentences are sometimes referred to in this report as **(syntactic) generalisations** of a sentence.

Example 1. Splitting a given sentence into all its concepts sentences.

Starting from the sentence:

```
The batter caught the ball in the air and sent it into the left field.
```

we can extract the following atomic sentences:

```
The batter made contact with the ball in the air.
The batter sent it into the left field.
```

From these two sentences, we can obtain four concept sentences:

```
The batter made contact with the ball in the air.
The batter made contact with the ball.
The batter sent it into the left field.
The batter sent it into the field.
The batter sent it.
```

These two tasks (i.e. the sentence-to-sentence transformer) serve as a replacement for the extraction part of the original CoDEx (Concept Discovery and Extraction) pipeline (4.1).

Notice that both tasks are purely syntactic; one requires no understanding of the sentence to atomise/generalise it. The benefit of a syntactic approach is its domain independence, allowing it to be seamlessly applied to sentences with a completely different meaning. In addition, the reason the project aims to extract all concept sentences from a particular sentence is two-fold:

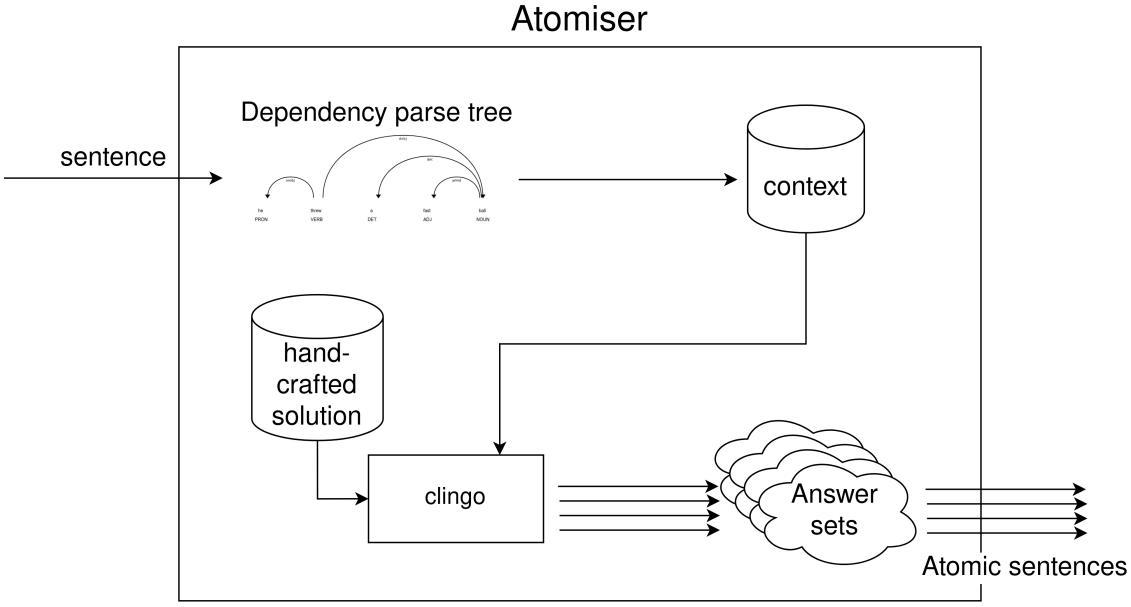
1. Concept sentences help associate differently worded explanations of the same concept.
2. The final generated sentences are immediately usable for explanations.

3.2 Solving Atomisation Task

The atomisation task is the more challenging of the two. The reason is the diversity of the input the model must handle, both with respect to size and the types of tokens encountered. The atomiser should ideally be able to process any declarative sentence, whereas the generaliser expects an atomic sentence as input, which is significantly easier to handle.

To tackle this task, we have constructed a hand-crafted solution whose application is shown in 3.2.

Figure 3.2: Diagrammatic representation of the atomiser



In this section, we will introduce the steps the atomiser consists of and demonstrate the idea behind a hand-crafted solution.

3.2.1 Logical Encoding of a Sentence

All of the logical encodings presented are compatible with the Answer Set Programming paradigm (2.1), a popular paradigm that deals with negation as failure well.

We first need to define a way to convert a sentence into a logical representation that we could use to reason about the sentence structure. Ideally, a sentence representation should satisfy the following criteria:

- Word dependency capture — The representation should capture syntacic dependencies between words to determine whether a word is crucial to the meaning of a sentence.
- Similar meaning → similar encoding — Slight word variations such as words replaced by synonyms should be encoded similarly. The task becomes easier for the learner when the exact representation captures the words with the same meaning.
- Compactness — Smaller representations are quicker to process.
- Domain independence — We want to apply the generalisation task in various domains, so the representation should not contain domain-specific information.
- Interpretability — It should be clear what the representation encodes. We can translate the learned ILASP solution into English if the predicates are interpretable. Hence, we can verify whether the system learned spurious correlations or valuable rules.
- Reconstuctability — One should be able to reconstruct a sentence as the final output of the task needs to be a syntactically well-formed sentence.

A common approach to encoding a sentence involves using dense-vector contextualised embeddings of words, such as the ones produced by word2vec. Dense-vector embeddings are practical because they are compact and tend to capture the semantics of words (i.e. map similar words to similar value embeddings). Transformers improve upon these embeddings by using the self-attention mechanism to provide an even better representation of a sentence.

However, dense-vector embeddings are not interpretable, making them difficult to use with our problem. The learning approach that we have used follows a similar idea of trying to capture semantic relationships within a sentence. We utilise the dependency parse tree (2.3.2) as a basis for the generalisation task as it captures syntactic relationships between words. Note that the syntactic relationships captured approximate the semantic relationships between words. In addition, words themselves are put into logical predicates, making the sentences reconstructible.

The dependency tree of a sentence gives rise to the following predicates:

```
dep(l, token1, token2).
root(token).
token(token, string).
```

It represents that there exists an arc from `token1` to `token2` with label `l` which are converted back to string form with `token` predicate. In addition, `root` encodes which token is the root of the sentence.

Example 2. Encoding *the batter swung and missed*, with a dependency graph shown in 3.3:

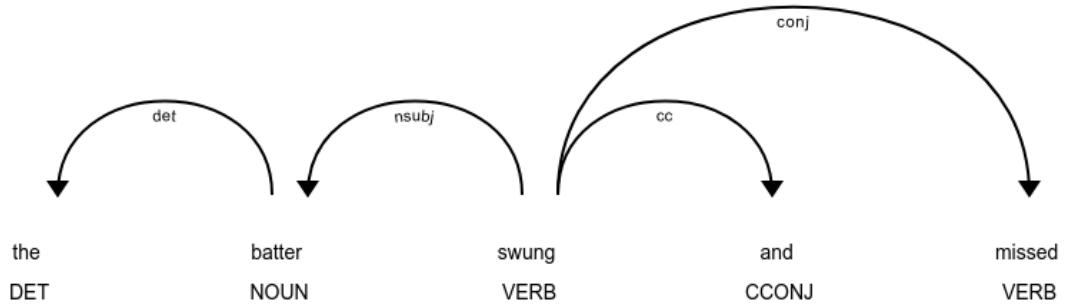
1. Convert each token in the tree to the logical form:

```
token(tok0, "the").
token(tok1, "batter").
token(tok2, "swung").
token(tok3, "and").
token(tok4, "missed").
```

2. Convert arcs and roots of the tree to predicates:

```
root(tok2).
dep(det, tok1, tok0).
dep(nsubj, tok2, tok1).
dep(cc, tok2, tok3).
dep(conj, tok2, tok4).
```

Figure 3.3: Dependency graph of *the batter swung and missed.*



Reflecting on the criteria outlined at the start of the section, we can see that it is mainly satisfied by the encoding. Even the similar meaning → similar encoding is captured somewhat in the context of this problem. The two sentences which only differ by one synonym would be identically represented by the `dep` predicates, resulting in any model treating them similarly. For example, the sentences:

`The batter hit the ball. The hitter hit the ball.`

would have the equal `dep` predicate representation. However, the sentences:

`The batter hit the ball. The ball was hit by the batter.`

would not have similar representations. This representation drawback was not a hurdle for the current dataset.

Encoding the Goal

By observing the examples, it is clear that atomic sentences only contained words used in the starting sentence. So, we could model the problem in this section as to whether or not we want to include the word in an atomic sentence. That goal is denoted with a predicate:

```
in_atomic_sent(t).
```

which represents that a token `t` is included in the concept sentence. The possibility of multiple concept sentences existing is modelled using multiple answer sets.

Example 3. Encoding *the batter missed* as atomisation of *the batter swung and missed*

From the previous example (2), we have:

```
token(tok0, "the"). token(tok1, "batter"). token(tok2, "swung").
token(tok3, "and"). token(tok4, "missed").
```

So, the example target is encoded as:

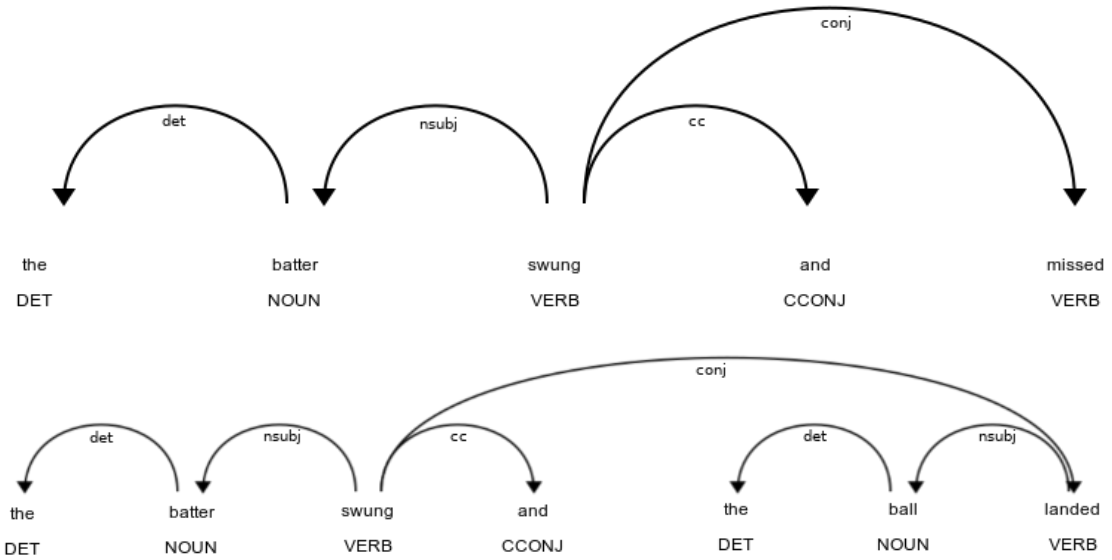
```
in_atomic_sent(tok0). in_atomic_sent(tok1).
in_atomic_sent(tok4).
```

3.2.2 Hand-crafting the Atomisation Solution

Consider the following two sentences and the desired atomisation, as well as the dependency graphs of the premises (figure 3.4):

```
the batter swung and the ball landed
  → the batter swung. the ball landed.
the batter swung and missed
  → the batter swung. the batter missed.
```

Figure 3.4: Dependency graph of two sentences which need to be atomised



The graph illustrates that the terms on both ends of the `conj` tag should appear in separate sentences. Determining splitting tags, tags whose words should be in distinct subsets, is the core idea of the hand-crafted solution. The words at the edges of these tags become starting points for generating sentences, which are constructed by adding tokens related to current `in_atomic_sent` tokens.

The other rules are constructed to add the other necessary sentence tokens. These may sometimes involve "jumping over" the words at the edge of the splitting tags. For example, the sentence `The batter swung and missed` needs to add the associated subject to the `missed` token to construct a valid sentence.

Following such an approach, we design the following set of rules:

Summary of the predicates used	
Predicate name	Explanation
splitting_tag(Tag)	Tag whose ends should be a part of different sentences
candidate_start(Token)	Token at the edge of a splitting tag
in_atomic_sent(Token)	Token which should be included in the atomic sentence
root(Token)	Root token of a dependency tree
dep(Tag, TokenFrom, TokenTo)	Arc of a dependency tree
adjacent_subj	Represents whether the current sentence has a subject next to one its tokens
do_not_include(Tag)	Excludes children arcs with a specific Tag from an atomic sentence.

```
% Capture tokens at the end of a splitting tag
candidate_start(T) :- splitting_tag(C), dep(C, T, _).
candidate_start(T) :- splitting_tag(C), dep(C, _, T).

% Start splitting tags. Splitting tags must be a part of
% distinct atomic sentences.
1 { in_atomic_sent(T) : candidate_start(T) } 1.

% Root should be a starting point if there are no
% splitting tags.
in_atomic_sent(T) :- root(T), not candidate_start(_).

% Tags linking tokens that should be in distinct answer
% sets, each with an example which sparked the choice:

% The batter swung therefore it is a strike.
% → The batter swung. It is a strike.
splitting_tag(ccomp).
% The batter did not swing so it was a ball.
% → The batter did not swung. It is a ball.
splitting_tag(advcl).
% The batter swung the bat but missed the ball.
% → The batter swung the bat. The batter missed the ball.
splitting_tag(conj).
% The batter hit the ball in play where it was caught
% mid air by a defender.
```

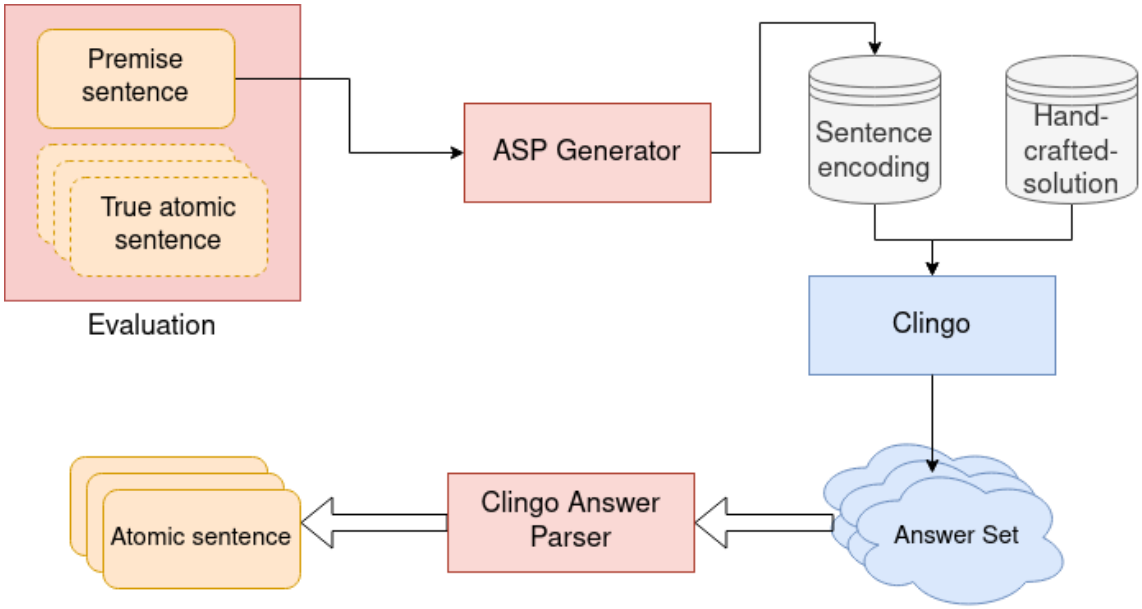
```
% → The batter hit the ball in play. It was caught mid air by a
    defender.
splitting_tag(relcl).



% Include all incoming relationships except candidate_starts.
% This allows us to reach the predicate of the current atomic
    sentence.
% Atulve's ball was fast and good.
in_atomic_sent(T) :- dep(_, T, T2), in_atomic_sent(T2), not
    candidate_start(T).


% Incoming relationships first conjunct (conj) should also be
    included for the second one.
% This holds for conj only. The clauses tend to be self-sufficient.
% Atulve's ball was good and quick. → Atulve's ball was quick.
    Atulve ball was good.
in_atomic_sent(T) :- dep(_, T, T2), dep(conj, T2, T3), in_atomic_sent
    (T3), not candidate_start(T).


% Include all children tags apart from those that are blacklisted (we
     do not want and, therefore...)
% Additionally, we do not want to include a candidate_start token.

% (Therefore) it is a strike
do_not_include(advmod).
% The batter hit the ball (and) it landed far away.
do_not_include(cc).
% Not including punctuation, it should not be in atomic sentences
do_not_include(punct).
% The umpire ruled (that) the batter did not swing.
do_not_include(mark).
% Skip all the splitting tags
do_not_include(C) :- splitting_tag(C).
in_atomic_sent(T) :- dep(C, T2, T), in_atomic_sent(T2), not
    do_not_include(C), not candidate_start(T).



% Every atomic sentence should have a subject.
% Include the subject of the first conjunct as a part of the second
    sentence if it does not contain its own.
% The batter swung but missed the ball → The batter swung. The
    batter missed the ball.
in_atomic_sent(T) :- dep(nsubj, T1, T), dep(C, T1, T2), splitting_tag
```

```
    (C), in_atomic_sent(T2), not adjacent_subj.

% There is no subject next to any currently included token
adjacent_subj :- dep(nsubj, T1, T2), in_atomic_sent(T1).
adjacent_subj :- dep(nsubjpass, T1, T2), in_atomic_sent(T1).
adjacent_subj :- dep(csubj, T1, T2), in_atomic_sent(T1).
adjacent_subj :- dep(csubjpass, T1, T2), in_atomic_sent(T1).
```

3.2.3 Full pipeline

The diagram of a whole process, starting from a given sentence to one or many atomic sentences, is shown in the figure 3.5.

Figure 3.5: Dependency graph of two sentences which need to be atomised



We showcase how it works through an example.

Example 4. Generating all of the atomisations of the sentence *The batter swung and missed.* at test time.

The ASPGenerator module carries out the first two steps:

1. Remove . and lowercase all the words in the sentence as a part of pre-processing. The string *the batter swung and missed* is the result of the operations.

2. Convert the sentence to the logical form, as shown by the example 2.

The result from step 2 is persisted to the file system as clingo [10], an answer set solver, is a command line tool.

3. The answer set solver is applied with atoms and predicates from the hand-crafted solution (3.2.2) and the sentence encoding. The resulting clingo output is:

```
Answer set #1:
{in_atomic_sent(tok0), in_generalised_sent(tok1),
 in_atomic_sent(tok2)}.

Answer set #2:
{in_atomic_sent(tok0), in_atomic_sent(tok1),
 in_atomic_sent(tok4)}.
```

The Clingo Answer Parser does the last two steps:

4. For each answer set generated, we reconstruct a sentence. The sentence reconstruction is done by converting each `in_atomic_sent` token to the back to its string representation. They are then joined in the same order they originally appeared.

For instance, the `Answer set #2` converts the tokens tok0 → "the", tok1 → "batter", tok4 → "missed". They are joined to a sentence *the batter missed*

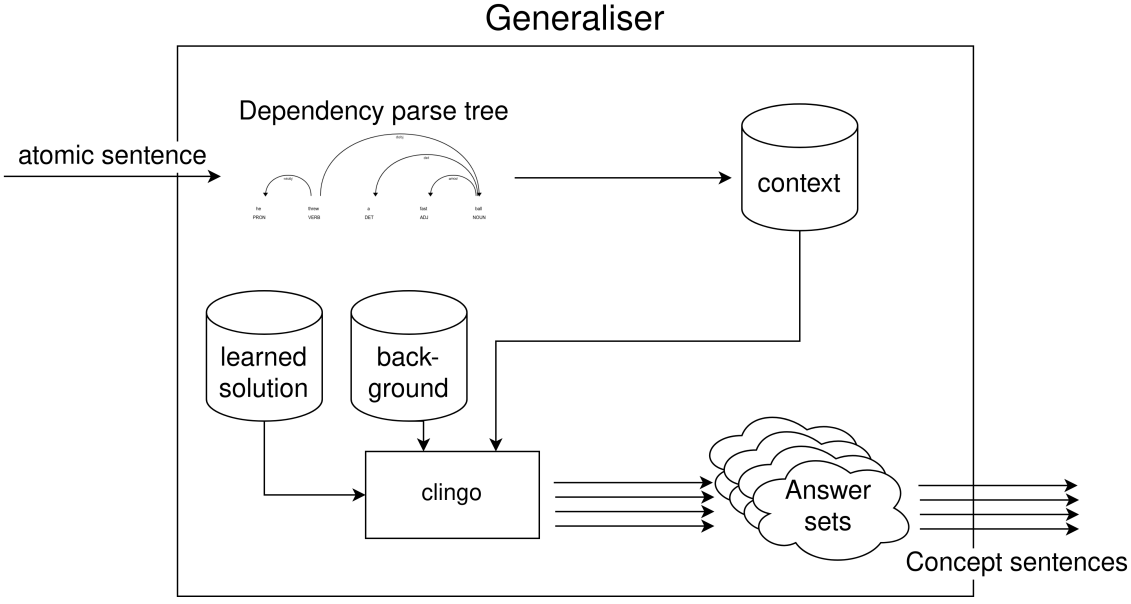
5. Post-processing clean-up involving truecasing (2.3.1), which determines the correct word capitalisation, and adding punctuation results in *The batter swung.* and *The batter missed.*

3.3 Solving Concept Generalisation

The concept generalisation task aims to find a concept sentence from given an atomic sentence. Recall that a concept sentence is the one obtainable from an atomic sentence through syntactic manipulation. It must be true if the atomic sentence is.

To tackle this task, we have learned a solution whose application is shown in 3.6.

Figure 3.6: Diagrammatic representation of the atomiser



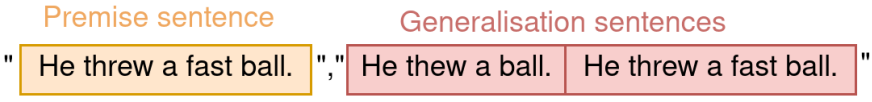
The rule learning is done using ILASP (2.2), a state-of-the-art ILP system.

3.3.1 Encoding the Learning Task

Learning a solution with ILP is different from determining a solution with Deep Learning. Inductive Logic Programming uses a different level of knowledge representation commitment than Deep Learning [16]. Deep Learning approaches do not give any inductive bias to a machine. The model must learn how to solve the problem from the data only. On the other hand, Inductive Logic Programming requires more human-generated knowledge representation. In particular, the logical structure is encoded to the learner, which is encouraged to find all possible theories within that structure and return the best one. Because of this property, the Inductive Logic Programming paradigm can incorporate existing knowledge into the final solution, making the task easier to solve.

Encoding the Example Premise and Target

Generalisation examples consist of a given sentence followed by one to many sentences, which are ways in which the provided sentence can be generalised. For instance, here is a possible generalisation example:



The example premise encoding is equivalent to the procedure for the atomisation task demonstrated in the example 2. Moreover, the example target encoding is also equivalent to the atomisation procedure showcased in the example 3. The only difference is that the target is represented by the `in_generalised_sent` rather than the `in_atomic_sent` predicate.

Background Knowledge Construction

Here is the background knowledge used for the generalisation task.

```
token(T) :- root(T).
token(T) :- dep(_, T, _).
token(T) :- dep(_, _, T).
label(L) :- dep(L, _, _).

% there must be a token in a concept sentence
:- #count{T : in_generalised_sent(T)}0.
```

The first four rules define unary predicates used as types in language bias, while the final rule encodes that a generalised sentence should not be empty.

