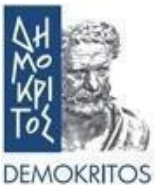


Comparison of NeuroSymbolic Programming frameworks in Human Activity Recognition

by

Iraklis Evangelinos

Submitted

in partial fulfillment of the requirements for the degree of

Master of Artificial Intelligence

at the

UNIVERSITY OF PIRAEUS

June 2024

Author . . . . . .

Iraklis Evangelinos

II-MSc “Artificial Intelligence”

June, 2024

Certified by. . . . . .

Nikolaos Kantzouris

NCSR - Demokritos

Thesis Supervisor

Certified by. . . . . .

George Paliouras

NCSR - Demokritos

Member of Examination Committee

Certified by. . . . . .

Alexandros Artikis

University of Piraeus

Member of Examination Committee

Comparison of NeuroSymbolic Programming frameworks in Human Activity Recognition

By

Iraklis Evangelinos

Submitted to the II-MSc “Artificial Intelligence” on April, 2024, in
partial fulfillment of the
requirements for the MSc degree

Abstract

Human activity recognition (HAR) is a fundamental task in artificial intelligence, with applications in healthcare, smart homes, and surveillance. Traditional deep learning approaches have achieved state-of-the-art performance in HAR tasks, but they often lack interpretability and struggle to capture complex semantic relationships between activities. Neurosymbolic AI, which combines the strengths of symbolic reasoning and deep learning, offers a promising alternative.

This thesis explores the application of NeuroSymbolic AI to HAR using DeepProbLog, a popular neurosymbolic framework that integrates logic-based probabilistic modeling with deep learning. We focus on DeepProbLog as it is more extensively used in the field and has shown great promise in various applications. Our primary goal is to investigate how well DeepProbLog performs in recognizing human activities and compare its results with traditional deep learning approaches.

We first introduce the DeepProbLog framework and discuss its underlying principles, including the integration of logic-based probabilistic modeling with neural networks. We then present a comprehensive evaluation of DeepProbLog's performance in HAR tasks, highlighting its strengths and limitations. Our experiments demonstrate that DeepProbLog outperforms traditional deep

learning methods in terms of accuracy and interpretability, particularly when dealing with complex activity relationships.

Thesis Supervisor: Nikolaos Kantzouris

Title: Research Associate NCSR - Demokritos

Acknowledgments

I would like to express my gratitude to my Supervisor Professor Dr. Nikos Kantzouris for his guidance and patience through the duration of the project and the writing of this thesis. Additionally, I extend my sincere appreciation to Vassilis Magginas and Nikos Magginas, for their valuable assistance and their willingness to share their expertise whenever I needed it. I would also like to thank my family for supporting me throughout my studies and in the pursuit of my goals.

Any opinions, findings, conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of the University of Piraeus and Institute of Informatics and Telecommunication of NCSR “Demokritos”.

Contents

1. Introduction

The subject of this thesis is to apply Neuro-Symbolic Artificial Intelligence frameworks on the complex task of Human Activity Recognition. The technology behind these frameworks touches upon different fields of artificial intelligence, such as Deep Learning and Logic Programming.

1.1 Complex event recognition

Complex event recognition systems(CER