Language Bias

The problem is modelled as a choice of whether a particular token (word) should always be included or sometimes be included in the solution, which the following types of rules can represent.

```
in_generalised_sent(T) :- ...
0 {in_generalised_sent(T) } 1 :- ...
```

This approach gives rise to the following language bias:

```
#modeh(in_generalised_sent(var(token))).
#modeha(in_generalised_sent(var(token))).

#modeb(in_generalised_sent(var(token))).
#modeb(root(var(token))).
#modeb(dep(const(label), var(token), var(token)), (positive)).

% The full set of labels is available at:
% https://universaldependencies.org/u/dep/
#constant(label, acl).
... all the constant declarations ...

#bias("
% Only allow rules 0 { in_generalised_sent(V1) } 1.
% Eliminates all generated rules that where lhs of
% the curly brace is >= 1.
:- in_head(_), lb(1).
").

% Disallow 2+ in_generalised_sent predicates occurring within {}.
#disallow_multiple_head_variables.
```

Example Encoding

As mentioned, generalisation examples consist of a given atomic sentence followed by 1 to many concept sentences. We need to convert those representations into ILASP examples. ILASP allows defining two types of examples as outlined in 2.2.

- positive (`#pos`) - an example which should be extended by at least one answer set.
- negative (`#neg`) - an example that any answer set should not extend.

Ideally, we want the solution learned by ILASP to produce the number of answer sets equal to the number of generalisations. Using these two constructs, we can define that we want precisely one answer set for each possible generalisation.
It is done by creating a positive example for each sentence we wish to produce. Just having positive examples is not sufficient. If we only had positive examples, the model could produce the following solution to satisfy all ILASP examples:

```
0{ in_generalised_sent(T) }1 :- dep(_, T, _).
0{ in_generalised_sent(T) }1 :- dep(_, _, T).
```

This solution will return a power set of all possible token combinations since all tokens are associated with at least one `dep` tag. The solutions we should produce are elements of this power set, so the positive ILASP examples will be trivially satisfied. Hence, we produce a negative example with the `goal` predicate in the exclusion for each text sentence example. The `goal` predicate, defined only in the

context of negative examples, is true if `in_generalised_sent` atoms correspond precisely to a positive example.

Example 5. Generating ILASP-compatible generalisation examples for *"He threw a fast ball.", "He threw a ball. He threw a fast ball."*

1. Convert the premise sentence to logical form (analogous to 2), resulting in tokens:

```
token(tok0, "he"). token(tok1, "threw").
token(tok2, "a"). token(tok3, "fast").
token(tok4, "ball"). root(tok1).
dep(nsubj, tok1, tok0). dep(dobj, tok1, tok4).
dep(det, tok4, tok2). dep(amod, tok4, tok3).
```

as shown in the example 2.

2. For each possible generalisation, create a positive example. The context consists of predicates produced in step 1, while the inclusion/exclusion of appropriate `in_generalised_sent` tokens (analogous to 3). The concept sentence *He threw a ball.* yields the following example:

```
#pos(example_id@noise_penalty,
{in_generalised_sent(tok0), in_generalised_sent(tok1),
 in_generalised_sent(tok2), in_generalised_sent(tok4),
 in_generalised_sent(tok5)},
{in_generalised_sent(tok3)}, % tok3 = "fast"
{
% all the predicates generated in step 1
}).
```

We would also construct a similar example for *He threw a fast ball.*

3. The negative example is generated as follows:

```
#neg(example_id@noise_penalty,
{ },
{ goal },
{
% all the predicates generated in 1)

% He threw a fast ball.
goal :- in_generalised_sent(tok0), in_generalised_sent(tok1),
        in_generalised_sent(tok2), in_generalised_sent(tok3),
        in_generalised_sent(tok4), in_generalised_sent(tok5).
% He threw a ball.
goal :- in_generalised_sent(tok0), in_generalised_sent(tok1),
        in_generalised_sent(tok2), in_generalised_sent(tok4),
        in_generalised_sent(tok5)}, not in_generalised_sent(tok3).
}).
```

All the examples constructed have a `noise_penalty=1` as the concept generalisation examples may be noisy.

3.3.2 Final Solution

ILASP is not nearly scalable enough to solve the task presented in this section. Still, with the optimisations from the next section, it can return a high-performing solution (example in Appendix D). In addition, we ran all ILASP tasks with the `--restarts` flag, which restarts the ASP solver between iterations. It can help reduce the running time of a program.

The generalisation task performance is evaluated in 3.5.2.

3.4 Managing Learning Scalability Constraints

A considerable challenge with logic-based learning systems, such as ILASP, is a lack of scalability when dealing with large-scale AI problems compared to other forms of machine learning.
For the generalisation task, two challenges needed to be mitigated:

- Lack of scalability w.r.t size of the hypothesis space – This issue arises due to ILASP enumerating search space S_M in full before finding a solution.
- Extreme RAM consumption during task solving.

Reducing the hypothesis space size

Since ILASP is not particularly scalable w.r.t to the size of the hypothesis space, the only option left was to reduce it.

For example, consider the following simplified language bias definition:

```
#modeh(in_generalised_sent(var(token))).
#modeb(in_generalised_sent(var(token))).
#modeb(root(var(token))).
#modeb(dep(const(label), var(token), var(token))).

... all the constant definitions ...
```

This program failed to return a solution, even after six days of running on a machine with an Intel Core i7 10510U 1.80GHz / 4.90GHz processor and 16 GiB of RAM.

FastLAS [34] is a system designed to alleviate this particular constraint, but it can only deal with a restricted version of the learning task. Its inability to deal with recursion made it inapplicable for the current problem.

The scalability issue was tackled using meta-level definitions of the hypothesis [29] space, which allow much greater flexibility than simple `modeb`, `modeh` statements. They allow constraining how rules are generated using ASP syntax, removing rules that we do not want as a part of the learned solution. We always want to remove the rules whose body could never be satisfied.

Here are some examples of how they are utilised to restrict the generalisation search space:

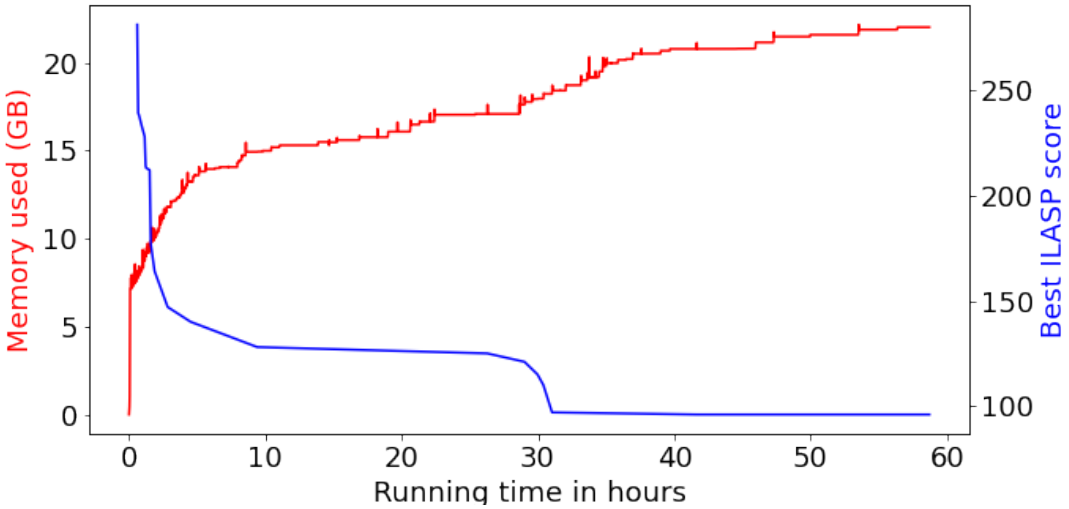
```
% Idea #1: Dep represents an arc in a tree. This allows
```

```
% cutting out rules which are impossible to be satisfied.

% It is impossible to have more than 1 root per example.
:- #count{T : body(root(T))} > 1.

% Trees cannot have a relation to itself.
:- body(dep(_, X, X)).
:- body(naf(dep(_, X, X))).

% A tree is not symmetric.
:- body(dep(_, X, Y)), body(dep(_, Y, X)).

% No dependency can go to the root
:- body(root(X)), body(dep(_, _, X)).

% Idea #2: Only allow two dep rules to occur in a body
% under certain conditions.

% Pairs of dependency tags which can co-occur.
% Much smaller set of all possible pairs.
dep_chain(prep, pobj).
dep_chain(pobj, amod).
...

% Allow any rule with at most one dep predicate
allowed_dep_rule :- #count{L, V4, V5 : body(dep(L, V4, V5))} <= 1.
% Allow rule with two dep predicates if its labels are white-listed
% by dep_chains and tokens are chained too.
allowed_dep_rule :- body(dep(L1, _, V2)), body(dep(L2, V2, _)),
                    dep_chain(L1, L2),
                    #count{L, V4, V5 : body(dep(L, V4, V5))} = 2.

:- not allowed_dep_rule.

% Idea 3: Remove rules that where simpler rules would suffice.

% Rule with root(V3) where V3 is not used in any dep is not needed.
% This predicate is trivially satisfied since every sentence has a
% root.
:- body(root(X)), not body(dep(_, X, _)), not body(dep(_, _, X)),
   body(dep(_, _, _)).

% If we have a in_generalised_sent predicate there must be some logic
% related to it.
% This predicate is trivially satisfied otherwise, since the
% background requires that at least one must always exist.
:- body(in_generalised_sent(X)), not body(dep(_, X, _)),
   not body(dep(_, _, X)).
```

These modifications allowed the search space generation times to take less than a minute, starting from over 13 hours. The final search space consisted of only 3652 rules.

Avoiding out-of-Memory Errors

Figure 3.7: Comparison of the ILASP solution performance and memory consumption over time. A lower ILASP score corresponds to a better solution.



ILASP memory consumption is drastic even for the simpler of the two tasks solved logically. The memory consumption graph over time is outlined in 3.7. As seen in the plot, the graph is around 15 GB of RAM at about the 15-hour mark. So, to prevent the usual 16 GB machines from going out of memory, the execution is terminated after 15 hours, and the best solution found until that point is returned. This technique prevents the ILASP process from exceeding the memory limit while returning a high-quality result.

The aforementioned ILASP behaviour modification is possible through the use of PyLASP scripts (Appendix C).

3.4.1 Atomisation Learning Challenges

Recall that the atomisation is only solved using the hand-crafted solution. In this section, we highlight why it is not currently possible to learn a good atomisation solution with ILASP.

We aimed to learn two things with ILASP for the atomisation learning task:

1. A set of rules which extend the number of tokens in the current atomic sentence.
2. A set of splitting tags.

Constructing the language bias by observing the hand-crafted solution results in the following code:

```
#modeh(in_atomic_sent(var(token))).
#modeh(splitting_tag(const(label))).

#modeb(1, dep(const(label), var(token), var(token)), (positive)).
#modeb(1, dep(var(label), var(token), var(token)), (positive)).
#modeb(1, splitting_tag(var(label))).
#modeb(1, in_atomic_sent(var(token)), (positive)).
#modeb(1, do_not_include(var(label))).
#modeb(1, candidate_start(var(token))).
#modeb(1, adjacent_subj).

% Add only negative atoms
#bias("
:- body(adjacent_subj).
:- body(candidate_start(_)).
:- body(do_not_include(_)).
").
```

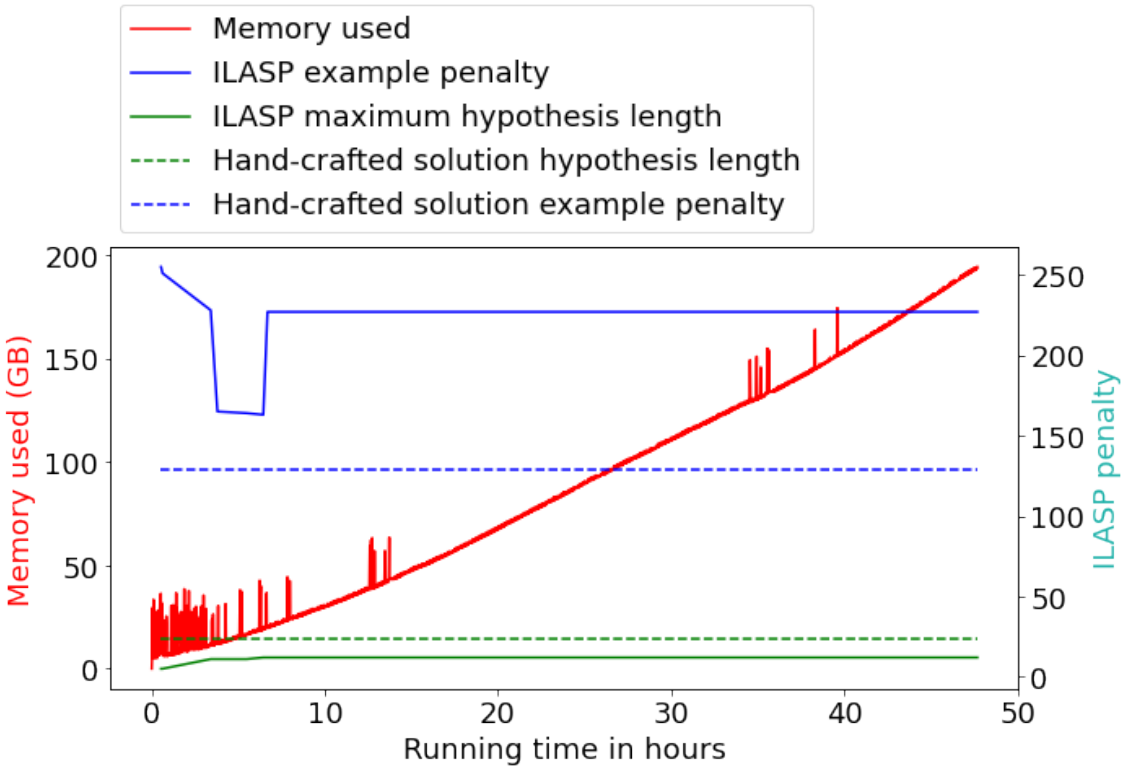
The language bias additionally included a more aggressive alternative to the search space reduction method presented in 3.4, but had all of the rules the hand-crafted solution uses.
In addition, we constructed the examples in the same manner as in 3.3.1. The learning was done on a specialised machine with 500 GiB of RAM, but it still required that an approximate solution be returned after some time (3.4).

ILASP can theoretically find the optimal solution for any noisy task, and the search space represents the hand-crafted solution in full. So, we expect the complete run of the task to produce at least as good of a solution as the hand-crafted one. However, as shown in 3.8, this expectation does not hold within the memory and time constraints available for the **atomisation** task.

Recall that in a noisy setting, the score ILASP minimises is the sum of the hypothesis length and uncovered example penalties. The hand-crafted solution would have had the score of 24 (hypothesis length) + 129 (uncovered example penalty) = 153. However, the graph 3.8 showcases that the learned solution never reaches those values. The maximum hypothesis length the ILASP allowed never went above 12. Had the learner raised the hypothesis length limit to at least 24, it could have found a solution with a smaller or equal noise penalty to the hand-crafted one.

The learner is unlikely to attempt as complex of a solution before the memory runs out. The memory consumption is quickly increasing while the maximum hypothesis score is staying flat in the graph 3.8. This behaviour may even suggest a presence of a bug.

Figure 3.8: Comparison of the ILASP atomisation solution performance and memory consumption over time. The dashed lines represent the values the hand-crafted solution would achieve had it been learned by ILASP.



3.5 Evaluation

3.5.1 Metrics

Both atomisation and generalisation problems had a set of solutions of unknown size. A needed metric would give a higher score if two sets are very similar compared to those far apart.

There were three metrics which we measured for that reason: Jaccard Index, Set-Recall, and Set-Precision:

Jaccard Index is a metric defined as follows:

- $Jaccard(A, B) = \frac{card(A \cap B)}{card(A \cup B)}$, where *card* represents set cardinatlity, A a set containing the true solutions, while B contains the predicated solutions.

It measures the overlap between the two sets.

$Set-Precision$ and $Set-Recall$ are defined in the similar manner as the *Precision* and *Recall* in the classification context:

- $Set - Precision(A, B) = \frac{card(A \cap B)}{card(B)} = \frac{number\ of\ correctly\ predicted\ sentences}{number\ of\ predicated\ sentences}$
- $Set - Recall(A, B) = \frac{card(A \cap B)}{card(A)} = \frac{number\ of\ correctly\ predicted\ sentences}{number\ of\ correct\ sentences}$

where *card* represents set cardinality, A a set containing true solutions, while B contains predicated solutions. These two metrics help better understand the type of errors the model makes.

3.5.2 Concept Generalisation

Cross-validation results

Due to only 130 examples available, we are using 10-fold cross validation to get the results for this chapter. In addition, to complete the execution with 16 GB of RAM, the computation is cut after 12 hours of running time (as argued in 3.4).

The results are summarised in the table below:

Summary of results			
	Jaccard Index	Precision	Recall
Training	0.92 ± 0.00	0.95 ± 0.00	0.95 ± 0.00
Test	0.85 ± 0.03	0.89 ± 0.02	0.89 ± 0.02

Overall, the results are relatively high, with >85% test Jaccard Index value. This result means that given a set of true sentences and a set of predicted sentences, more than 85% will match exactly. Looking into more detail the types of mistakes the final model makes, they can be subdivided into three groups:

- Genuine errors — The learned solution fails to produce the examples correctly.
- Borderline errors — The solution does not precisely match the provided gold standard. However, the produced solution might have been made by another data annotator. So, it is probably sufficiently good.
- Parser errors — The dependency parser is imperfect, which can be seen as the cause of some incorrect solutions. For example, a more accurate transformer parser labels *foul ball* as a compound of nouns, whereas the CPU-optimised one used in the project believes *foul ball* has an adjective. The reported LAS accuracy of the CPU-optimised parser is 0.9.

Finally, in all experiments, *Set-Precision* and *Set-Recall* were similar, meaning that missing a solution and producing an incorrect one were similarly likely.

Comparison with the Manually-Engineered Solution

The hypothesis was learned and engineered on a subset of the MLB-V2E dataset [4]. We compare their performance on the MLB-V2E dataset and a subset of the CUB-bird dataset [66], an out-of-domain dataset. The former dataset consists of 120 generalisation examples, while the latter has 100 samples.

The results are summarised in tables below:

In Domain Dataset			
	Jaccard Index	Precision	Recall
Hand-crafted H	0.79 ± 0.03	0.86 ± 0.03	0.80 ± 0.03
Learned H	0.92 ± 0.02	0.95 ± 0.02	0.94 ± 0.02

Out Of Domain Dataset			
	Jaccard Index	Precision	Recall
Hand-crafted H	0.71 ± 0.04	0.72 ± 0.03	0.83 ± 0.04
Learned H	0.83 ± 0.03	0.84 ± 0.03	0.86 ± 0.03

As expected, the performance is higher with the in-domain dataset. The out-of-domain dataset evaluation leads to a 10% Jaccard Index reduction for both methods. The drop is a result of the distribution shift between the two datasets. In particular, one of the main reasons for the drop in performance is the lack of accounting for the `acomp` tag (e.g. tag between *is* and *red* in *A bird is red*). It barely occurs in the MLB-V2E dataset, so both solutions fail to account for it.

3.5.3 Atomisation

There is no cross-validation evaluation of the ILASP output due to a poorly learned solution and high computation requirements (3.4.1). However, the best-learned solution is still compared to the hand-crafted one to showcase how far it is from the manually generated one. To better showcase the poor performance of the learned solution, simple H, a hypothesis which always returns the sentence in full, is introduced.

Comparison of the Learned and Manually Generated Solutions

The results averages and standard errors are presented in the tables below:

In Domain Dataset			
	Jaccard Index	Set-Precision	Set-Recall
Simple H	0.18 ± 0.04	0.18 ± 0.04	0.18 ± 0.04
Hand-crafted H	0.55 ± 0.04	0.61 ± 0.04	0.63 ± 0.04
Learned H	0.38 ± 0.04	0.42 ± 0.04	0.43 ± 0.04

Out Of Domain Dataset			
	Jaccard Index	Set-Precision	Set-Recall
Simple H	0.06 ± 0.02	0.06 ± 0.02	0.06 ± 0.02
Hand-crafted H	0.51 ± 0.04	0.55 ± 0.04	0.57 ± 0.04
Learned H	0.09 ± 0.02	0.11 ± 0.03	0.11 ± 0.03

The hand-crafted solution vastly outperforms the learned one, with a 45% Jaccard Index improvement. Nevertheless, even the hand-crafted solution has room for improvement as the Jaccard Index value is at 0.55. This result means that given a set of true sentences and a set of predicted sentences, we expect that 55% will match exactly. The Jaccard Index value, however, is the worst-case estimate. In some cases, the output is well enough even though it is not exactly correct. For instance, consider the sentence:

```
The pitcher threw the ball which was high.
  → The pitcher threw the ball. The ball was high.
```

This sentence is spilt with the current model into `The pitcher threw the ball. Which was high.`, resulting in the Jaccard Index value of 0.333.

Nevertheless, the sentence `Which was high` is often sufficiently good for downstream tasks. For example, it would be grouped with a valid sentence `It was high` in our concept bottleneck pipeline (Chapter 4).
Also, the presented sentence contains a relative clause, which was a significant hurdle for the hand-crafted solution. It requires that the ends of the splitting tag be in the same atomic sentence, which clashes with the existing core atomisation idea (3.2.2).

The out-of-domain dataset results in a 0.04 Jaccard Index drop for the hand-crafted hypothesis. However, that dataset seems better suited for the `splitting_tag` logic used since the sentences presented were less grammatically complex. The sentences mainly contained conjunctions with sub-clauses rarely appearing. The poor result comes from splitting the sentences with multiple conjuncts, which were unaccounted for in the hand-crafted solution.

On the other hand, the learned solution performs terribly on the out-of-domain dataset. There is no conclusive evidence that it consistently outperforms the simple hypothesis, as there is an overlap between the respective *Jaccard-Index* values.

Finally, in all experiments, *Set-Precision* and *Set-Recall* values were similar, meaning that missing a solution and producing an incorrect one was similarly likely.

3.6 Discussion

This chapter demonstrated that finding a high-quality solution for sequence to sequence NLP tasks is possible using logic-based learning techniques even when few examples are present.

To achieve a further increase in performance, we could get more examples so that a transformer model could tackle this problem well. This option, however, will require a lot of annotation effort.

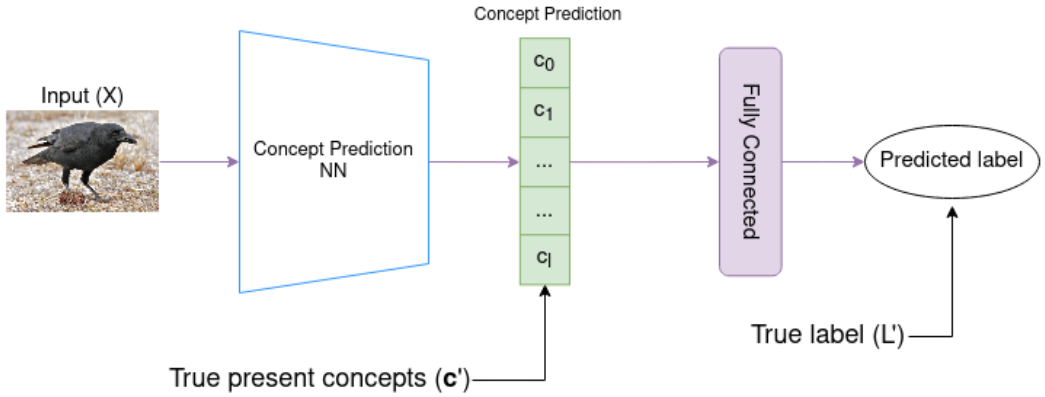
Chapter 4

Concept Bottleneck Model

Despite their tremendous success in recent years, end-to-end neural networks suffer from their black-box nature. They enable approximation of unknown functions exceptionally well but provide no interpretation of the actual function being approximated. The reasoning behind decision-making is almost a necessity in high-risk environments, such as the medical domain, where an error is extremely costly.

Concept Bottleneck Models [27] are a recent class of neural networks that tackle the lack of NN interpretability while achieving a competitive performance compared to the end-to-end counterparts. The architecture of such a model is shown in 4.1.

Figure 4.1: Schematic representation of the original concept bottleneck pipeline. The input, in this case, is an image but other types of input are also possible.



An end-to-end NN is transformed into a concept bottleneck one by forcing it to predict human-defined concepts before the final labels. The final result of the model can then be interpreted by verifying the concepts used when making a decision. For example, consider a bird prediction problem where human-defined concepts consist of the colour of the beak, wings and body. If a bird is a crow, the model would predict that it has a black body, wings and beak before indicating that it is a crow. From those results, an explanation *The bird is a crow because it has black wings, black beak and black body* could be generated.

In this chapter, we take the concept bottleneck idea further. We mine human-

understandable concepts from text explanations of a label in place of a predefined set of concepts. Such an approach would significantly simplify applying concept bottleneck pipelines in any domain. It removes the need to predetermine an appropriate set of concepts and manually label all data points where they occur. Text explanations, on the other hand, are sometimes readily available. For example, we could immediately use a patient's medical history report. If the explanations are not readily available, they are easy to produce.

4.1 Inherited Work

The work on the concept bottleneck pipeline has not been done from scratch.

This chapter itself is a continuation of the ideas proposed by Jeyakumar et al. in *Automatic Concept Extraction for Concept Bottleneck-based Video Classification* [4]. This paper failed to meet the acceptance threshold for the ICLR 2022 conference.

The main contribution by Jeyakumar et al. is the *Concept Discovery and Extraction Module* (CoDEx), a module that automatically uses explanations to extract text concepts. That module has been applied to the MLB-V2E dataset, a baseball video dataset with explanations.
The architecture involving the *Concept Discovery and Extraction Module* is shown in the figure 4.2. Generally, it is very similar to the standard architecture of the concept bottleneck architecture shown in the figure 4.1, only with the CoDEx module replacing the true present concepts.

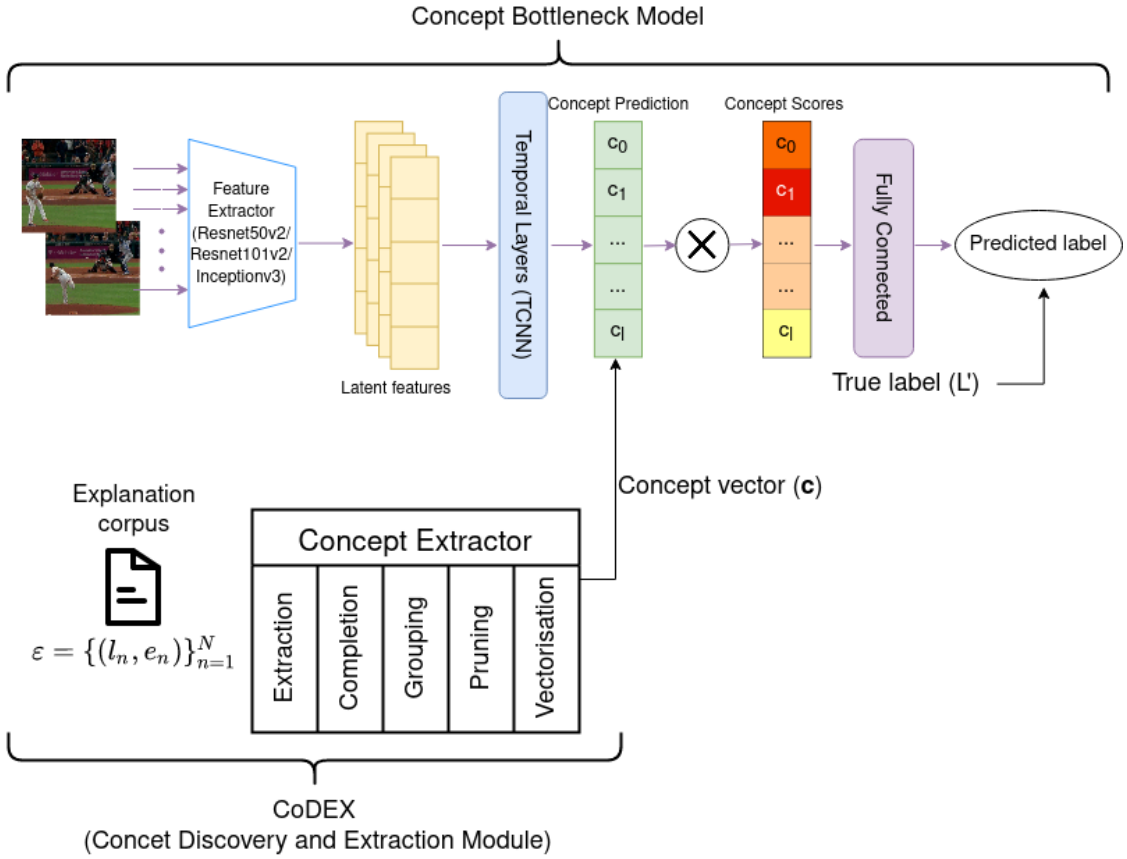
Jeyakumar et al. replace the set of human-crafted present concepts with the output of a CoDEx. In addition, to better understand each concept's impact on the final prediction, they add an attention layer after the concept layer to determine the concepts have more impact on the final prediction. They extract the top three concepts with the highest attention score from that layer, which they use as explanations for a particular label.

Concept Discovery and Extraction Module design is a vital element of the paper. The CoDEx presented in the paper consists of 6 stages: **cleaning**, **extraction**, **grouping**, **completion**, **pruning**, and **vectorisation**.
The **cleaning** stage removes videos/explanations which are corrupted.
Using a constituency parser and a rule-based methodology, the **extraction** step determines if a portion of the phrase should be included as a candidate concept. The constituency parser uses the rules shown in the following table to extract concepts:

Rules determining whether a candidate concept should be included or excluded	
rule name	rule
Inclusion 1	noun/pronoun → auxillary (optional) → particle (optional) → verb (optional)
Inclusion 2	noun/pronoun → auxiliary whose lemma is 'be' → any token
Exclusion	subordinating conjunction

The **completion** stage looks up for concepts identified in some explanations while not other ones due to the behaviour of the constituency parser. That stage is done

Figure 4.2: Schematic representation of the concept bottleneck pipeline presented by Jeyakumar et al. (adapted from [4])



by the substring lookup of existing concepts in all sentences.
The **grouping** step attempts to group concepts with similar meanings using agglomerative clustering [50].
The **pruning** stage attempts to keep highly informative concepts by picking the smallest concept subset such that the mutual information [42] between the label and a concept vector does not fall below a certain percentage γ. This problem is not solved precisely, but rather concepts are inserted using a greedy approach until the mutual information between the label and the newly constructed vector is not below a percentage γ. The authors chose $\gamma = 0.9$ as an appropriate value.
The vectorisation constructs the N x K concept matrix, where K is the number of concepts and N is the number of data points. The cell (n, k) is set to one if the concept k occurs for the data point n, zero otherwise.

After all these stages, a binary vector is produced for each data point, which is used in the NN training procedure. Jeyakumar et al. train the neural network using the joint training approach, where they optimise for both the concept loss and final loss at the same time. The concept loss is a binary cross entropy loss between a predicted concept vector and the CoDEx-produced true vector. In contrast, the final loss is a standard categorical cross entropy loss used for classification.

Using the CoDEx and the joint training procedure, the authors showed that their concept bottleneck pipeline had comparable performance to a standard end-to-end

model for video classification problems. In addition, the explanations they extracted were overwhelmingly better than the most common explanations.

4.2 Adaptions to the Concept Bottleneck Pipeline

In this chapter, we adapt the CoDEx part of the concept bottleneck pipeline to improve the performance and explainability of the entire model.

The **extraction** and the **completion** phase of the CoDEx have been removed from the pipeline. The former used a rule-based approach to extract concepts. The latter finds all concepts discovered by the constituency parser in one sentence but not in another using substring matching.

The removed stages are pretty and failed to account for a lot of information the video explanations conveyed.

To replace them extraction and completion we included **atomisation**, **generalisation** and **simple pruning** steps.
The chapter 3 explains the first two stages (tasks) in great detail. The **atomisation** stage splits provided sentences into one or more atomic sentences. Recall that the atomic sentences are sentences which an NLP expert cannot decompose into multiple valid sentences. This procedure is done using hand-crafted interpretable rules, which are presented in the section 3.2 along with the thought process for choosing the final solution.

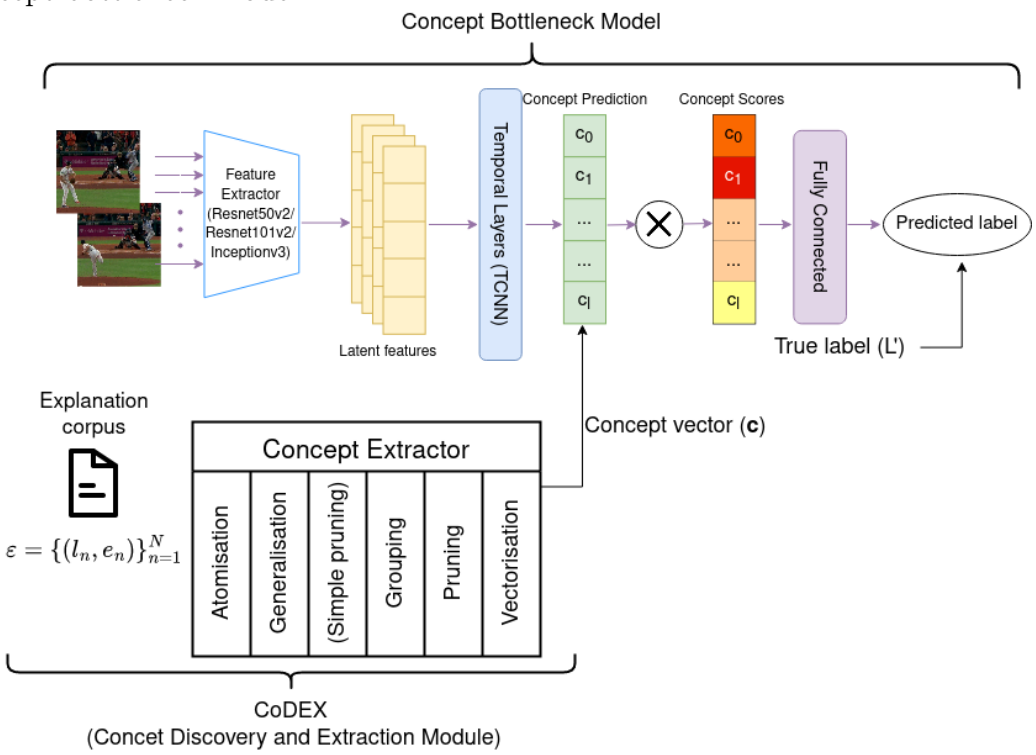
The **generalisation** stage should extract all concept sentences from an atomic one. As explained in the section 3.1, the concept sentence is a syntactically correct sentence obtained by modifying a sentence's syntax tree. To extract the concept sentences from an atomic sentence, we use a solution learned by ILASP, with the learning procedure described in 3.3.

The **simple pruning** stage, included between **generalisation** and **grouping** stage, is the simplest out of any of the steps present. It removes the concept if it fails to occur at least three times in the dataset. The sole reason for including this stage is to speed up the subsequent steps because it slightly reduces the overall performance of the pipeline. CoDEx pipeline for the bird-flowers dataset (4.3.4) takes less than an hour when the **simple pruning** stage is not included, which increases to over 10 hours without it. This stage is only included when the CoDEx pipeline takes too long to compute otherwise because it is lossy.

The diagram summarising the new concept bottleneck pipeline is shown in the figure 4.3.

There were recent concerns about the joint training procedure not truly accounting for concepts in their final predictions [44]. So, we also attempt to use the sequential training procedure, isolating the training of concept and label prediction parts of the network. Such a training procedure performs similarly in [27], so we expect the same.

Figure 4.3: The entire pipeline for the classification of the MLB-V2E dataset using the concept bottleneck model.



4.3 Evaluation

This section will discuss how well the newly presented CoDEx pipeline performs. Any results produced will be compared with the previous iteration of the concept bottleneck pipeline and the uninterpretable end-to-end model, if applicable. To keep concept bottleneck pipelines easily comparable, we prune the concepts such that only 78 of them remain. This number was a desirable value for the original concept bottleneck pipeline [4].

Each concept bottleneck model is also trained using both joint and sequential models, unlike the 4.1 which only uses the joint model. The difference between the two is that the joint optimises losses for both concepts and labels simultaneously, which sometimes overrules the nature of the concept bottleneck pipeline. The sequential training model freezes the learned concept prediction layers, so the model must determine the final label from the concept predictions themselves [44].

For completeness, the architectures used to train the models are presented in the Appendix B.

Also, we evaluate the concept bottleneck model on a new birds-flowers dataset, which combines randomly selected images from the CUB-200-2011 [66] and 102 Category Flower Dataset [51] along with their descriptions. This evaluation inspects how well the concept bottleneck pipeline translates to different domains.

In addition, the simple pruning stage is used for the birds-flowers experiment (as it takes too long to generate CoDEx matrix otherwise) while not for the MLB-V2E

dataset.

Finally, we will critically analyse the strengths and limitations of the implemented method and suggest areas for future improvement.

4.3.1 CoDEx based evaluations

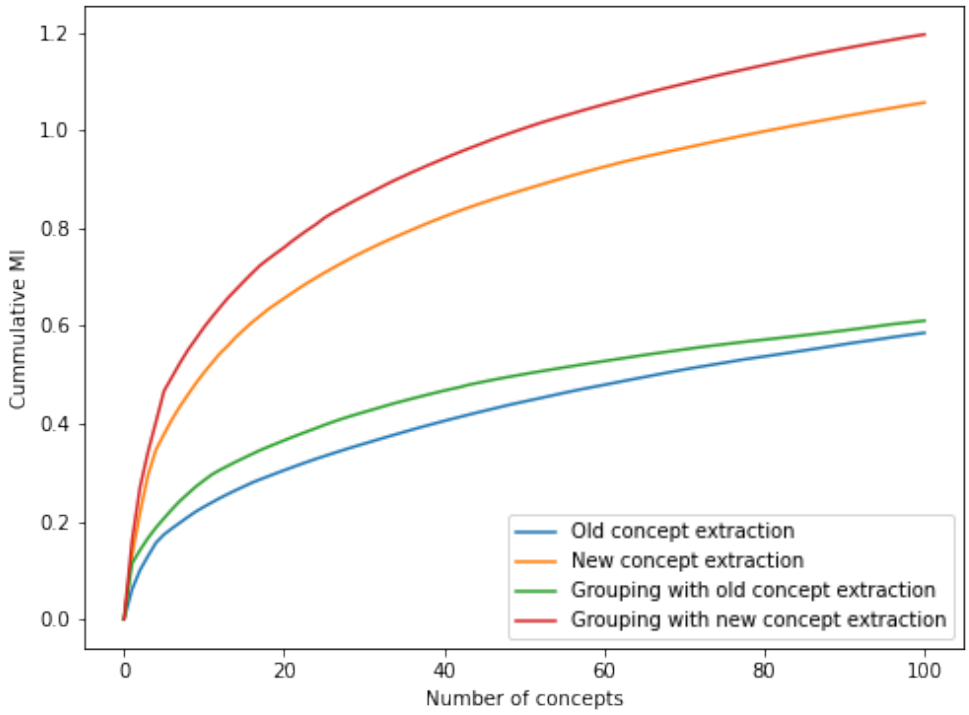
By manually observing the concepts produced (Appendix E), both methods extract relevant concepts for the problem, although not all are perfect. For example, the new method extracts a concept *It was* which is not a relevant concept. In general, the new method seems to extract concepts which better describe the labels, but we evaluate this belief concretely in this subsection.

Measuring Cumulative MI

Mutual Information $I(X;Y)$ [42] is defined as $I(X;Y) \equiv H(X) - H(X|Y)$, where H represents entropy measure. It estimates the average reduction in uncertainty about x caused by understanding y's value or vice versa. It is the average quantity of information conveyed by x regarding y.

In the context of this project, the discrete variable Y is a set of label outcomes. A possible outcome is a number between 0 and 4, where each number represents an event which occurred. The discrete variable X represents a set of extracted concepts with size k, where k is a parameter tested in this evaluation.

Figure 4.4: Cumulative MI graphs using old and new concept extraction pipeline



In addition, given that the cumulative MI is under consideration, the set of size k should ideally contain k *maximally informative* concepts, i.e. those which will result in the highest mutual information score. However, finding such a set is infeasible as the problem is combinatorial [4]. Still, we can get a highly informative set by greedily adding a single concept that improves MI the most.

So, by measuring cumulative MI, we can determine how good the extracted concepts are at describing the labels.

The results can be seen in the figure 4.4.

It shows that the new concept extraction method drastically outperforms the old approach for the first 100 concepts. Since only 78 are taken for the entire pipeline training, these results indicate that the new concept bottleneck performance should drastically improve.

Concept Prediction Performance

A corollary of the higher concept mutual information is a higher concept prediction accuracy.

Consider a classifier which would result in the highest possible accuracy for a dataset from binary vectors to labels. Such a classifier assigns the correct label to any non-conflicting binary vector and the most common label to any conflicting binary vector. A conflicting vector is one whose outcome can result in multiple different labels. The maximum possible accuracies are shown in the table below:

Maximum accuracy comparison		
	Old concept extraction	New concept extraction
Training dataset	51.1%	78.6%
Test dataset	47.7%	87.7%

The highest possible accuracy with the new concept extraction for training and the test set is much greater than the values achievable with the old procedure.

To validate these results translate into practice, we train an MLP classification model from binary vectors to labels with both old and new concept extraction. In addition, a RoBERTa transformer model [39] is trained directly from an unedited human-generated explanation to estimate how much information is lost through concept extraction. The results are summarised in the table below:

Practical accuracy results		
Transformer	Old Concepts	New concepts
0.953 ± 0.002	0.450 ± 0.001	0.820 ± 0.002

These experiments convince us that the performance of a concept bottleneck performance should improve. They also suggest that there is still room for improvement for the concept extraction since there is a 19% accuracy improvement by training a model directly from text.

4.3.2 Performance of the Full Concept Bottleneck Pipeline

The model performance is measured on the entire concept bottleneck pipeline. We measure the final accuracy of the models with identical architectures (Appendix B) using new and old concept extraction. In addition, the quality of the concept prediction is also measured as it can help understand whether the concept bottleneck nature of the model is indeed considered.
The precision metric is used to measure the concept prediction performance because detecting a subset of present concepts is often enough to determine the final label correctly. Furthermore, most concept values are set to 0 for any data point, so metrics considering true negatives would have a high value even when the actual concept prediction is poor. The final label prediction is measured using accuracy, as we want the predictions to be correct most of the time. The predictive accuracies are also compared with an end-to-end model with an identical architecture, the highest value the concept bottleneck models could achieve.

The model accuracies are presented in the below:

Final label prediction accuracy		
No concepts	Old Concepts	New concepts
0.687 ± 0.005	0.623 ± 0.005	0.685 ± 0.005

The model without concepts has a higher mean accuracy than both concept bottleneck models. This outcome is expected as a concept bottleneck model should force the network to conform with human reasoning. In addition, we expect that the new concept extraction outperforms the old one due to the higher mutual information value its concepts have.
However, we expected a much more significant performance difference between the concept bottleneck approaches due to a stark contrast in mutual information values and concept-only predictive accuracies. To understand why that is the case, we need to consider the precision values of the concepts:

Old concepts precision	New concept precision
0.163 ± 0.003	0.172 ± 0.005

The concept predictions are extremely poor compared to the final results. So, it seems that concepts serve more as proxies for holding NN final prediction information instead of genuinely being learned by the network. This statement is backed up by the fact that old concept prediction even outperforms the prediction results directly from concepts, which would not have been possible in an actual concept bottleneck model [44]. Nevertheless, such low prediction values may not result in poor explanations since concepts could exist in the video but not in the CoDEx matrix.

To force the concept bottleneck nature of the model, we train another model using the sequential approach. Such an approach first trains the concept prediction part of the network before training the network from concept predictions to final labels, with the results shown in the tables below.

Sequential training full model results			
	No concepts	Old concepts	New concepts
Final accuracy	0.687 ± 0.005	0.354 ± 0.019	0.579 ± 0.008
Concept precision	/	0.610 ± 0.018	0.479 ± 0.016

The sequential training has drastically improved concept precision values as the predictions from the final layer could not have overridden the concept predictions. In addition, the sequential training has reduced the performance of both models. It has made the difference between the results much bigger, clearly showcasing that new concept predictions are significantly better than the old ones.

However, the decrease in performance is undesirable. If the concepts present in the video coincide with the concepts predicted by the network, joint training should be preferred as it has a higher performance. We further quantitatively evaluate the concepts predicted by the joint network to see if they would constitute good explanations.

4.3.3 Explainability of the Labels

The main reason we use the concept bottleneck pipeline is to allow interpretation of final labels using the intermediate concepts. To do so, we compare the concept values after attention as human subjects prefer them over all alternatives in almost 70% of the cases in [4].

We consider a concept as active for a data point if it has a score close to the maximum. This behaviour was chosen because there was a sharp concept value drop of more than 50% from the few top values, making these concepts a lot less relevant for the final decision.

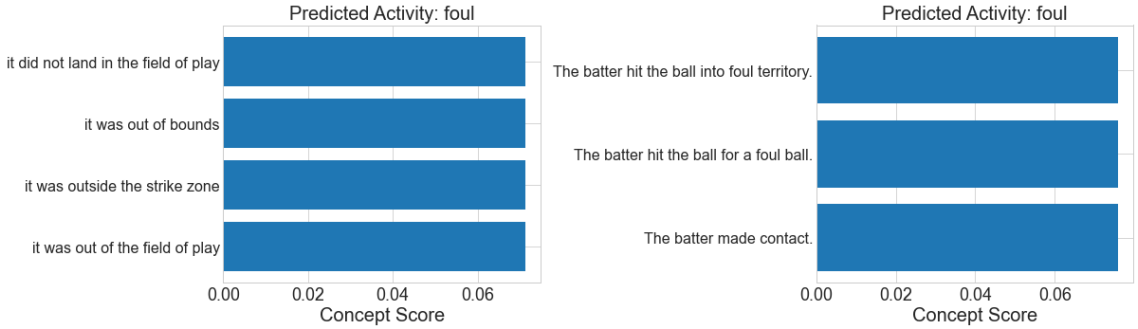
Unfortunately, due to a lack of resources, we have not been able to repeat the Mechanical Turk study done in [4]. Instead, we only discuss three randomly selected test examples and qualitatively compare the old and new concept predictions for them:

Sourced explanation 1: *The batter swung the bat and hit it into foul territory. This was called a foul ball by the ump.*
True label 1: foul.

The results are shown in 4.5. The video consisted of a batter hitting the ball in the air behind him in the foul territory. Both networks accurately predict this sequence as a foul.

For the new concept bottleneck network, all predicted concepts are correct. The batter has indeed made contact; the ball went into the foul territory and was undoubtedly a foul ball. They describe well all the events that happened. On the other hand, old concepts also perform well in this case. One should be able to determine that the final label should be foul from the concepts. The concept *it was outside of strike zone* is wrong. If that piece of information is indeed correct, the true label would have been *ball* instead of the *foul*. In addition, *it was out of bounds*, *it did not land in the field of play*, *it was out of field of play* all capture that the ball

Figure 4.5: Most relevant concepts for example 1, along with their concept scores. The LHS shows the ones predicted by the old concept bottleneck, while the RHS is predicted by the new one.

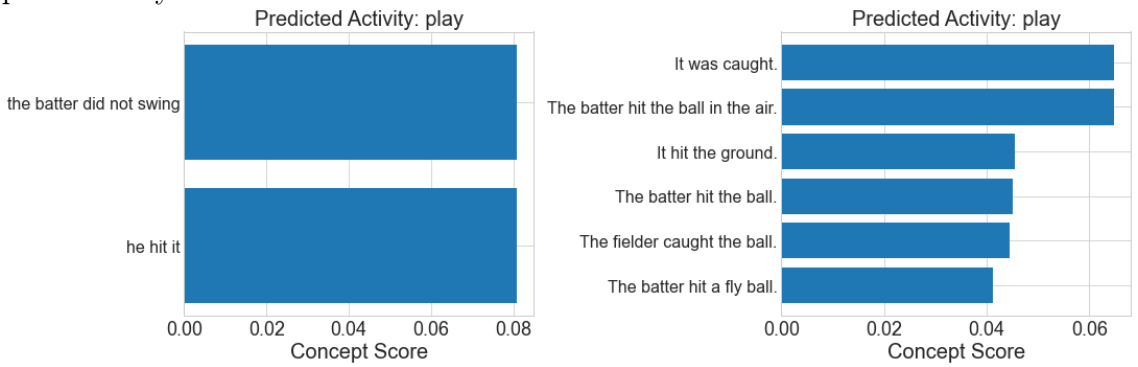


landed outside of the allowed territory. It may be sufficient only to present one such concept.

Although both concepts are good enough to predict the final label accurately, a human subject would usually prefer the new one. It is not only because the old concept bottleneck presents an incorrect sentence but also because the new concepts seem much clearer. The user does not need to infer what "it" is referring to.

Sourced explanation 2: *The batter hit it in play and it was not caught.*
True label 2: play

Figure 4.6: Most relevant concepts for example 2, along with their concept scores. The LHS shows the ones predicted by the old concept bottleneck, while the RHS is predicted by the new one.



The results are shown in 4.6. The video in explanation 2 showed a batter hitting a fly ball which an outfielder caught after it hit the ground. Both networks accurately predict this sequence as a play.

Unlike the previous video, a baseball expert would not determine the final label from both explanations. The old concept prediction only predicted two conflicting concepts, out of which only *he hit it* is correct.

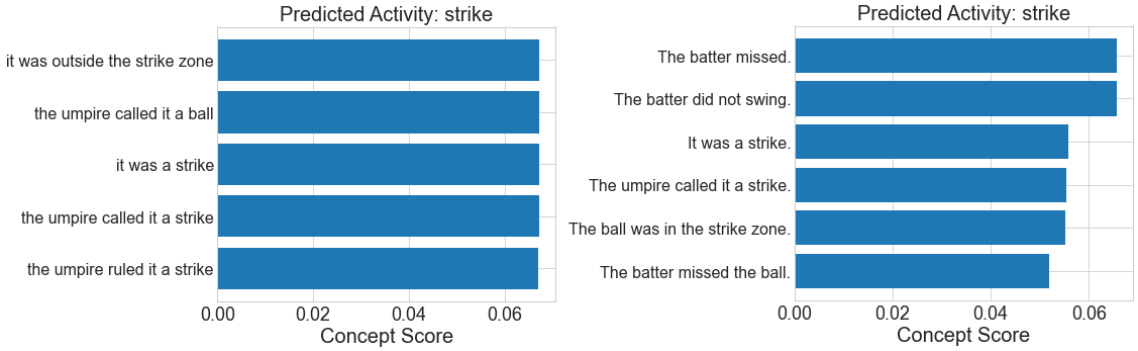
On the contrary, the new concept extraction predicted all relevant events that have

occurred in the video. These concepts have some redundancy, such as *It was caught* and *The fielder caught the ball* presenting the same thing. Moreover, the new concepts showcase a finer level of granularity in the event prediction by predicting that a fly ball has occurred. The concept scores might slightly confuse a user, in this case, as they highlight two concepts more than the others. If these were showcased on their own, they would seem more related to *out* rather *play*.

Sourced explanation 3: *the batter did not swing at a ball in the strike zone.*
True label 3: strike

The results are shown in 4.7. Both networks correctly predict this sequence as a strike.

Figure 4.7: Most relevant concepts for example 3, along with their concept scores. The LHS shows the ones predicted by the old concept bottleneck, while the RHS is predicted by the new one.



Again, the new concept extraction pipeline seemed to have extracted better concepts for describing the label. There are two reasons for it: they are more accurate and describe multiple events from which the final label could be inferred.

The old concept prediction pipeline predicted two incorrect concepts: *it was outside of the strike zone* and *the umpire called it a ball*. On the contrary, All of the newly inferred concepts are correct.

The new concept prediction describes multiple relevant events. For example, it describes both the umpire ruling the event as a strike and the ball being in the strike zone. On the other hand, only the umpire calling the pitch a strike was predicted. The other two concepts either describe the same thing *the umpire ruled it a strike* or have predicted the final label instead of a concept *it was a strike*. Note that the latter was also predicted in the new concept bottleneck pipeline. That sentence should be omitted since it would not be a meaningful statement in any explanation describing why the label occurred.

To sum up, there seems to promise that the new method improves explanations in both accuracy and the level of detail. We should create a human study in the future to validate these claims.

In addition, notice that the model should not predict a few of the concepts chosen

if CoDEx output was used as the ground truth. However, they were mostly correct, especially for the new concept extraction. So, the poor precision value may not necessarily hurt explanations, but rather correct concepts may not often be captured by the CoDEx module. These results may even suggest that joint network training helps extract a good set of concepts by using the final label information.

4.3.4 Applicability in other Domains

As argued, the idea presented in this section, in theory, is not tied to video applications. This evaluation measures how well the CoDEx module fares when applied to a birds-flowers dataset. This dataset consists of 2000 birds/flowers images taken from CUB-birds [67] and 102 flowers [51] dataset, combined with image descriptions generated for these datasets. As such, the explanations are written to describe the image rather than its relevance to the label. It was chosen due to its simplicity for a standard NN, as demonstrated by the high accuracy of a non-concept bottleneck model. In addition, the explanations were also readily available to use for concept prediction.

By observing the concepts extracted, we see that the set of concepts may be applicable. For example, the CoDEx extracts the following concepts together with their occurrence in the dataset:

```
The bird has feathers. % 18
This flower has pistil. % 18
```

Despite the sentence structure immediately highlighting whether it is a bird or a flower, the model would nevertheless try to learn what are feathers and pistils. The issue is the count occurrence for doing so. More than 18 birds in the dataset have feathers, and more than 18 flowers have pistils. This poor count results from label explanations extracted from the two label descriptions focusing on differences between different species of birds/flowers instead of the differences between the two. In addition, not all concepts are helpful, such as *This bird has*, but in general, there seems to be enough information to determine the final labels correctly.

On the other hand, the concepts produced by the old approach seem much worse. It is often quite unclear which label they correspond to, such as *that are oval shaped*. Or, there are a few very specific concepts grouped with other similarly long concepts. For example, a concept *this is a white and black bird with a black crown and a long black beak* would fall into that category.

The dominant (most frequent) extracted concepts are included in Appendix E, along with the sum of all concept frequencies in that group.

To validate whether the initial observations translate to tangible results, we evaluate how well can an MLP predict the final label. The input are the concept prediction vectors produced by old and new CoDEx modules. The results are compared against a fine-tuned RoBERTa transformer [39] trained directly from text to estimate how much information is lost by mining concepts.

Results from explanations only			
	Transformer	Old concepts	New concepts
Final accuracy	1.000 ± 0.000	0.844 ± 0.008	0.874 ± 0.006

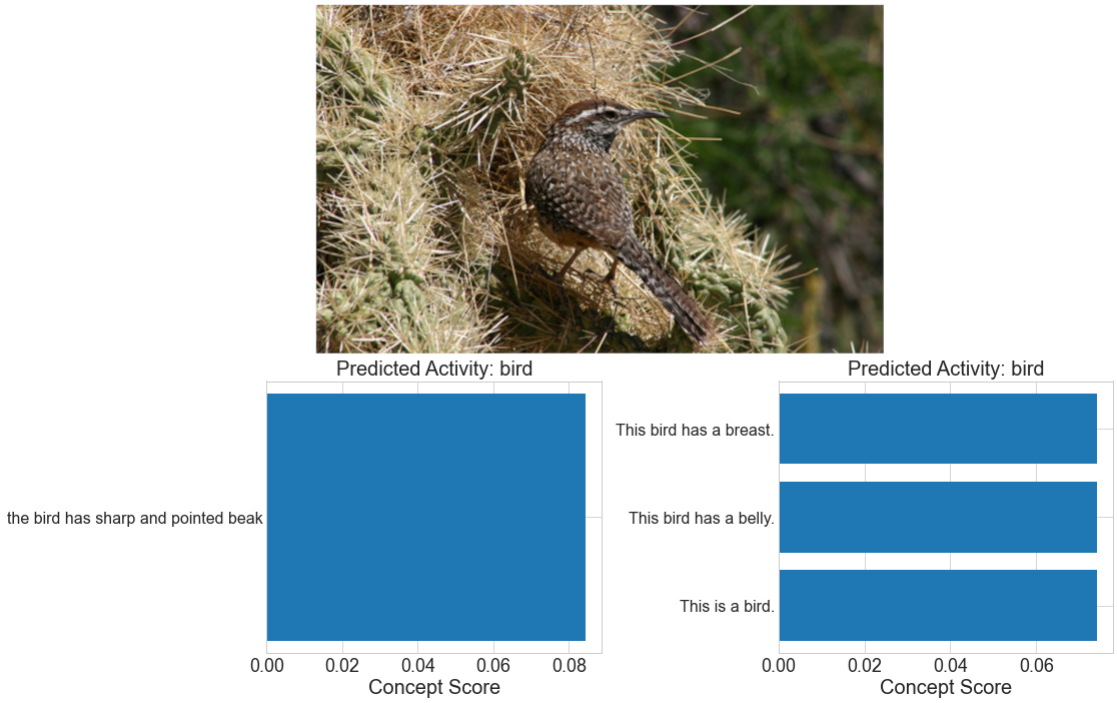
The newly extracted concepts can better distinguish between labels, although the difference is not as big as for the baseball dataset. The main reason for a loss in accuracy is the case where all inputs are 0, where any concept that was detected ended up being pruned. This case happens more often for the old concept extraction, but the new one is not immune. As the transformer has perfect accuracy, it is clear that it is straightforward to distinguish between the two labels from descriptions.

We further evaluate the jointly trained models with identical architecture (Appendix B) and hyper-parameters using old, new or no concepts in the intermediate layers:

Full model results			
	No concepts	Old concepts	New concepts
Final accuracy	0.979 ± 0.004	0.987 ± 0.002	0.989 ± 0.002
Concept precision	/	0.367 ± 0.021	0.212 ± 0.009

The final accuracies of the models all have similarly high values, which is desirable. However, the concept precision results are similarly poor as the results for the MLB-V2E [4] dataset. As mentioned, the text explanations do not describe all features that a bird/flower has. For that reason, the CoDEx pipelines often fail to account for many concepts present in the image but not the explanations during training. In addition, the reason behind higher concept accuracy may be related to the NN optimising for a positive concept value much more often with new concept extraction. The training set for the new concept bottleneck pipeline has more than twice the number of detected concepts.

Figure 4.8: The most relevant concepts for the image are shown below, along with their concept scores. The LHS shows the ones predicted by the old concept bottleneck, while the RHS is predicted by the new one.



We further present the predicted concept values for randomly selected data points

shown in the figure 4.8 and 4.9.

For the bird example in 4.8, both outcomes could be better. The old concept bottleneck pipeline only shows a true but very specific feature. On the other hand, the new concept bottleneck pipeline predicted better concepts that would describe some birds well. However, given the bird's orientation in the image, specifying that it has a breast and belly seems counter-intuitive.

Figure 4.9: The most relevant concepts for the image are shown below, along with their concept scores. The LHS shows the ones predicted by the old concept bottleneck, while the RHS is predicted by the new one.

The results for the example in 4.9 could be better. The new concept bottleneck extracts well that *this flower has petals*. However, the other two concepts are not even valid sentences. The old concept bottleneck similarly extracts concept texts missing crucial information.

To sum up, neither method extracts good concepts for the bird-flowers dataset. The conjunction of bird/flower image descriptions did not prove sufficient information to extract good concepts for the final explanation. We believe that for the method to work, the explanations need to be generated to highlight why this data point should be assigned precisely to some class and not the other.

4.3.5 Discussion

In this chapter, we show that the addition of atomisation and generalisation greatly helps the concept bottleneck pipeline performance. When incorporated into the concept bottleneck pipeline, the new CoDEx has a higher mutual information, predictive accuracy, and performance. In addition, there is an indication that its explanation generation is also better.

We also highlight a possible issue with joint training of a concept bottleneck model that may not truly consider the concept bottleneck nature of the problem. However, to achieve comparable performance with the sequential training, we need to find a way to capture all the concepts that occur in the video.

Further improvements of the concept bottleneck CoDEx should incorporate a mechanism for detecting concepts detected in the video/image, but not the explanation. With such a mechanism, training the concept prediction part of the network would yield a much better performance, which should improve the accuracy of generated explanations.

Although the birds/flowers experiment failed to extract good explanations for images, it demonstrated that the method presented in this chapter applies to data of different modalities, not just videos. The experiment required no change of the CoDEx pipeline to mine concepts from image explanations.

Chapter 5

Logic-Based Classification

The concept bottleneck pipeline [27] uses human-explainable high-level concepts in the intermediate layer to find reasons for choosing a particular label. These concepts should help a human understand why a prediction happened. However, just because concepts are predicted does not mean they are indeed used in the final prediction.

For example, a network may learn a pattern which predicts a label *out*, if some concept has a value greater than 0.2. Yet, the model would not show such a concept to the user as relevant for the final prediction. As we have done in the Chapter 4, showcasing the concepts after the attention layer would present the concepts more relevant for the final prediction. Still, the attention layer presents the weight each concept has in the final prediction, but not the logic the MLP uses to choose the final label.

A fully interpretable method for predicting the final label would provide a clear link between the concepts and the outcomes. In addition, models from concepts to the label can be validated and should, in an ideal case, follow the same logic as humans.

This chapter presents Prob-FF-NSL, an ILP-based classification method to improve the concept bottleneck model interpretability. The method works with probabilistic facts, allowing it to be incorporated into a concept bottleneck pipeline (Chapter 4). Nevertheless, it is not limited to text concepts; it applies to any interpretable probabilistic NN output. In section 5.4.1, we evaluate it with outcomes of an image classification task. The ILP system utilised for this task is FastLAS [34], a scalable system that incorporates criteria for domain-specific optimisation.

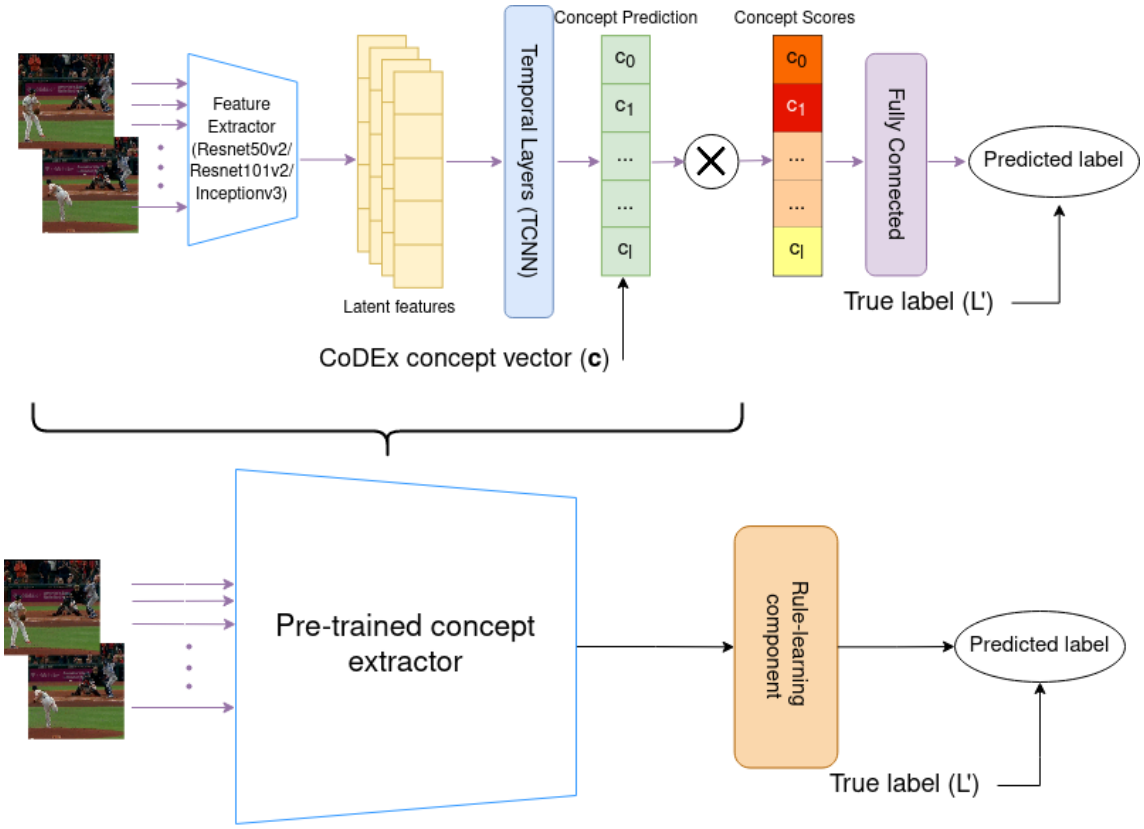
5.1 Changes to the Concept Bottleneck Architecture

An introduction of the rule learning component would result in the change in the concept bottleneck architecture shown in the figure 5.1.

Training a concept bottleneck pipeline then turns into a 3-step process:

1. Train a model in the same manner as in the Chapter 4, by minimising the loss CoDEx concept vector and final label prediction loss.

Figure 5.1: A flowchart to select an outcome of a multi-class classification problem.



2. Generate examples for the rule-learning component using the concepts predicted by the model from the step 1). Each example needs to take into account the probabilities of all concepts.

3. Train the rule-learning component.

At run-time, predict the concepts using the model trained in step 1 before applying the rules learned in step 3 to obtain the final prediction.

The ASP paradigm does not support any notion of probability; an atom can either be in or out of an answer set. So, to force the learner to treat less likely cases differently, we carefully choose the FastLAS example penalties, similar to Cunnington et al. [12]. However, we aim to use those penalties to find the most likely solution.

5.2 Choosing FastLAS Parameter Values

FastLAS can find an optimal solution with respect to a scoring function of kind $(\mathcal{S} + \mathcal{S}_{pen})$, where $\mathcal{S}$ is a decomposable scoring function (2.2) and $\mathcal{S}_{pen}$ is a sum of all uncovered example penalties.

In this section, we attempt to find the appropriate scoring function $\mathcal{S}$ and values for penalties such that the final solution is highly likely given the learning task tuple $T = \langle B, M, E_{prob} \rangle$.

5.2.1 Used Notation

Let our set of examples E_{prob} be $\{(\mathbf{x}_e, y_e)\}_{e=1}^{E} \in \mathcal{E}$ with E examples. An example $(\mathbf{x}_e, y_e)$ consist of a vector of concept probabilities $\mathbf{x}_e = (x_{ec})_{c=1}^{C}$ and a label $y_e \in \mathcal{L}$ assigned to the example e from the set of possible labels $\mathcal{L}$. The symbol x_{ec} denotes a priori probability that concept c takes truth value (=1) in example e. Moreover, C is the number of extracted concepts for a given problem.
We can represent examples as a matrix of concept probabilities. The examples can be represented as an input matrix of concept probabilities $\mathbf{X} = (\mathbf{x}_e^T)_{e=1}^{E}$ and an output vector of target labels $\mathbf{y} = (y_e)_{e=1}^{E}$.

Let $\mathcal{H}$ be the set of all possible hypotheses. The term $p(y_e|H, \mathbf{x}_e)$ represents the probability that the example e is covered, for any hypothesis $H \in \mathcal{H}$,

We introduce notation for grounded examples, as answer sets can only contain grounded atoms. A grounded example $(\mathbf{z}_e, y_e)$ is an example where each concept c in an example e is assigned a truth value $z_{ec} \in \{0, 1\}$ (integer form). The term $\mathbf{z}_e = (z_{ec})_{c=1}^{C} \in \mathcal{Z}_e$ is a binary vector representing the ground assignment.

5.2.2 Determining Optimal Example Penalties

We want to choose the example penalties such that the FastLAS output hypothesis H is the maximum-likelihood hypothesis H_{ML}, i.e. the output hypothesis should satisfy the following equation:

$$H_{\text{ML}} = \arg\max_{H} p(\mathbf{y}|H, \mathbf{X}) \tag{5.1}$$

Making an assumption that given a model and concept probabilities, the labels for any two examples are conditionally independent we can rewrite the term $p(\mathbf{y}|H, \mathbf{X})$ in the following manner:

$$p(\mathbf{y}|H, \mathbf{X}) = \prod_{e} p(y_e|H, \mathbf{x}_e) \tag{5.2}$$

As the $ln = log_e$ function is monotonically increasing in the range $(0, \infty)$ and as the above equation produces values in that range, it is equivalent to maximising the ln, resulting in the following optimisation target:

$$\begin{aligned} H_{\text{ML}} &= \arg\max_{H} \ln p(\mathbf{y}|H, \mathbf{X}) \\ &= \arg\max_{H} \sum_{e} \ln p(y_e|H, \mathbf{x}_e) && \text{(by 5.2)} \\ &= \arg\min_{H} -\sum_{e} \ln p(y_e|H, \mathbf{x}_e) && \text{(alternative objective)} \end{aligned} \tag{5.3}$$

We can now calculate the probability that a label y_e, for example e, is covered by a hypothesis H where we know the set of concept probabilities $\mathbf{x}_e$. That probability

is computed by considering all possible grounding that $\mathbf{x}_e$ induces. It is given by:

$$\begin{aligned} p(y_e|H,\mathbf{x}_e) &= \sum_{\mathbf{z}\in\mathcal{Z}_e} p(y_e,\mathbf{z}|H,\mathbf{x}_e) && \text{(sum rule)} \\ &= \sum_{\mathbf{z}\in\mathcal{Z}_e} p(y_e|\mathbf{z},H,\mathbf{x}_e)p(\mathbf{z}|H,\mathbf{x}_e) && \text{(conditional probability)} \\ &= \sum_{\mathbf{z}\in\mathcal{Z}_e} p(y_e|\mathbf{z},H)p(\mathbf{z}|\mathbf{x}_e) && \text{(by independence)} \\ &= \mathbb{E}_{\mathbf{z}\sim p(\mathbf{z}|\mathbf{x}_e)}[p(y_e|\mathbf{z},H)] && (5.4) \end{aligned}$$

Considering the ln of the term above, we can use the Jensen's inequality to push it inside the expectation:

$$\begin{aligned} \ln p(y_e|H,\mathbf{x}_e) &= \ln \mathbb{E}_{\mathbf{z}\sim p(\mathbf{z}|\mathbf{x}_e)}[p(y_e|\mathbf{z},H)] \\ &\leq \mathbb{E}_{\mathbf{z}\sim p(\mathbf{z}|\mathbf{x}_e)}[\ln p(y_e|\mathbf{z},H)] \end{aligned} \tag{5.5}$$

We further assume that the bounds of Jensen's inequality are sufficiently tight, i.e. we assume that:

$$\ln p(y_e|H,\mathbf{x}_e) \approx \mathbb{E}_{\mathbf{z}\sim p(\mathbf{z}|\mathbf{x}_e)}[\ln p(y_e|\mathbf{z},H)] \tag{5.6}$$

Now we can estimate the expectation by sampling I ground examples for example e, where ground examples are $\mathbf{z}_i \sim p(\mathbf{z}|\mathbf{x}_e)$, then:

$$\mathbb{E}_{\mathbf{z}\sim p(\mathbf{z}|\mathbf{x}_e)}[\ln p(y_e|\mathbf{z},H)] \approx \frac{1}{I}\sum_{i=1}^{I} \ln p(y_e|\mathbf{z}_i,H) \tag{5.7}$$

For some hypothesis, $H \in \mathcal{H}$, a grounded example is or is not covered by H. We allow a label to be incorrect with some small error $\epsilon > 0$, resulting in the following probabilistic interpretation:

$$p(y_e|\mathbf{z},H) = \begin{cases} 1-\epsilon & \text{if } H,\mathbf{z} \models y_e \\ \epsilon & \text{otherwise} \end{cases} \tag{5.8}$$

Combining all the results presented, we can derive the following:

$$
\begin{aligned}
H_{\text{ML}} &= \arg\min_H -\sum_e \ln\left(\mathbb{E}_{\mathbf{z}\sim p(\mathbf{z}|\mathbf{x}_e)}[p(y_e|\mathbf{z},H)]\right) && \text{(by 5.3 and 5.4)}\\
&\approx \arg\min_H -\sum_e \mathbb{E}_{\mathbf{z}\sim p(\mathbf{z}|\mathbf{x}_e)}[\ln\left[p(y_e|\mathbf{z},H)\right]] && \text{(by 5.6)}\\
&\approx \arg\min_H \sum_e \frac{1}{I}\sum_{i=1}^{I} -\ln\left[p(y_e|\mathbf{z},H)\right] && \text{(by 5.7)}\\
&= \arg\min_H \sum_e \frac{1}{I}\sum_{i=1}^{I} \left(-\ln[p(y_e|\mathbf{z}_{ei},H)] + \ln(1-\epsilon)\right) && \text{(constant shift)}\\
&= \arg\min_H \sum_e \frac{1}{I}\left(\sum_{H,\mathbf{x}_{ei}\models y_e} 0 + \sum_{H,\mathbf{x}_{ei}\not\models y_e} (-\ln\epsilon + \ln(1-\epsilon))\right) && \text{(by 5.8)}\\
&= \arg\min_H \sum_e \sum_{H,\mathbf{x}_{ei}\not\models y_e} -\frac{1}{I}\ln\left(\frac{\epsilon}{1-\epsilon}\right) && (5.9)
\end{aligned}
$$

Optimising only for the $\mathcal{S}_{pen}$, FastLAS would return the following solution:

$$H = \arg\min_H \sum_e \sum_{H,\mathbf{x}_{ei}\not\models y_e} e_{pen} \tag{5.10}$$

Hence, by setting the penalty for each example to $-\frac{1}{I}\ln\left(\frac{\epsilon}{1-\epsilon}\right)$ and prior (hypothesis) penalties to 0, FastLAS would return a solution close to the maximum likelihood for all examples.

Integer penalties

FastLAS does not support any floating point number calculations, including penalty values, making the expression derived above unusable.

But, we can overcome this issue. Let us consider a large value $K \in \mathbb{R}^+$, and then round K multiplied by each term in the sum to the nearest integer. Because for large enough K:

$$Kt \approx round(Kt) \tag{5.11}$$

In addition, minimising any $t \in \mathbb{R}^+$ is equivalent to minimising Kt for any $K \in \mathbb{R}^+$. So, the function we wish to minimise becomes:

$$
\begin{aligned}
H_{\text{ML}} &= \arg\min_H \sum_e \sum_{H,\mathbf{x}_{ei}\not\models y_e} -\frac{1}{I}\ln\left(\frac{\epsilon}{1-\epsilon}\right) && (5.9)\\
&= \arg\min_H K\sum_e \sum_{H,\mathbf{x}_{ei}\not\models y_e} -\frac{1}{I}\ln\left(\frac{\epsilon}{1-\epsilon}\right) && \text{(scaling } t\in\mathbb{R}^+ \text{ by } K\in\mathbb{R}^+\text{)}\\
&= \arg\min_H \sum_e \sum_{H,\mathbf{x}_{ei}\not\models y_e} -\frac{K}{I}\ln\left(\frac{\epsilon}{1-\epsilon}\right)\\
&= \arg\min_H \sum_e \sum_{H,\mathbf{x}_{ei}\not\models y_e} \text{round}\left(-\frac{K}{I}\ln\left(\frac{\epsilon}{1-\epsilon}\right)\right) && \text{(by 5.11)} \quad (5.12)
\end{aligned}
$$

Therefore, after choosing some large $K > 0$, we assign the non-coverage penalty of round $\left(-\frac{K}{I} \ln\left(\frac{\epsilon}{1-\epsilon}\right)\right)$ to each example.

5.2.3 Incorporating a Prior over the Hypothesis Space

Now, we can extend the approach done in the previous section by considering a prior over hypothesis space ($p(H)$) too. The extension would give a plausibility value to each potential H before evaluating its fitness on the examples. We can combine this with the likelihood function to give a posterior probability for the data and hypothesis, namely:

$$p(\mathbf{y}, H|\mathbf{X}) = p(\mathbf{y}|H, \mathbf{X})p(H) \tag{5.13}$$

We can log this to get:

$$\ln p(\mathbf{y}, H|\mathbf{X}) = \ln p(\mathbf{y}|H, \mathbf{X}) + \ln p(H) \tag{5.14}$$

Maximising the above equation leads to what we would call the maximum posterior estimate for H, or maximum a posteriori (MAP) estimate:

$$H_{\text{MAP}} = \arg\max_H \ln p(\mathbf{y}|H, \mathbf{X}) + \ln p(H) \tag{5.15}$$

Choosing a Meaningful Prior over the Hypothesis Space

There are several ways to place a prior on hypothesis space. The default ILASP [33] and the usual FastLAS approach assign smaller penalties, i.e., higher prior probabilities, to shorter clauses. Some problems benefited from assigning lower penalties to "ideal length" rules, such as Drozdov et al. [15].

We have opted for a slightly different approach that we can incorporate well into the theory presented thus far.
Starting with a set of potential rules $\mathcal{R}$, let q_r be an independent probability that r is included in H (written $r \in H$) for every potential rule $r \in \mathcal{R}$. Then the prior probability can be written as:

$$p(H) = \left(\prod_{r \in H} q_r\right)\left(\prod_{r \in \mathcal{R} \setminus H} (1 - q_r)\right) \tag{5.16}$$

We can log this for convenience:

$$\ln p(H) = \sum_{r \in H} \ln q_r + \sum_{r \in \mathcal{R} \setminus H} \ln(1 - q_r) \tag{5.17}$$

For a fixed reference hypothesis such as $H_0 = \emptyset$, maximising the posterior is equivalent to maximising the posterior divided by the prior, resulting in:

$$\begin{aligned} H_{\text{MAP}} &= \arg\max_H p(\mathbf{y}|H, \mathbf{X})p(H) \\ &= \arg\max_H \left(\frac{p(\mathbf{y}|H, \mathbf{X})p(H)}{p(H_0)}\right) \\ &= \arg\min_H \left(-K \ln p(\mathbf{y}|H, \mathbf{X}) - K \ln \frac{p(H)}{p(H_0)}\right) \end{aligned} \tag{5.18}$$

The above equation holds for some $K > 0$, added to deal with FastLAS inability to work with non-integers as in 5.2.2.

The ratio of the overall change in prior for any hypothesis $H \in \mathcal{H}$ from the empty hypothesis $H_0 = \emptyset$ can then be calculated using 5.17 as follows:

$$\begin{aligned}\ln\left(\frac{p(H)}{p(H_0)}\right) &= \sum_{r\in H}\ln q_r + \sum_{r\in\mathcal{R}\setminus H}\ln(1-q_r) - \sum_{r\in\emptyset}\ln q_r - \sum_{r\in\mathcal{R}}\ln(1-q_r) \\ &= \sum_{r\in H}(\ln q_r - \ln(1-q_r)) \end{aligned} \tag{5.19}$$

As FastLAS finds an optimal solution with respect to the scoring function $(\mathcal{S}_{pen}+\mathcal{S})$, it can satisfy the equation 5.18 if we choose the examples penalties as in 5.2.2 and the following $\mathcal{S}$:

$$\mathcal{S}(H,T) = -K\sum_{r\in H}(\ln q_r - \ln(1-q_r)) \tag{5.20}$$

By inspection, we see that the decomposition of the function $\mathcal{S}$ is given by:

$$\mathcal{S}^{rule}(r,T) = -K(\ln q_r - \ln(1-q_r)) \tag{5.21}$$

Finally, we need to choose appropriate values probabilities q_c.
Imagine that we have a rule of length l, we would expect to find m_l clauses of length l in a specific hypothesis, and there are n_l clauses of length l in the set of all possible rules $\mathcal{R}$. This setup suggests a value for $q_{r_l} = \frac{m_l}{n_l}$, resulting in the following change to the $\mathcal{S}^{rule}$:

$$\begin{aligned}\mathcal{S}^{rule}(r,T) &= -K(\ln q_r - \ln(1-q_r)) \\ &= -K\left(\ln\frac{m_l}{n_l} - \ln\frac{n_l-m_l}{n_l}\right) \\ &= K\left(\ln(n_l-m_l) - \ln m_l\right) \end{aligned} \tag{5.22}$$

5.3 Implementation

5.3.1 Encoding Binary Classification Task

To encode a solution to a binary classification problem using ASP, we can do the following:

- Write rules such as `label1 :- ...` for one of the labels.
- Add the rule of the form `label2 :- not label1`.

With this approach, we are guaranteed to have precisely one label in each answer set of the task. We always want exactly one answer set as a solution, as we always want just one outcome. To prevent that possibility from occurring, the following types of ASP rules are not used:

- Choice rules — Choice rules give rise to multiple answer sets, which we do not want.

- "Not-chain" — Combination of normal rules which can only be satisfied by multiple answer sets, such as `{p :- not q. q :- not p.}`. The program satisfying the "non-chain" requirement is stratified. This means that its rules can be split into disjoint multiple strata $P = P_0 \cup P_1 \cup ... \cup P_n$ where each strata P_i can only have negations of atoms defined in $P_0 \cup ... \cup P_{i-1}$.
- Constraints — Constraints are used to eliminate an answer set from occurring. If we were to remove the single occurring answer set, we would not get a solution.

An alternative to this encoding which would return a solution with the highest score is also possible. However, it is not nearly as interpretable.

Example 6. Encoding the sudoku learning task:

The sudoku 4x4 learning task (5.4.1) should deduce whether a grid is or is not valid.

1. *Construct the background knowledge:*
The background knowledge for sudoku could contain the following rules that specify which row/column/block a cell belongs to.

```
num(1..4).
col(1..4).
row(1..4).
block(1..4).
% value(C, N) represents that number N is written in cell C
cell(C) :- value(C, _).

col((X, Y), Y) :- row(X), col(Y).
row((X, Y), X) :- row(X), col(Y).
block((X, Y), 1) :- row(X), col(Y), X <= 2, Y <= 2.
block((X, Y), 2) :- row(X), col(Y), X <= 2, Y > 2.
block((X, Y), 3) :- row(X), col(Y), X > 2, Y <= 2.
block((X, Y), 4) :- row(X), col(Y), X > 2, Y > 2.

neq_cell(C1, C2) :- cell(C1), cell(C2), C1 != C2.
```

2. *Account for the alternative label:*
We add the following rule to the background knowledge:

```
selected(valid) :- not selected(invalid).
```

3. *Construct the language bias:*
We chose that we wish to learn what makes a cell invalid in this example, resulting in the following language bias:

```
#modeh(selected(invalid)).

#modeb(row(var(cell), var(row))).
#modeb(col(var(cell), var(col))).
#modeb(block(var(cell), var(block))).
#modeb(neq_cell(var(cell), var(cell))).
#modeb(value(var(cell), var(num))).
```

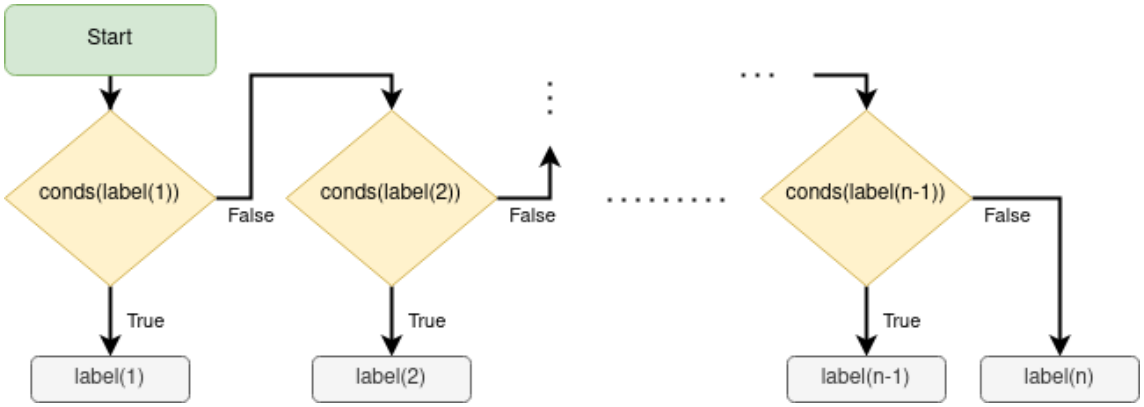
```
#maxv(4).
```

4. *Add examples and prior penalty definitions:* We will discuss this step in the example 8.

5.3.2 Encoding Multi-label Classification Task

The multi-label classification outcome is decided using the flowchart in 5.2, where the `conds(label(x))` predicate is true when conditions for a particular label are satisfied.

Figure 5.2: A flowchart to select an outcome of a multi-label classification problem.



We chose this flowchart structure to aid interpretability because FastLAS learns the following set of rules:

```
conds(label(1)) :- ...
conds(label(1)) :- ...
...
conds(label(n-1)) :- ...
conds(label(n-1)) :- ...
```

So, a human can easily determine the outcome by going through the rules from top to bottom and selecting a label matching the first satisfied rule. Learning these types of rules is encoded in FastLAS with the following syntax:

```
#modeh(conds(const(learnable_label))).

... all the modeb declarations ...

% Avoid constraints
#bias("
:- constraint.
").
```

Furthermore, to make the learner use the logic presented in the flowchart, we need to encode the following:

1. The position of a label in the flowchart chain.
2. Selection of the highest priority label whose conditions are satisfied.
3. Selection of the lowest priority label when no conditions are satisfied.

The enumerated criteria result in the following ASP encoding added to the background of a learning task:

```
% Encoding 1 with label(name, position) (lower=better)
label(l1, 0).
label(l2, 1).
...
label(ln, n-1).

% Definitions of types for FastLAS
label(L) :- label(L, _).
learnable_label(L) :- exists_lower_priority(L, _).


% Encoding 2: Select the highest priority label whose conditions are
   satisfied
selected(L) :- label(L, P), conds(L),
               not higher_priority_selection(L, P).
higher_priority_selection(L, P) :- label(L, P), label(L2, P2),
                                   P2 < P, selected(L2).

% Encoding 3: Select default label if no higher priority selection is
    made
selected(L) :- label(L, P), not higher_priority_selection(L, P),
               not exists_lower_priority(L, P).
exists_lower_priority(L, P) :- label(L, P), label(L2, P2), P2 > P.
```

Notice that the presented encoding also produces exactly one answer set, the same as the encoding for the binary classification.

Example 7. Encoding the multi-class sudoku 4x4 learning task:

Consider an extension of the binary sudoku 4x4 task in the example 6, which introduces an additional label `conflict`. That label should be true only when an example contains two digits written to the same cell. The following steps are needed to solve the task using the presented multi-label classification encoding:

1. *Construct the background knowledge:*
The background is the same as for the example 6.

2. *Add the multi-label selection encoding to the background:*
We add the following rules to the background knowledge:

```
% Encoding 1 with label(name, position) (lower=better)
label(conflict, 0).
label(invalid, 1).
label(valid, 2).
```

```
% Definitions of types for FastLAS
label(L) :- label(L, _).
learnable_label(L) :- exists_lower_priority(L, _).


% Encoding 2: Select the highest priority label whose conditions are
    satisfied
selected(L) :- label(L, P), conds(L),
               not higher_priority_selection(L, P).
higher_priority_selection(L, P) :- label(L, P), label(L2, P2),
                                   P2 < P, selected(L2).

% Encoding 3: Select default label if no higher priority selection is
     made
selected(L) :- label(L, P), not higher_priority_selection(L, P),
               not exists_lower_priority(L, P).
exists_lower_priority(L, P) :- label(L, P), label(L2, P2), P2 > P.
```

2. *Construct the language bias:*
We chose that we wish to learn what makes a cell invalid in this example, resulting in the following language bias:

```
#modeh(conds(const(learnable_label))).

#modeb(row(var(cell), var(row))).
#modeb(col(var(cell), var(col))).
#modeb(block(var(cell), var(block))).
#modeb(neq_cell(var(cell), var(cell))).
#modeb(value(var(cell), var(num))).

#maxv(4).
```

4. *Add examples and prior penalty definitions:* We will discuss this step in the example 8.

5.3.3 Creating the Example File

The example file incorporates the theory presented in 5.2. Recall from 2.2 that the FastLAS task allows defining positive and negative examples. Positive examples are bravely entailed, i.e. the final solution should be extended by at least one answer set. On the other hand, the negative examples are cautiously entailed, so the final solution must not be extended by any answer set. Because of our classification encodings, the learned hypothesis can only ever return one answer set, making the positive examples sufficient to encode any task.

To account for FastLAS's inability to deal with probabilistic atoms, we need to sample I examples, each of the form:

`#pos(example_id@round` $\left(-\frac{K}{I}\ln\left(\frac{\epsilon}{1-\epsilon}\right)\right)$`,`
`{selected(true_label)},`

```
{selected(incorrect_label) for each incorrect_label},
{... context atoms derived from the raw example ...}
```

The constants K, I, and ϵ are currently set to $1000, 100$ and 0.01, respectively.

We further need to encode the prior (hypothesis) penalties. Any custom-defined FastLAS scoring must be decomposable (2.2), and we can only directly input the scoring function decomposition as FastLAS code. Recall that the scoring function decomposition we wish to encode is given by $\mathcal{S}^{rule}(r, T) = K\left(\ln(n_l - m_l) - \ln m_l\right)$. As mentioned in 2.2, defining a scoring function decomposition is done by defining the predicate `penalty/2`. Any logic related to `penalty/2` is defined with atoms `in_head/1` and `in_body/1` which contain all the head and body predicates of a rule r. Using FastLAS's ASP syntax, we define the `penalty` predicate by looking up the penalty value for a rule of a certain length, or assign an extremely large value for undefined length-penalty pairs:

```
#bias("
penalty(P, custom) :- L = #count{X : in_head(X); X : in_body(X)},
                      pen(L, P).
pen(1, K ln(n_1 − m_1) − K ln m_1).
pen(2, K ln(n_2 − m_2) − K ln m_2).
... other similarly defined penalties ...
pen(L, 100000000000) :- L = #count{X : in_head(X); X : in_body(X)},
                        L >= threshold.
").
```

Example 8. Encoding a valid sudoku 4x4 board shown below.

	4		2
1		2	

1. Extract the relevant information from an example. Here we extract the positions of the digits on the board. These can be represented by the predicates:

```
value(2, 2, 4). value(3, 1, 1).
value(2, 4, 2). value(3, 3, 2).
```

where `value` stores row, column and digit information in that order.

2. Generate the example string. The example context will consist of information extracted in step 1, the inclusion of the true label, while the exclusion would contain all of the incorrect labels. Finally, the penalty is computed from round $\left(-\frac{K}{I}\ln\left(\frac{\epsilon}{1-\epsilon}\right)\right)$. If we set the values for constants K, I, and ϵ to $1000, 100$ and 0.01 and consider the binary version of the sudoku task, we get the following example:

```
#pos(id@46,
{ selected(valid) },
{ selected(invalid) },
{
```

```
 value(2, 2, 4). value(3, 1, 1).
 value(2, 4, 2). value(3, 3, 2).
}).
```

3. Generate the prior penalties. Choosing the values for parameters m_l is done through empirical evaluation, while n_l is determined from a language bias. The former is the number of rules of length m_l that we expect in a final solution, while the latter is the total number of rules with length l from the current language bias:

```
#modeh(selected(invalid)).

#modeb(row(var(cell), var(row))).
#modeb(value(var(cell), var(num))).

#maxv(4).
```

Clearly we can have 2 rules of length 1 (head + each body), so n_1 is 2. Also, $n_2 = 2$ since only the following rules exists:

```
selected(invalid) :- row(V3,V4); value(V1,V2).
selected(invalid) :- row(V1,V3); value(V1,V2).
```

They need to be defined so that their types match, are unique, and are not trivially replaceable by another rule. The last requirement means that we do not allow rules such as $selected(invalid)$:- $row(V1, V3), row(V2, V4)$ since row(1, 1) would make this rule true.

Setting empirically determined values $m_1 = m_2 = 1$, results in the following hypothesis penalties.

```
penalty(P, custom) :- L = #count{X : in_head(X); X : in_body(X)}, pen
    (L, P).
pen(1, 0).
pen(2, 0).
pen(L, 100000000000) :- L = #count{X : in_head(X); X : in_body(X)},
    L >= 3.
```

The values for parameters n_l are in practice approximated using $\binom{r}{l}$, the number of possible selections of l predicates from r `#modeb` definitions. Note that this approximation is completely accurate when we only have constant `#modeb` declarations, which is the case for the concept bottleneck pipeline.

Further extensions to this system should make this value more accurate, but it did not hurt performance.

A simple, yet extremely beneficial, implemented performance optimisation is **example aggregation**.
When sampling I values, track each outcome and its number of occurrences before generating the actual examples. The tracking allows to replace C identical examples of penalty round $\left(-\frac{K}{I}\ln\left(\frac{\epsilon}{1-\epsilon}\right)\right)$ with only one example of penalty $C * \text{round}\left(-\frac{K}{I}\ln\left(\frac{\epsilon}{1-\epsilon}\right)\right)$, reducing the overall FastLAS running time.

5.4 Evaluation

The evaluation is carried out on two tasks:

1. Sudoku grid validity
2. MLB-V2E classification

The former is a much simpler task which will evaluate whether the method does indeed work well, while the latter will consider the Prob-FF-NSL framework within the context of a concept bottleneck pipeline.

5.4.1 Sudoku grid learning

This task aims to determine whether a sudoku grid with hand-written digits is valid, i.e. there are no repeated numbers in a block, row or column. It is the repeat of the task studied in [12]. There are two versions of it, 4x4 and 9x9 sudoku grid validity tasks.

The digit values are treated as concepts in this example, while their probabilities are obtained from a digit-predicting neural network.

Given that this task is simple to tackle for a logic-based learning system and a standard CNN, it is explored when different percentages of digit images are subject to a distribution shift. The distribution shift, in this case, is a 90-degree image rotation. It significantly impacts the overall task, reducing the accuracy of digit prediction from 99% to 14%.

We will compare our approach to three baselines: random forest, CNN-LSTM architecture and FF-NSL architecture. The latter similarly uses example penalties to give higher penalties to more likely examples, but Cunnington et al. [12] selected the example penalties through empirical evaluation. It is explained in more detail in 6.6. Moreover, the FF-NSL and Prob-FF-NSL are tested using the same digit predictor network.

The learned models are compared with the true and shifted test sets. The former always presents the correct digit prediction to the framework, while the latter has the same proportion of digit images shifted as the Prob-FF-NSL/FF-NSL training set.

The results for the sudoku 4x4 task are shown in the figure 5.3.

The results demonstrate that the Prob-FF-NSL and FF-NSL can learn the correct solution in spite of a high distribution shift. They cover all of the true test examples even when 70%/80% of examples are subject to a distribution shift. Both approaches have an almost identical performance dealing with a shifted test set. On the other hand, the LSTM-CNN and the random forest fail to learn a correct solution even when given ten times more examples. The models with more examples perform comparably to the FF-NSL approaches on a shifted test set. With only 320 examples, they fail to learn a solution with their performance slightly above 0.5, which is a threshold that a random model would achieve.

The results for sudoku 9x9, presented in 5.4, show similar results.

Figure 5.3: A comparison of the sudoku 4x4 task performance with an increasing level of distribution shifts. Adapted from Cunnington et al. [12]

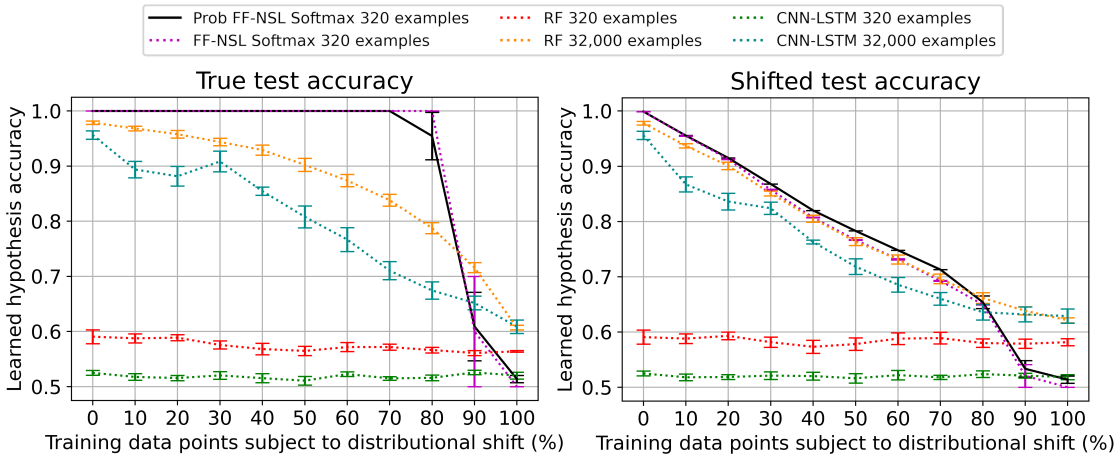
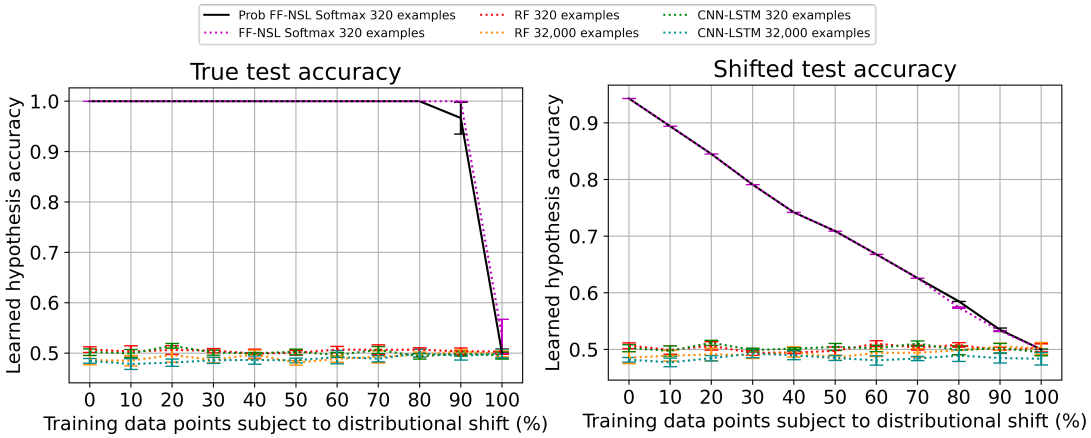


Figure 5.4: A comparison of the sudoku 9x9 task performance with an increasing level of distribution shifts. Adapted from Cunnington et al. [12]



The main difference in performance is present for the random forest and CNN-LSTM baseline models, which failed to learn a good solution.

The model returned the following rules in all cases where the true test accuracy was 100%:

```
selected(invalid) :- neq(V0,V1), neq(V1,V0), value(V0,V2), value(V1,
    V2), row(V0,V3), row(V1,V3), cell(V0), cell(V1), num(V2), row(V3)
    .
selected(invalid) :- block(V1,V0), block(V2,V0), neq(V1,V2), neq(V2,
    V1), value(V1,V3), value(V2,V3), block(V0), cell(V1), cell(V2),
    num(V3).
selected(invalid) :- neq(V0,V1), neq(V1,V0), col(V0,V2), col(V1,V2),
    value(V0,V3), value(V1,V3), cell(V0), cell(V1), col(V2), num(V3).
```

These rules are clearly correct as they outline that no row, block or column can have

two of the same digits.

All in all, the Prob-FF-NSL performs well, having a perfect true test score in spite of 70%/80% of examples under a distribution shift for the sudoku problems. However, it performs slightly worse than the FF-NSL method on this particular task. The advantage of the FF-NSL approach is that it assigns more importance to training examples that are not under the distribution shift. But, the Prob-FF-NSL also accounts for the non-maximum predictions in its training, so the slight difference may only be down to the random seeds used when sampling. Finally, the FF-NSL models performed much better than the others as they incorporated background knowledge.

5.4.2 Concept Bottleneck Pipeline

We also test the presented Prob-NSL framework within the context of a concept bottleneck model. The architecture that the Prob-NSL model used for this task is presented in 5.1.

As baselines, we compare the model to an end-to-end network and the concept bottleneck model with identical architectures. The latter is the same model used for the concept prediction for the Prob-FF-NSL model.

Their results are summarised in the table below:

Test Accuracy Comparison		
End-to-end model	Concept bottleneck model	Prob-NSL concept bottleneck
0.687 ± 0.004	0.685 ± 0.005	0.910 ± 0.023

The Prob-NSL model greatly outperforms the other two with 33% test higher test set accuracy. The results could have been even higher if the concept bottleneck model could distinguish between the labels *strike* and *ball* well. In some instances, the model predicts a *strike* in every case where the true label was *ball*. When the model learns well enough to distinguish between the two, the accuracy jumps to values as high as 96%.

A learned solution can be post-processed into the following form:

```
conds(strike) :- concept("The batter fouled it.").
conds(strike) :- concept("The batter hit the ball into foul territory
    ."), concept("The ball was."), concept("The batter did not swing
    .").
conds(strike) :- concept("The batter made contact."), concept("The
   batter missed."), concept("The umpire called it a ball.").
conds(strike) :- concept("The batter did not swing."), concept("The
   ball was hit.").
conds(strike) :- concept("It was a strike."), concept("The batter did
    not swing.").
conds(foul) :- concept("It was outside the strike zone."), concept("
   The batter hit a fly ball.").
```

```
conds(foul) :- concept("The batter hit the ball into foul territory
   ."), concept("The umpire called it a ball.").
conds(foul) :- concept("The batter hit the ball for a foul ball."),
   concept("The batter did not swing.").
conds(foul) :- concept("The batter hit the ball into foul territory
   ."), concept("The batter hit the ball.").
conds(out) :- concept("The batter hit a fly ball."), concept("It hit
   the ground.").
conds(out) :- concept("The batter hit the ball for a foul ball."),
   concept("It was caught.").
conds(play) :- concept("The batter hit the ball in the air.").
```

At test time, the solver returns the head of the first rule whose rule body is satisfied if reading the rules from top to bottom. If none of these rules is matched, the *ball* label is returned. Such an interpretation allows for a clear link between the concepts and the final labels.

However, inspecting the solution shows that it does not always follow baseball logic precisely. For example, the concept *the batter fouled it* should not result in a strike. The most likely cause of this result is that the neural network does not learn concepts as intended but instead uses them as proxies to incorporate its features. With this in mind, we attribute the difference in performance to the concept prediction architecture of the original models, which only has a single linear layer.

5.5 Discussion

The proposed logic-based classification method works well but fails to learn an interpretably correct solution for the concept bottleneck model. Some cases do not follow human logic. To account for this issue, we should improve upon the concept prediction pipeline, which currently has a precision of 17.5%.

In general, the performance of the logic-based classification method on its own could further be improved by, for example, incorporating the example importance into the choice of the parameters. Moreover, the approach for choosing a prior may not be flexible enough for all possible applications. Incorporating the configurable prior approach shown by Drozdov et al. [15] would have a lot of benefit in some cases.

Chapter 6

Related Work

6.1 Video Classification

Even though the work on end-to-end video classification is not closely related to this thesis's contributions, different neural network architectures significantly impact the concept bottleneck performance.

There are a few popular datasets for video classification benchmarks out there, such as YouTube-8M [1] and Kinetics [7]. The YouTube-8M [1] is an extensive benchmark for general video classification with around 8 million clips and 4800 visual entities. Mao et al. [43] achieved very high performance on the YouTube-8M dataset using a deep convolutional graph neural network and a multi-level feature extractor. Kinetics-700 [8] dataset is the latest version of the popular Kinetics dataset, which contains 700 human action classes and around 650000 video clips. Yan et al. designed the top-performing model for the dataset [69], which uses Multiview Transformers for Video Recognition. The core idea of this approach is to input multiple different "views" of an input video into a transformer model to achieve better results. This approach resulted in a model with a top-1 accuracy of 82.2% and a top-5 accuracy of 95.7% on the Kinetics-700 dataset.

YouTube-8M and Kinetics datasets are somewhat different from the dataset most of the project will address. The MLB-V2E sequences look very similar, given that all of the sequences start with a pitcher throwing a ball.

A dataset much more closely related to our problem is the MLB-YouTube [52] dataset. The segmented video classification part of this dataset is the base for the MLB-V2E [4] dataset, which also includes crowd-sourced explanations. The scenes in the MLB-YouTube dataset are very similar to each other as only a single camera view is used, and the activity itself is only different because of the movement of one person.

The MLB-YouTube paper validates the performance of InceptionV3 [64] and I3D [7] networks on the constructed datasets shown in the table below.

Per-class average precision for the multi-class baseball classification								
Method	Ball	Strike	Swing	Hit	Foul	In Play	Bunt	Hit by Pitch
Random	21.8	28.6	37.4	20.9	11.4	10.3	1.1	4.5
InceptionV3	66.9	93.9	90.3	90.9	60.7	89.7	12.4	29.2
I3D	62.5	91.3	88.5	86.5	47.3	75.9	16.2	21.0

The results suggest that InceptionV3-like architecture is a great candidate to improve upon the NN performance presented in Chapter 4.

6.2 Definition of Concept in Other Settings

There are different angles to the definition of concepts in various works. This project defines a concept as a syntactic generalisation of an atomic sentence (defined in 3.1). However, this approach is not common in other works which do concept extraction.

Formal concept analysis [20] is a method for knowledge representation based on the lattice theory, which can be used to extract hierarchical concepts. It defines two fundamental notions: a formal context and a formal concept. Formal context K := (G, M, I) consists of a set of objects G, a set of attributes M and relation I on (G, M). If (g, m) $\in$ I, object g has attribute m. Using these attributes, set A$'$ is defined containing all attributes common to the objects in a set of objects A. Additionally, set B$'$ contains objects with all attributes in the set of relations B. From these definitions, formal concept of the context (G, M, I) is defined as a pair (A, B) such that A $\subseteq$ G, B $\subseteq$ M, B$'$ = A and A$'$ = B.

The Formal Concept Analysis with text analysis is mainly used to construct the concept hierarchy, where a concept refers to a phrase containing two words.

For example, work by Cimiano et al. [9] extracts verb/subject, verb/object/ and verb/prepositional phrase as candidate concepts. Moreover, the Formal Concept Analysis is used by Anoop et al. [2] to extract concepts with their relationships from unstructured text. The authors manually extract stemmed noun phrases as key phrases and use indications such as "is-a" to generate a formal context table. This table is then used to extract hierarchical concepts.

On the other hand, TaxoLearn [14] by Dietz et al. does not use Formal Concept Analysis but also extracts noun phrases as concepts. The relevance of noun phrases is checked by computing how often they appear in a particular context compared to others. TaxoLearn also uses a hierarchical clustering algorithm to construct a taxonomy.

Moreover, approaches such as the one by Koh et al. [27] define concepts as properties that the image may have. These properties need to be manually-engineered beforehand.

Finally, work such as the one by Fan et al. [18] use expert defined terms, such as *Lecture Presentation for Gastrointestinal Surgery*, as semantic concepts.

6.3 Concept-Based Explanations for Images and Text

Concept-based explanations for images and text is a more straightforward related problem. Koh et al. [27] designed the original concept bottleneck pipeline, which predicts a set of pre-labelled human-engineered concepts before the final label. At test time, a trained network predicts both the final label and the concepts, which one could then use to explain the output class. The model by Koh et al. inspired this project, which automatically mines concepts from text explanations rather than using a set of provided ones. Yeh et al. [70] propose a method that automatically extracts sets of pixels from an image representing valuable concepts. Similarly, Ghorbani et al. [22] suggest another method that automatically captures a set of visual concepts which are meaningful to humans. All outlined methods are based on image concept extraction, unlike this project's concept extraction, which mainly focuses on the events/actions in a baseball video sequence.

6.4 Video Explanation

Another related problem is that of Video Explanations. We can generate the explanations for the MLB-V2E dataset using the logic-based classification (Chapter 5), but those explanations aim to describe the outcome rather than the video itself.

One famous dataset for video understanding is the MSR-VTT dataset [68]. It is a large-scale dataset with more than 10000 video clips in 257 popular categories of a video search engine. Each clip is annotated with roughly 20 sentences. A few approaches to tackling this issue are presented as viable options in the MSR-VTT paper, such as 2D Convolutions, 3D Convolutions, and RNNs. One of the most performant methods on this dataset is CLIP2TV [21], which utilises transformer-based techniques for both video and text representation.

The authors of the MLB-V2E have also constructed the MSR-V2E dataset [4], which uses the clips made available by the MSR-VTT and provides new classification labels and explanations. This dataset was also explored by the inherited work (4.1), obtaining similar results. Instead, we focused on the dataset with a different modality in an attempt to showcase the generality of the new concept bottleneck model.

6.5 Semantic Concept Video Classification

Semantic Concept Video Classification attempts to predict a set of predefined concepts from a video. Predicting a set of predefined concepts from a video is closely related to a part of the pipeline in this project, which indicates which extracted concepts occur before predicting the outcome.

The work by Fan et al. [18] tries to predict semantically defined concepts by medical experts. The paper proposes using salient objects, visually-distinguishable video components that human semantics understands, to model these semantical concepts. Examples of notions salient objects attempt to represent are a face, voice, or a lecture slide. Despite its benefits, the semantics of salient objects is quite simple, and it does not model relationships that happen over multiple frames in a video. Newer work by Fan et al. [17] also incorporates the possibility of a hierarchical concept

classification. The approach also predicts atomic video concepts before constructing the higher-level ones and existing concept ontology.

Assari et al. [3] represent a video by the co-occurrence of the semantic concepts before applying a classifier. The concepts in the work by Assari et al. are also predefined, but they represent a set of events rather than objects.

6.6 Probabilistic Rule Learning

The approach for learning presented in Chapter 5 learns rules whilst accounting for uncertainty.

Learning logical rules in an uncertain setting is closely related to the field of Probabilistic Inductive Logic Programming [54]. The standard ILP aims to find a hypothesis H such that the $covers(e, H, B)$ is true for all positive examples and false for all negative [53]. The most popular way for defining $covers$ is in terms of the semantic entailment relation, i.e. $covers(e, H, B) = true$ iff $B \cup H \models e$. The Probabilistic ILP framework modifies the covers $covers(e, H, B)$ relation such that it becomes the probability that an example e is covered given the hypothesis H and background knowledge B ($P(e|H, B)$).

Focusing on the Probabilistic Rule Learning, ProbFOIL [55] is a system which targets ProbLog [55], a Prolog-like language which further allows clauses to be valid with some probability p. ProbFOIL examples consist of facts associated with target probabilities that the rule-learner aims to achieve. A further extension of this system, ProbFOIL$^+$ [56], makes it possible to learn probability values and allows specifying the space of the possible clauses the system should search within. Another extension of ProbFOIL is presented in [65], where background knowledge can have negative loops.

However, all of these approaches are not nearly scalable enough for the problem in this thesis, mainly having been evaluated on small sets of examples. In addition, the presented approaches allow specifying the target probability for an example, which is not a requirement for this project. We need to produce as likely of a solution as possible, which for the ProbFOIL-like learners would mean that each example should have a target probability of 1.

A much more scalable approach with a similar goal is presented in the NSL [13] and FF-NSL [12] papers. It is related to the previously mentioned approaches as it also aims to determine a hypothesis given a set of probabilistic facts. Cunnington et al. wish to cover a set of examples generated by the neural network, whose correctness depends on the quality of the NN predictions. Each example consists of facts generated by multiple NN predictions. Maximum prediction probabilities of an example are combined into one example probability using Gödel's t-norms, i.e. the probability of an example is the minimum of prediction probabilities. The FF-NSL framework utilises existing ILASP [25] and FastLAS [34] ILP systems to learn a hypothesis. To account for the example probability, it uses noise penalties these two systems provide. A penalty assigned is proportional to the example probability, with the exact value determined empirically. With this approach, Cunnington et al. specify the importance of satisfying each example using the noise penalties.

Chapter 7

Conclusion

This paper demonstrates that a concept bottleneck model can be combined with human-generated explanations to improve NN explainability. In doing so, we have discovered the following findings:

1. **Logic-based methods can effectively be used to tackle some NLP seq2seq problems**. Such an approach was crucial as the datasets were tiny, consisting of only around 100 examples per problem. The dataset size made it impossible to use state-of-the-art NLP techniques such as transformers. The solutions made by hand-crafting or learning a set of rules, on the other hand, could achieve good performance despite the dataset size. However, the limitation is the lack of scalability of the underlying ILASP [25] system used to learn a set of rules. It was applied successfully on the generalisation task, while it was far off its theoretical optimality guarantee for the atomisation task. When we applied it successfully, it did produce a better solution than a hand-crafted set of rules.

2. **The newly proposed CoDEx pipeline significantly outperforms its predecessor**. We have replaced the extraction part of the original pipeline with the new atomisation and generalisation stages. With them, the new set of concepts conveys twice as much information about the final label compared to its predecessor. There is also an indication that the concepts produced by the new pipeline provide better explanations as the qualitative analysis preferred them in all of the cases tested. However, the CoDEx module still has room for improvement, as it cannot account for concepts occurring in the video but not explicitly in the explanation. This limitation makes applying the sequential training procedure challenging, which splits the training of concept and label prediction parts of the network. Sequential training would be preferred, given that the joint training may not always truly use concepts to predict the results [44]. We further show that applying the method to a dataset of a different modality was possible with no difference to the concept extraction pipeline. The same experiment demonstrated that the CoDEx pipeline could produce highly accurate results when applied to a dataset with general image descriptions. The quality of the explanations was worse since the well-mined concepts that not labelled in most images where they occurred. So, we believe that the proposed method should work in any domain where the human-generated ex-

planation is written to describe why the data point should be classified with a particular label. Such a style of writing presents relevant pieces of information much more consistently.

3. **The proposed logic-based classification produces high-performing and easily explainable solutions**. When applied to the sudoku validity dataset, we show that the method produces a completely accurate solution even when 80% of training examples are subject to the distribution shift. Additionally, applying the method to the concept bottleneck pipeline helped increase the model performance whilst providing a clear link between the predicted concepts and the final label.

7.1 Future work

There are several directions possible to take this project further. We highlight some of the ideas for doing so in this section.

The most pressing concern is the CoDEx sparsity. The human-generated explanations do not capture all of the concepts present in the concept matrix explicitly. For example, both *The ball was caught by the outfielder* and *The outfielder got the pitch* would be extracted as separate concepts. These sentences could have been grouped with some combination of hyper-parameters. However, finding such values often had an undesirable knock-on effect. For example, *The ball was outside of the strike zone* and *The ball was inside of the strike zone* would almost always be grouped before the first two sentences are.
To mitigate that issue, we might use models that solve the semantic entailment problem. The semantic entailment task aims to determine whether some sentence S could be inferred by a sentence T. Such information would allow us to group *The ball was caught by the outfielder* and *The outfielder caught the pitch* seamlessly [59].

Another possible direction could involve improving upon the video classification network. We could enhance the current neural network architecture. It consists of image features extracted by ResNet [23], from which motion features are captured using 1D convolution. Much better architectures do exist, such as the ones discussed in 6.1. We expect that an improved architecture should improve concept prediction performance, which in turn improves the final class prediction.

Comparing the performance with the original concept bottleneck [27] would also be beneficial. This work extends upon the original idea by mining the concepts from explanations. The original work uses a human-engineered set of valuable concepts, which they applied to the CUB-birds dataset [66]. Comparing with the same dataset would give more insight into the generality of the concepts extracted by the CoDEx pipeline. Moreover, the concept explanation quality could be evaluated using a human study, such as a Mechanical Turk analogous to [4]. Such a study would give more confidence regarding the quality of generated explanation. In addition, we should also construct a human study interpreting the logic-based classification framework.

Finally, we could improve upon the atomisation procedure. The atomisation proce-

dure is the main culprit behind any concepts that are invalid sentences. With the Jaccard score at around 0.55, the atomisation task has a room for improvement. A possible approach for handling this issue may involve trying to solve this problem using a seq2seq transformer such as T5 [57]. It would require greatly extending the existing dataset, but it would undoubtedly perform better with a sufficient number of examples.

Appendix A

Ethics

The following section is based on the ethics checklist provided by the Computing Department. The checklist consists of a series of yes/no questions. If an answer was yes, there is an ethical issue that needs to be considered. These answers will form a basis for the discussion in this chapter.

"Does the project involve human participants?" The project will involve human participants in two ways:

- For result validation. Some results obtained in the project have been obtained with human participants, e.g. generated explanation correctness.

- For dataset construction. The datasets used for this project, MLB-V2E [4] and MSR-V2E [4], used human-generated explanations.

Both cases are not harmful to anyone involved. The only care that needs to be taken is regarding the personal information of the subjects involved, which is discussed in the following question. Additionally, the participants involved in the dataset construction were compensated 15$ per hour, as discussed in the paper under review [4].

"Does your project involve personal data collection and/or processing?"

The dataset construction required subjects to explain a video in their own words. But, this data is entirely anonymous and cannot be de-anonymised. No explanation that is provided is in any way, shape or form personally identifiable information so that it can be linked back to a specific person.

Any data acquired from new participants during this project will be fully anonymised as well.

Will your project use or produce software for which there is a copyrighting licensing implication? The project uses three different, third-party pieces of software. These are *spacy* [60], *FastLAS* [34], *ILASP* [33], and *clingo* [10]. *FastLAS*, *spacy* and *clingo* use the MIT License [45]. The MIT License is a permissive license that does not block any future publication and only requires the preservation of copyright and license notices. On the other hand, *ILASP* is free for use for university research, but

it will require reaching out to Mark Law for commercial purposes [25]. The licenses are not an issue as it stands.

To sum up, this project has no outstanding ethical issues which need to be resolved.

Appendix B

Concept Bottleneck Model

This appendix presents networks used in Chapter 4. In order to reproduce the results similar to this project, one needs to train them for 100 epochs, using the adam optimiser, and the batch size of 32.
The number next to layer name, such as `5 Dense`, represents the number of outputs of that layer. The network for the MLB-V2E dataset is shown in B.1 and B.2, while the bird-flowers datset network is in B.3.

Figure B.1: MLB-V2E baseball network part 1

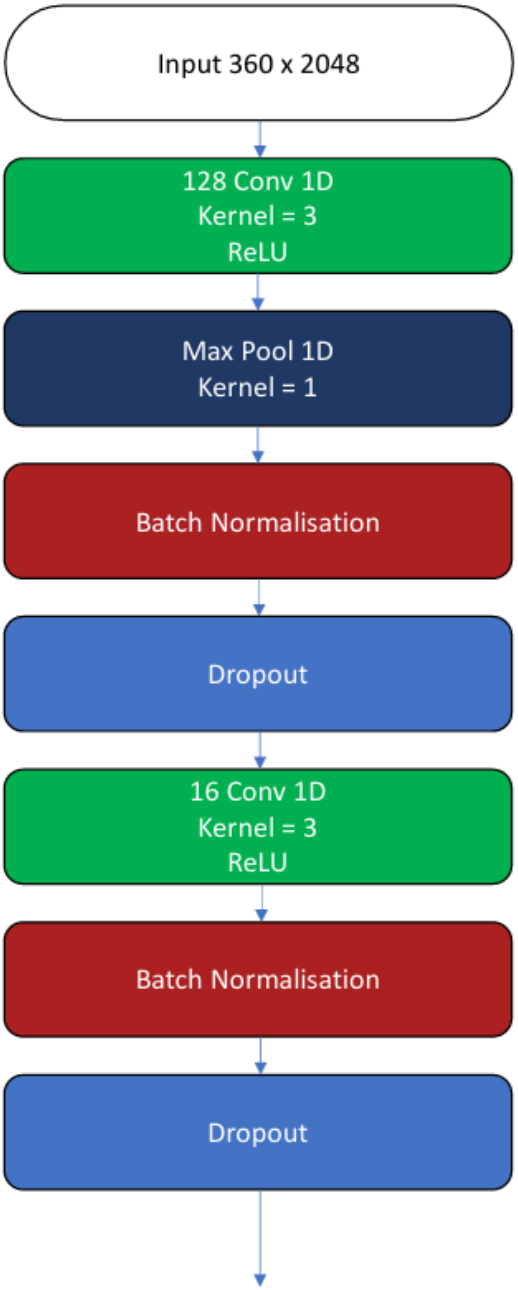


Figure B.2: MLB-V2E baseball network part 1

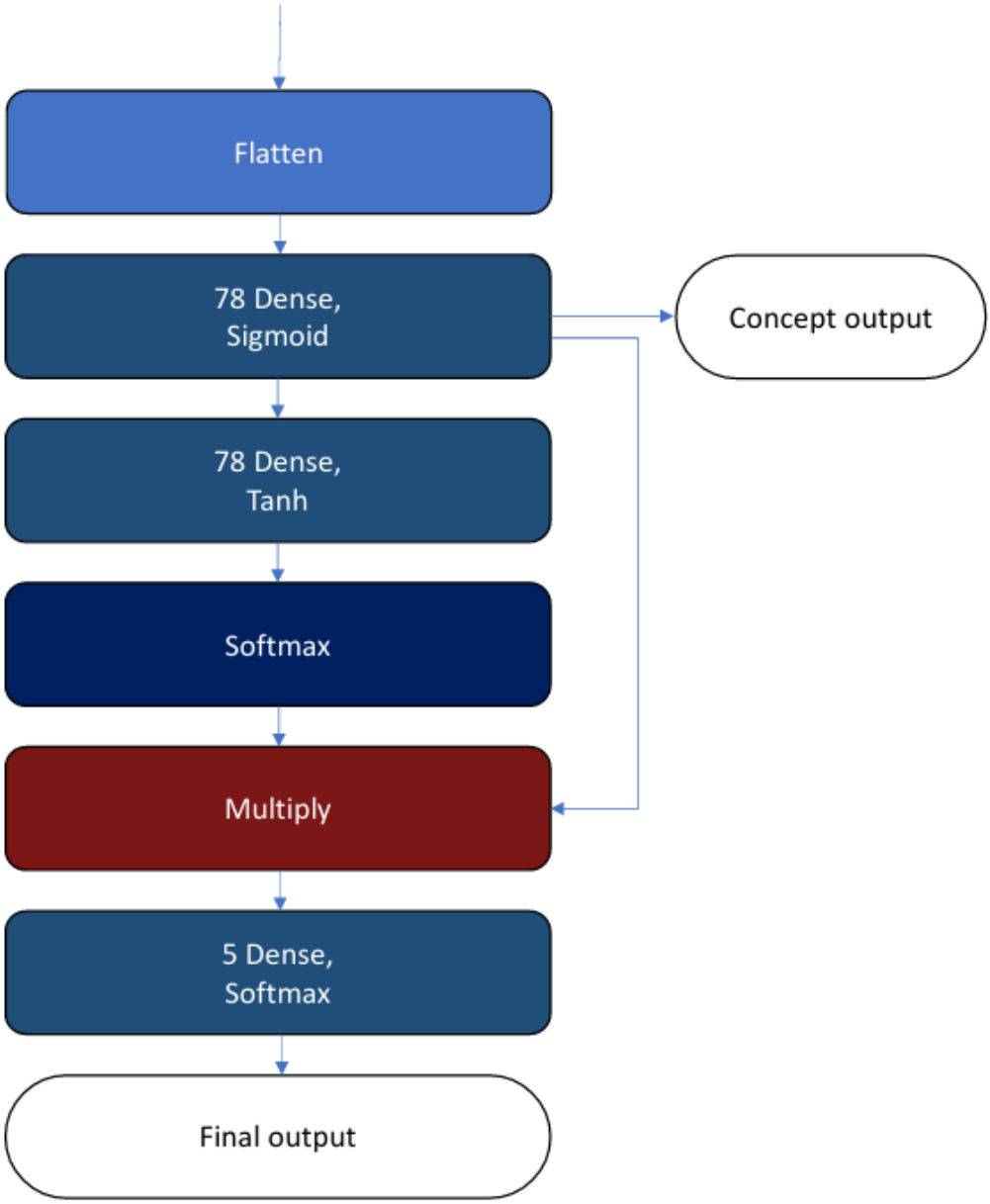
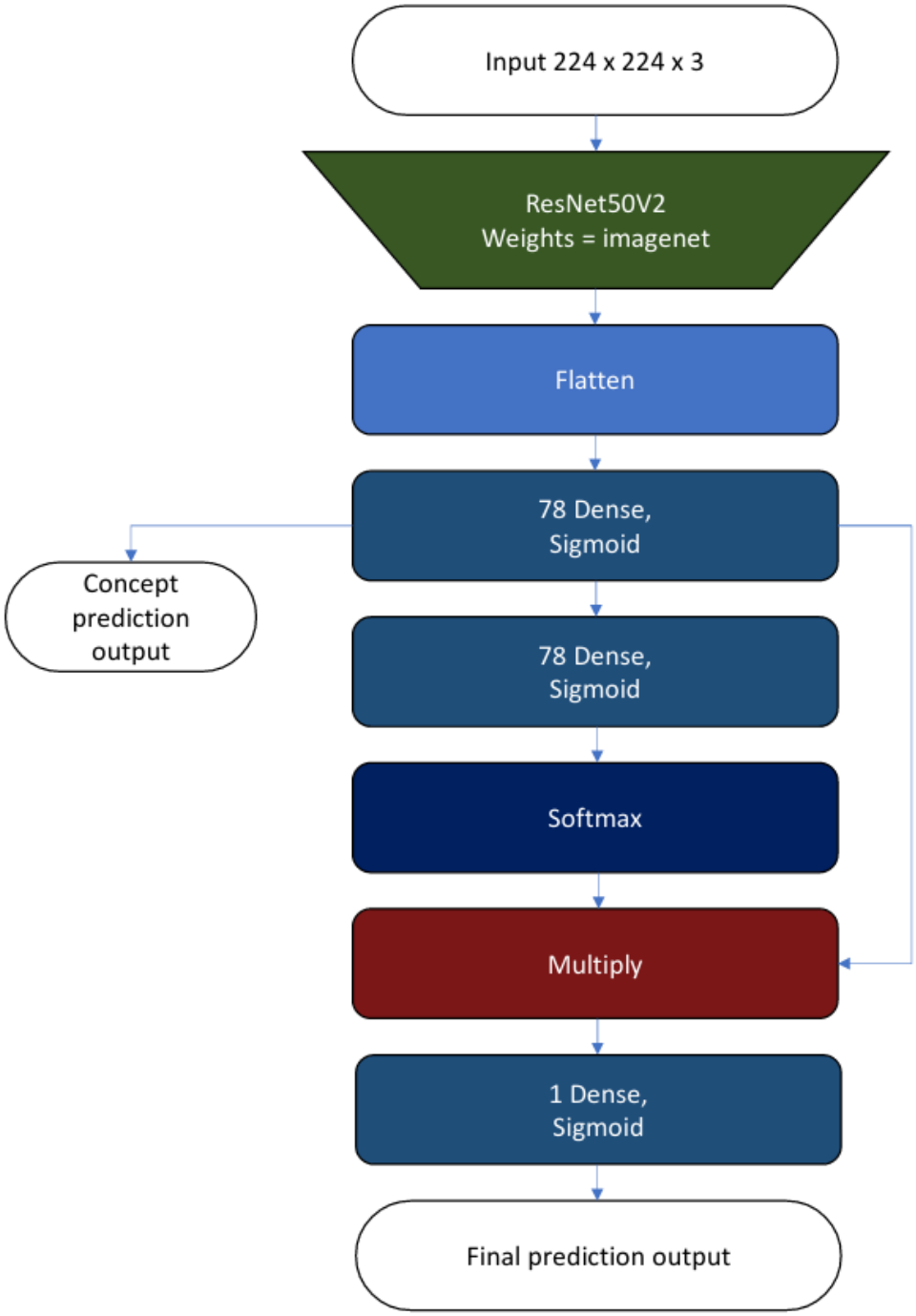


Figure B.3: MLB-V2E baseball network part 1



Appendix C

PyLASP scripts

We show the PyLASP scripts used for running the learning tasks. Atomisation:

```
#ilasp_script
import time

ilasp.cdilp.initialise()
solve_result = ilasp.cdilp.solve()

ilasp.stats.print_new_iteration()
debug_print('Searching for counterexample...')

c_egs = None
if solve_result is not None:
  c_egs = ilasp.find_all_counterexamples(solve_result)

conflict_analysis_strategy = {
  'positive-strategy': 'single-ufs',
  'negative-strategy': 'single-as',
  'brave-strategy':  'single-ufs',
  'cautious-strategy': 'single-as-pair'
}

start_time = time.time()
max_time = 15 * 60 * 60 # 15 hours
best_solve_result = solve_result
best_score = sum(list(map(lambda x: x['penalty'], c_egs)))

print(solve_result)

while c_egs and solve_result is not None and (time.time() -
   start_time) < max_time:

  ce = ilasp.get_example(c_egs[0]['id'])
  debug_print('Found', ce['type'], 'counterexample:', ce['id'], '(a
     total of', len(c_egs), 'counterexamples found)')
```

```
constraint = ilasp.cdilp.analyse_conflict(solve_result['hypothesis
    '], ce['id'], conflict_analysis_strategy)

# An example with recorded penalty of 0 is in reality an example
    with an
# infinite penalty, meaning that it must be covered. Constraint
    propagation is,
# therefore, unnecessary.
if not ce['penalty'] == 0:
  c_eg_ids = list(map(lambda x: x['id'], c_egs))
  debug_print('Computed constraint. Now propagating to other
      examples...')
  prop_egs = []
  if ce['type'] == 'positive':
    prop_egs = ilasp.cdilp.propagate_constraint(constraint,
        c_eg_ids, {'select-examples': ['positive'], 'strategy': '
        cdpi-implies-constraint'})
  elif ce['type'] == 'negative':
    prop_egs = ilasp.cdilp.propagate_constraint(constraint,
        c_eg_ids, {'select-examples': ['negative'], 'strategy': 'neg
        -constraint-implies-cdpi'})
  elif ce['type'] == 'brave-order':
    prop_egs = ilasp.cdilp.propagate_constraint(constraint,
        c_eg_ids, {'select-examples': ['brave-order'], 'strategy': '
        cdoe-implies-constraint'})
  else:
    prop_egs = [ce['id']]

  ilasp.cdilp.add_coverage_constraint(constraint, prop_egs)
  debug_print('Constraint propagated to:', prop_egs)

else:
  ilasp.cdilp.add_coverage_constraint(constraint, [ce['id']])

solve_result = ilasp.cdilp.solve()

if solve_result is not None:
  debug_print('Found hypothesis:', solve_result['hypothesis'],
      solve_result['expected_score'])
  debug_print(ilasp.hypothesis_to_string(solve_result['hypothesis
      ']))

  print("", flush=True)

  ilasp.stats.print_new_iteration()
  debug_print('Searching for counterexample...')
```

```
    c_egs = ilasp.find_all_counterexamples(solve_result)
    score = solve_result['expected_score'] + sum(list(map(lambda x: x
       ['penalty'], c_egs)))
    debug_print("Previous hypothesis score: ", score)
    if best_score == -1 or best_score > score:
      best_score = score
      best_solve_result = solve_result
      debug_print("Best hypotesis so far found")


if solve_result:
  debug_print('\n\nFinal Hypothesis:\n\n')
  print(ilasp.hypothesis_to_string(best_solve_result['hypothesis']))
else:
  print('UNSATISFIABLE')

ilasp.stats.print_timings()

#end.
```

Generalisation:

```
#ilasp_script
import time

ilasp.cdilp.initialise()
solve_result = ilasp.cdilp.solve()

ilasp.stats.print_new_iteration()
debug_print('Searching for counterexample...')

c_egs = None
if solve_result is not None:
  c_egs = ilasp.find_all_counterexamples(solve_result)

conflict_analysis_strategy = {
  'positive-strategy': 'single-ufs',
  'negative-strategy': 'single-as',
  'brave-strategy':  'single-ufs',
  'cautious-strategy': 'single-as-pair'
}

start_time = time.time()
max_time = 12 * 60 * 60 # 12 hours
best_solve_result = solve_result
best_score = sum(list(map(lambda x: x['penalty'], c_egs)))

while c_egs and solve_result is not None and (time.time() -
   start_time) < max_time:
```

```
ce = ilasp.get_example(c_egs[0]['id'])
debug_print('Found', ce['type'], 'counterexample:', ce['id'], '(a
    total of', len(c_egs), 'counterexamples found)')

constraint = ilasp.cdilp.analyse_conflict(solve_result['hypothesis
    '], ce['id'], conflict_analysis_strategy)

# An example with recorded penalty of 0 is in reality an example
    with an
# infinite penalty, meaning that it must be covered. Constraint
    propagation is,
# therefore, unnecessary.
if not ce['penalty'] == 0:
  c_eg_ids = list(map(lambda x: x['id'], c_egs))
  debug_print('Computed constraint. Now propagating to other
      examples...')
  prop_egs = []
  if ce['type'] == 'positive':
    prop_egs = ilasp.cdilp.propagate_constraint(constraint,
        c_eg_ids, {'select-examples': ['positive'], 'strategy': '
        cdpi-implies-constraint'})
  elif ce['type'] == 'negative':
    prop_egs = ilasp.cdilp.propagate_constraint(constraint,
        c_eg_ids, {'select-examples': ['negative'], 'strategy': 'neg
        -constraint-implies-cdpi'})
  elif ce['type'] == 'brave-order':
    prop_egs = ilasp.cdilp.propagate_constraint(constraint,
        c_eg_ids, {'select-examples': ['brave-order'], 'strategy': '
        cdoe-implies-constraint'})
  else:
    prop_egs = [ce['id']]

  ilasp.cdilp.add_coverage_constraint(constraint, prop_egs)
  debug_print('Constraint propagated to:', prop_egs)

else:
  ilasp.cdilp.add_coverage_constraint(constraint, [ce['id']])

solve_result = ilasp.cdilp.solve()

if solve_result is not None:
  debug_print('Found hypothesis:', solve_result['hypothesis'],
      solve_result['expected_score'])
  debug_print(ilasp.hypothesis_to_string(solve_result['hypothesis
      ']))
  print("", flush=True, end="")
  ilasp.stats.print_new_iteration()
  debug_print('Searching for counterexample...')
```

```
    c_egs = ilasp.find_all_counterexamples(solve_result)
    score = solve_result['expected_score'] + sum(list(map(lambda x: x
        ['penalty'], c_egs)))
    if best_score == -1 or best_score > score:
      best_score = score
      best_solve_result = solve_result


if best_solve_result:
  debug_print('\n\nFinal Hypothesis:\n\n')
  print(ilasp.hypothesis_to_string(best_solve_result['hypothesis']))
else:
  print('UNSATISFIABLE')

ilasp.stats.print_timings()

#end.
```

Appendix D

Sample ILASP Learned Hypothesis

```
 :- dep(relcl, V1, V2).
 :- dep(acl, V1, V2).
in_generalised_sent(V1) :- dep(dobj, V2, V1).
in_generalised_sent(V1) :- dep(attr, V2, V1).
in_generalised_sent(V1) :- dep(nsubj, V2, V1).
in_generalised_sent(V1) :- dep(nsubj, V1, V2).
in_generalised_sent(V1) :- dep(neg, V2, V1).
in_generalised_sent(V1) :- dep(oprd, V2, V1).
in_generalised_sent(V1) :- dep(nsubjpass, V2, V1).
in_generalised_sent(V1) :- dep(nsubjpass, V1, V2).
in_generalised_sent(V1) :- dep(mark, V2, V1).
in_generalised_sent(V1) :- dep(prt, V2, V1).
in_generalised_sent(V1) :- in_generalised_sent(V2); dep(det, V2, V1).
in_generalised_sent(V1) :- in_generalised_sent(V2); dep(prep, V1, V2)
    .
in_generalised_sent(V1) :- in_generalised_sent(V2); dep(pobj, V1, V2)
    .
0 {in_generalised_sent(V1) } 1 :- dep(pobj, V2, V1).
in_generalised_sent(V1) :- in_generalised_sent(V2); dep(advmod, V2,
    V1).
in_generalised_sent(V1) :- in_generalised_sent(V2); dep(aux, V2, V1).
in_generalised_sent(V1) :- in_generalised_sent(V2); dep(case, V2, V1)
    .
in_generalised_sent(V1) :- in_generalised_sent(V2); dep(auxpass, V2,
    V1).
in_generalised_sent(V1) :- in_generalised_sent(V2); dep(compound, V2,
     V1).
0 {in_generalised_sent(V1) } 1 :- in_generalised_sent(V2); dep(amod,
    V2, V1).
0 {in_generalised_sent(V1) } 1 :- in_generalised_sent(V2); dep(poss,
    V2, V1).
```

Appendix E

CoDEx Mined Concepts

The dominant concept strings along with the frequencies. Note that the frequencies are computed such as a sum of concept counts, rather than the number of data-points where the grouped concepts occurred.

Figure E.1: Old extracted bird/flowers concepts part 1

final_id	freq	text
0	261	that are red with yellow stamen
1	98	the bird is yellow and green with a small black beak
2	37	the flower shown
3	33	that are oval shaped
4	33	that are white with yellow centers
5	13	the petals of this flower are white with a short stigma
6	11	that are white with shades of pink
7	10	bell shaped
8	16	this bird is almost all white with black primaries yellow tarsus and feet and beak
9	9	that are rounded
10	8	the petals of this flower are pink with a long stigma
11	7	that are pointed at the tips
12	6	that are ruffled
13	6	that are wavy and ruffled
14	6	that are softsmooth and separately arranged around stamens
15	6	this flower has a bunch of light pink petals with no visible stamen on it
16	6	this flower is red in color with only one large petal
17	6	that are multi colored
18	10	the bird has a black bill and a black crown and eyering
19	7	star shaped
20	6	sepals are green in color
21	5	that have veins
22	5	that are oddly shaped
23	5	that are spotted
24	4	that are ruffled on the edges
25	4	that are spotted on the inside
26	4	trumpet shaped
27	4	that are leaf like
28	8	this bird has a white belly gray breast and medium gray wings
29	8	this is a black and white bird with a long black beak
30	7	that is black
31	3	that are green near the center
32	3	that has yellow anthers
33	3	that are curled at the edges
34	3	that are very skinny
35	3	that are orange with orange stigma
36	3	that are closely wrapped around the center
37	3	that are wavy and rounded
38	3	that are layered
39	3	cup shaped
40	3	that are wavy
41	3	the petals of this flower are purple with a long stigma
42	5	this bird is grey with brown wings and a black crown
43	5	rust colored
44	3	that are white in color
45	3	oval shaped
46	2	that are overlapping and red with green pedicel
47	2	that are large and pink with a yellow stigma
48	2	stamen sticking out from the centre
49	2	that are dark pink with white centers
50	2	that overlap and stamen
51	2	that are drooping downward
52	2	that are burgundy with white spots
53	2	a protruding yellow pollen tube surrounded by one curved pointed white petal with smooth edges
54	2	that are orange with green pedicel
55	2	stamens forming bowl shape
56	2	this flower has large white petals and no stigma in the center of it
57	2	that are bunched together
58	2	it is a flower with many white pedals and lots of yellow anther
59	2	that are pale purple with white filaments and purple anthers
60	2	that are pink and folded together

Figure E.2: Old bird concepts part 2

61	2	that are red with green stamen
62	2	that are purple ruffled around the edges
63	2	that are fluffy in appearance
64	2	that are soft smooth thin and arranged separately around pistil
65	2	needle shaped
66	2	that are purple with white and yellow shading and dark lines
67	2	are all connected
68	2	that are curled inward
69	2	that are blue with purple lines
70	2	that are pointed on the ends
71	2	that are ruffled and bunched together
72	2	the petals of this flower are orange with a long stigma
73	2	the flower has long clear pollen tubes and a large white petal
74	2	that are multicolored
75	2	that are vertically layered
76	5	the beak is pointy and sharp
77	4	the bird has a yellow breast and belly as well as a small bill

Figure E.3: Newly extracted bird/flowers concepts part 1

final id	freq	text
0	909	This flower has petals.
1	90	The petals are.
2	202	This flower is.
3	217	This bird has a belly.
4	145	This bird has a beak.
5	132	This is a bird.
6	92	The bird has a bill.
7	30	The flower is.
8	261	That are.
9	20	This flower is orange in color with petals.
10	29	The bird is.
11	24	This bird has a body.
12	134	This bird has a breast.
13	61	This bird has.
14	16	This flower has stamen.
15	9	A bird with wingbars.
16	24	This bird has a bill.
17	18	The bird has a body.
18	31	This bird has a head.
19	11	The bird has.
20	38	A bird with a crown.
21	23	This flower has.
22	33	A bird with a head.
23	14	The beak is.
24	13	The flower has.
25	7	They are.
26	102	This flower has pistil.
27	22	A bird with feathers.
28	8	The body is.
29	50	This bird has wings.
30	40	A bird with wings.
31	27	The bird has a eyering.
32	14	This bird has feathers.
33	11	The bill is.
34	48	This bird has a crown.
35	6	A flower with stamen.
36	3	This flower has a pollen tube.
37	3	That is fused.
38	36	That is.
39	37	A bird with throat.
40	6	A small bird with a tail.
41	5	A bird with markings.
42	13	A bird with feet.
43	4	This bird looks.
44	3	A bird with spots.
45	4	It's.
46	8	A bird with a cheek patch.
47	18	It has beak.
48	5	The head is.

Figure E.4: Newly extracted bird/flowers concepts part 2

49	4	A bird with red eyes.
50	4	This is.
51	3	A flower with anther.
52	9	Which are.
53	4	A bird with coverts.
54	10	With wings.
55	3	White striped.
56	3	The bird has a wingspan.
57	3	He has.
58	8	That has.
59	13	This flower is orange.
60	3	White in color.
61	8	This flower has a center.
62	4	That are stamens.
63	3	Sepals are green in color.
64	3	This flower has anther.
65	3	The flower has a green.
66	11	This bird has with nape.
67	30	This bird has a white breast.
68	13	A bird with black wings.
69	11	The bird has tarsus.
70	6	With belly.
71	3	This bird has eyes.
72	5	The belly is.
73	3	The wings are.
74	4	A bird with hues.
75	7	This bird has primaries.
76	3	The crown is.
77	3	That has feathers.

Figure E.5: Old extracted baseball concepts part 1

final_id	freq	text
0	226	it was outside the strike zone
1	32	the ball was not thrown through the strike zone
2	34	it was called a ball
3	27	the umpire called it a ball
4	1391	the batter did not swing
5	30	it was a strike
6	24	the umpire called it a strike
7	19	this is a foul ball
8	28	the ball went out of play
9	10	the pitch was out of the strike zone
10	18	it was below the strike zone
11	10	it was called a ball by the umpire
12	10	the batter is out
13	9	it was caught for an out
14	8	who caught the ball in the air
15	8	it was caught in the air
16	11	it was caught
17	8	he swung and missed
18	10	the umpire ruled it a strike
19	11	it was a home run
20	15	the pitch was called a ball by the umpire
21	11	the batter's knees
22	23	the batter made contact
23	13	it was caught by the catcher
24	48	the pitcher threw the ball
25	9	the ball landed outside of the strike zone without being swung at by the batter
26	6	the pitch was not in the strike zone
27	7	it was not in the play of field
28	9	the pitch went through the strike zone
29	13	it was a hit
30	8	the ball was hit
31	16	the hitter hit the ball
32	17	the ball was not caught
33	9	it went into foul territory
34	7	it was not a strike
35	11	the umpire called the pitch a ball
36	5	which made it a strike
37	7	the batter hit the ball back to the pitcher
38	4	the batter flew out to center
39	6	the batter missed it
40	4	the catcher's glove
41	7	it was out of bounds
42	5	the ball landed outside of fair play without being caught
43	7	the umpire did not signal for a strike
44	3	he catches it in the air for the out
45	7	it is a foul
46	4	however the ball did not land in the field of play and is considered foul
47	3	the batter swung but missed resulting in the 3rd strike and a strike out
48	5	the ball did not cross the plate inside of the strike zone and was called a ball by the umpire
49	5	who threw him out at first
50	8	it was not caught
51	7	the ball was over the plate
52	4	it was ruled strike three
53	3	the ball was in the center of the strike box
54	3	he hit a foul ball
55	3	the ball flew over the left field fence resulting in a home run for the batter
56	4	who caught it

Figure E.6: Old extracted baseball concepts part 2

57	4	the ball was low
58	4	it was low and outside of the strike zone
59	3	it hit the dirt and got away from the catcher
60	3	it goes too wide to the left of the batter away from the strike zone
61	3	which went out of play
62	4	which resulted in an out
63	3	you hit a foul ball with 2 strikes
64	3	it went behind him
65	3	it went backwards and out of play
66	3	the hitter did not swing and just took the pitch
67	3	the catcher caught it
68	3	a fielder's choice
69	3	the ball was hit in play on the ground by the batter
70	3	he threw runner out at first
71	3	the ball hit out to center field and was not caught
72	3	it was fielded by the shortstop
73	3	it hit the ground before being fielded
74	3	it is a called strike
75	3	it was above the strike zone
76	3	the pitch went low
77	3	it was not in the strike zone

Figure E.7: Newly extracted baseball concepts part 1

final_id	freq	text
0	273	The batter did not swing.
1	119	The batter missed.
2	101	It was caught.
3	797	The batter hit the ball.
4	24	The batter missed the ball.
5	135	It was.
6	49	The batter hit the ball into foul territory.
7	78	The batter hit the ball on the ground.
8	59	It hit the ground.
9	31	The batter hit a ground ball.
10	87	The ball was.
11	78	The batter made contact.
12	97	The batter hit it.
13	30	It was a strike.
14	22	The umpire called it a strike.
15	54	The batter hit the ball in the air.
16	53	The ball was hit.
17	27	The fielder caught the ball.
18	45	It was a ball.
19	19	The batter hit the ball for a foul ball.
20	20	The ball was in the strike zone.
21	71	It went.
22	26	The hitter hit the ball.
23	11	Who caught the ball.
24	15	The outfielder caught the ball.
25	27	It landed.
26	44	The ball landed.
27	44	The ball went.
28	45	The batter hit the ball into field.
29	36	It was hit.
30	15	It was in the strike zone.
31	23	The umpire called it a ball.
32	10	The fielder caught it.
33	34	The batter hit the ball out of play.
34	15	The hitter swung.
35	9	The ball was caught.
36	46	The pitcher threw the ball.
37	15	The ball flew.
38	21	It bounced.
39	12	He swung.
40	18	The batter hit a fly ball.
41	25	It was out of the strike zone.
42	22	The pitch was.
43	17	The ball crossed the plate.
44	7	The batter did not make contact.
45	6	A ball settled on foul territory.
46	6	The pitch was called a strike.
47	17	The batter is.
48	6	Resulting in a ball.

Figure E.8: Newly extracted baseball concepts part 2

49	11	It was not caught.
50	15	The ball was not caught.
51	6	The outfielder caught it.
52	10	The ball hit the ground.
53	6	This is labeled as a foul.
54	9	The umpire called the pitch a ball.
55	5	The batter to center.
56	11	It was not.
57	7	The umpire ruled it a strike.
58	3	It was caught outside of the strike zone.
59	8	The ball went into the strike zone.
60	6	This resulted.
61	13	It was fielded.
62	3	The ball was in territory.
63	7	The ball crossed.
64	4	It passed through the strike zone.
65	7	The ball crossed home plate.
66	6	The batter leaves the ball.
67	6	It crossed the plate.
68	12	The ball bounced.
69	13	The umpire ruled it.
70	84	The batter did not swing at the pitch.
71	11	The hitter did not swing.
72	12	The batter hit the ball into center field.
73	4	The center fielder caught the ball.
74	11	Who caught it.
75	4	The batter fouled it.
76	6	The batter hit a pop up.
77	3	The center fielder caught it.

Bibliography

[1] Sami Abu-El-Haija et al. "YouTube-8M: A Large-Scale Video Classification Benchmark". In: (Sept. 2016). URL: https://arxiv.org/abs/1609.08675.

[2] V. S. Anoop and S. Asharaf. "Extracting Conceptual Relationships and Inducing Concept Lattices from Unstructured Text". In: *Journal of intelligent systems* 28.4 (Sept. 2019), pp. 669–681. DOI: 10.1515/jisys-2017-0225. URL: http://www.degruyter.com/doi/10.1515/jisys-2017-0225.

[3] Shayan Modiri Assari, Amir Roshan Zamir, and Mubarak Shah. "Video Classification Using Semantic Concept Co-occurrences". In: IEEE, Jun 2014, pp. 2529–2536. ISBN: 1063-6919. DOI: 10.1109/CVPR.2014.324. URL: https://ieeexplore.ieee.org/document/6909720.

[4] *Automatic Concept Extraction for Concept Bottleneck-based Video Classification*. Oct. 2021. URL: https://openreview.net/pdf?id=66kgCIYQW3.

[5] Laura Banarescu et al. "Abstract meaning representation for sembanking". In: *Proceedings of the 7th linguistic annotation workshop and interoperability with discourse*. 2013, pp. 178–186.

[6] K. B. Broda, M. Law, and A. Russo. "Iterative Learning of Answer Set Programs with Context Dependent Examples". In: Cambridge University Press (CUP): STM Journals, Jul 25, 2016. ISBN: 1475-3081. DOI: 10.1017/S1471068416000351. URL: http://hdl.handle.net/10044/1/52844.

[7] Joao Carreira and Andrew Zisserman. "Quo Vadis, Action Recognition? A New Model and the Kinetics Dataset". In: IEEE, Jul 2017, pp. 4724–4733. ISBN: 1063-6919. DOI: 10.1109/CVPR.2017.502. URL: https://ieeexplore.ieee.org/document/8099985.

[8] Joao Carreira et al. *A Short Note on the Kinetics-700 Human Action Dataset*. July 2019. URL: https://explore.openaire.eu/search/publication?articleId=od_______18::2a62c0566d4ac6facf800e04a34462f8.

[9] P. Cimiano, A. Hotho, and S. Staab. "Learning Concept Hierarchies from Text Corpora using Formal Concept Analysis". In: *The Journal of artificial intelligence research* 24 (Aug. 2005), pp. 305–339. DOI: 10.1613/jair.1648. URL: https://search.proquest.com/docview/2554129817.

[10] *Clingo ASP solver*. URL: https://potassco.org/clingo/.

[11] Domenico Corapi, Alessandra Russo, and Emil Lupu. *Inductive Logic Programming as Abductive Search*. Jan. 2010. DOI: 10.4230/lipics.iclp.2010.54. URL: https://explore.openaire.eu/search/publication?articleId=datacite____::7c2d6219485a0794937043e4a9de60b9.

[12] Daniel Cunnington et al. "FF-NSL: Feed-Forward Neural-Symbolic Learner". In: *arXiv preprint arXiv:2106.13103* (2021).

[13] Daniel Cunnington et al. “NSL: Hybrid interpretable learning from noisy raw data”. In: *arXiv preprint arXiv:2012.05023* (2020).

[14] Emmanuelle-Anna Dietz, Damir Vandic, and Flavius Frasincar. “TaxoLearn”. In: vol. 1. WI-IAT ’12. IEEE Computer Society, Dec 4, 2012, pp. 58–65. DOI: 10.1109/WI-IAT.2012.129. URL: http://dl.acm.org/citation.cfm?id=2457664.

[15] Arthur Drozdov et al. “Online Symbolic Learning of Policies for Explainable Security”. In: *2021 Third IEEE International Conference on Trust, Privacy and Security in Intelligent Systems and Applications (TPS-ISA)*. IEEE, 2021, pp. 269–278.

[16] R. Evans and R. Craven. *Knowledge Representation*. Notes from Imperial College London Module. 2021.

[17] Jianping Fan et al. “Incorporating Concept Ontology for Hierarchical Video Classification, Annotation, and Visualization”. In: *IEEE transactions on multimedia* 9.5 (Aug. 2007), pp. 939–957. DOI: 10.1109/TMM.2007.900143. URL: https://ieeexplore.ieee.org/document/4276710.

[18] Jianping Fan et al. “Semantic video classification and feature subset selection under context and concept uncertainty”. In: JCDL ’04. New York NY: ACM, Jun 7, 2004, pp. 192–201. DOI: 10.1145/996350.996395. URL: http://dl.acm.org/citation.cfm?id=996395.

[19] Melvin Fitting. “Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. Logic programming, Proceedings of the fifth international conference and symposium, Volume 2, edited by Robert A. Kowalski and Kenneth A. Bowen, Series in logic programming, The MIT Press, Cambridge, Mass., and London, 1988, pp. 1070–1080. - Kit Fine. The justification of negation as failure. Logic, methodology and philosophy of science VIII, Proceedings of the Eighth International Congress of Logic, Method”. In: *The Journal of symbolic logic* 57.1 (Mar. 1992), pp. 274–277. DOI: 10.2307/2275201.

[20] Bernhard Ganter and Rudolf Wille. *Formal Concept Analysis*. Berlin: Springer Berlin / Heidelberg, 2012. ISBN: 9783540627715. URL: https://ebookcentral.proquest.com/lib/[SITE_ID]/detail.action?docID=1263038.

[21] Zijian Gao et al. “CLIP2TV: An Empirical Study on Transformer-based Methods for Video-Text Retrieval”. In: (Nov. 2021). URL: https://arxiv.org/abs/2111.05610.

[22] Amirata Ghorbani et al. “Towards Automatic Concept-based Explanations”. In: (Feb. 2019). URL: https://arxiv.org/abs/1902.03129.

[23] Kaiming He et al. “Deep residual learning for image recognition”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[24] Peter G. Hinman. *Fundamentals of mathematical logic*. CRC Press, 2018.

[25] *ILASP Releases*. URL: https://github.com/ilaspltd/ILASP-releases.

[26] Dan Jurafsky and James H. Martin. *Speech and language processing*. 2. ed., Pearson new internat. ed. Upper Saddle River, NJ [u.a.]: Prentice Hall, Pearson Education International, 2014. ISBN: 9781292025438. URL: http://bvbr.bib-bvb.de:8991/F?func=service&doc_library=BVB01&local_base=BVB01&doc_number=026203617&sequence=000002&line_number=0001&func_code=DB_RECORDS&service_type=MEDIA.

[27] Pang Wei Koh et al. "Concept Bottleneck Models". In: (July 2020). URL: https://arxiv.org/abs/2007.04612.

[28] M. Law and A. Russo. *Logic Based Learning*. Notes from Imperial College London Module. 2021.

[29] Mark Law. *ILASP Meta-level ASP definition*. URL: https://doc.ilasp.com/specification/meta_asp_hypothesis_space.html.

[30] Mark Law, Alessandra Russo, and Krysia Broda. "Inductive Learning of Answer Set Programs". In: Logics in Artificial Intelligence. Cham: Springer International Publishing, Jan. 2014, pp. 311–325. ISBN: 331911557X. DOI: 10.1007/978-3-319-11558-0_22. URL: http://link.springer.com/10.1007/978-3-319-11558-0_22.

[31] Mark Law, Alessandra Russo, and Krysia Broda. "Inductive Learning of Answer Set Programs from Noisy Examples". In: (Aug. 2018). URL: https://arxiv.org/abs/1808.08441.

[32] Mark Law, Alessandra Russo, and Krysia Broda. "Inductive learning of answer set programs from noisy examples". In: *arXiv preprint arXiv:1808.08441* (2018).

[33] Mark Law, Alessandra Russo, and Krysia Broda. "The ILASP system for Inductive Learning of Answer Set Programs". In: (May 2020). URL: https://arxiv.org/abs/2005.00904.

[34] Mark Law et al. "FastLAS: Scalable Inductive Logic Programming Incorporating Domain-Specific Optimisation Criteria". In: *Proceedings of the ... AAAI Conference on Artificial Intelligence* 34.3 (Apr. 2020), pp. 2877–2885. DOI: 10.1609/aaai.v34i03.5678.

[35] Mark Law et al. "Scalable Non-observational Predicate Learning in ASP." In: *IJCAI*. 2021, pp. 1936–1943.

[36] Vladimir Lifschitz. *Answer set programming*. Cham: Springer, 2019. ISBN: 9783030246570.

[37] Vladimir Lifschitz. "What is Answer Set programming?" In: *AAAI 2008*. July July, 2008. URL: http://www.aaai.org/Library/AAAI/2008/aaai08-270.php.

[38] Lucian Vlad Lita et al. "Truecasing". In: *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics*. 2003, pp. 152–159.

[39] Yinhan Liu et al. "Roberta: A robustly optimized bert pretraining approach". In: *arXiv preprint arXiv:1907.11692* (2019).

[40] Amy Luo. *Capitalization Rules in English*. Jan. 2020. URL: https://www.scribbr.com/language-rules/capitalization-rules/.

[41] Andrew L. Maas et al. "Learning Word Vectors for Sentiment Analysis". In: *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*. Portland, Oregon, USA: Association for Computational Linguistics, June 2011, pp. 142–150. URL: https://aclanthology.org/P11-1015.

[42] David J. C. MacKay. *Information theory, inference, and learning algorithms*. Reprinted with corr. Cambridge [ua.]: Cambridge Univ. Press, 2004. ISBN: 9780521642989. URL: http://www.loc.gov/catdir/toc/cam031/2003055133.html.

[43] Feng Mao et al. "Hierarchical Video Frame Sequence Representation with Deep Convolutional Graph Network". In: Computer Vision – ECCV 2018

Workshops. Cham: Springer International Publishing, Jan. 2019, pp. 262–270. ISBN: 9783030110178. DOI: 10.1007/978-3-030-11018-5_24. URL: http://link.springer.com/10.1007/978-3-030-11018-5_24.

[44] Andrei Margeloiu et al. "Do Concept Bottleneck Models Learn as Intended?" In: *arXiv preprint arXiv:2105.04289* (2021).

[45] *MIT License*. URL: https://opensource.org/licenses/MIT.

[46] Khalil Mrini et al. "Rethinking Self-Attention: Towards Interpretability in Neural Parsing". In: (Nov. 2019). URL: https://arxiv.org/abs/1911.03875.

[47] Stephen Muggleton. "Inductive Logic Programming". In: *New generation computing* 8.4 (Feb. 1991), pp. 295–318. DOI: 10.1007/BF03037089.

[48] Stephen H. Muggleton and Christopher H. Bryant. "Theory Completion Using Inverse Entailment". In: Inductive Logic Programming. Berlin, Heidelberg: Springer Berlin Heidelberg, Aug. 2000, pp. 130–146. ISBN: 9783540677956. DOI: 10.1007/3-540-44960-4_8. URL: http://link.springer.com/10.1007/3-540-44960-4_8.

[49] Stephen H. Muggleton, José Carlos Almeida Santos, and Alireza Tamaddoni-Nezhad. "TopLog: ILP Using a Logic Program Declarative Bias". In: vol. 5366. Logic Programming. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 687–692. ISBN: 9783540899815. DOI: 10.1007/978-3-540-89982-2_58. URL: http://link.springer.com/10.1007/978-3-540-89982-2_58.

[50] Daniel Müllner. *Modern hierarchical, agglomerative clustering algorithms*. Sept. 2011. URL: https://explore.openaire.eu/search/publication?articleId=od_______18::b323e546e5152b4b48ac7e1352f1cca7.

[51] Maria-Elena Nilsback and Andrew Zisserman. "Automated flower classification over a large number of classes". In: *2008 Sixth Indian Conference on Computer Vision, Graphics Image Processing*. IEEE, 2008, pp. 722–729.

[52] AJ Piergiovanni and Michael S. Ryoo. "Fine-Grained Activity Recognition in Baseball Videos". In: IEEE, Jun 2018, pp. 1821–18218. DOI: 10.1109/CVPRW.2018.00226. URL: https://ieeexplore.ieee.org/document/8575389.

[53] Luc De Raedt. "Inductive Logic Programming". In: *Inductive Logic Programming*. (2010).

[54] Luc De Raedt and Kristian Kersting. "Probabilistic inductive logic programming". In: Probabilistic inductive logic programming. Springer, 2008, pp. 1–27.

[55] Luc De Raedt and Ingo Thon. "Probabilistic rule learning". In: *International conference on inductive logic programming*. Springer, 2010, pp. 47–58.

[56] Luc De Raedt et al. "Inducing probabilistic relational rules from probabilistic examples". In: *Twenty-fourth international joint conference on artificial intelligence*. 2015.

[57] Colin Raffel et al. "Exploring the limits of transfer learning with a unified text-to-text transformer." In: *J.Mach.Learn.Res.* 21.140 (2020), pp. 1–67.

[58] Oliver Ray, Krysia Broda, and Alessandra Russo. "Hybrid Abductive Inductive Learning: A Generalisation of Progol". In: Inductive Logic Programming. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 311–328. ISBN: 3540201440. DOI: 10.1007/978-3-540-39917-9_21. URL: http://link.springer.com/10.1007/978-3-540-39917-9_21.

[59] Rodrigo de Salvo Braz et al. "An inference model for semantic entailment in natural language". In: *Machine Learning Challenges Workshop*. Springer, 2005, pp. 261–286.
[60] *Spacy*. URL: https://spacy.io/.
[61] *Spacy Benchmarks*. URL: https://spacy.io/usage/facts-figures#benchmarks.
[62] *Spacy POS Tagging*. URL: https://spacy.io/usage/linguistic-features#pos-tagging.
[63] *Subject vs Predicate*. URL: https://www.thesaurus.com/e/grammar/subject-vs-predicate/.
[64] Christian Szegedy et al. "Rethinking the Inception Architecture for Computer Vision". In: IEEE, Jun 2016, pp. 2818–2826. DOI: 10.1109/CVPR.2016.308. URL: https://ieeexplore.ieee.org/document/7780677.
[65] David Tuckey, Krysia Broda, and Alessandra Russo. "Towards Structure Learning under the Credal Semantics." In: *ICLP Workshops*. 2020.
[66] Catherine Wah et al. "The caltech-ucsd birds-200-2011 dataset". In: (2011).
[67] Catherine Wah et al. "The caltech-ucsd birds-200-2011 dataset". In: (2011).
[68] Jun Xu et al. "MSR-VTT: A Large Video Description Dataset for Bridging Video and Language". In: IEEE, Jun 2016, pp. 5288–5296. DOI: 10.1109/CVPR.2016.571. URL: https://ieeexplore.ieee.org/document/7780940.
[69] Shen Yan et al. "Multiview Transformers for Video Recognition". In: (Jan. 2022). URL: https://arxiv.org/abs/2201.04288.
[70] Chih-Kuan Yeh et al. "On Completeness-aware Concept-Based Explanations in Deep Neural Networks". In: (Oct. 2019). URL: https://arxiv.org/abs/1910.07969.
[71] Jiawei Zhou et al. "Structure-aware Fine-tuning of Sequence-to-sequence Transformers for Transition-based AMR Parsing". In: *arXiv preprint arXiv:2110.15534* (2021).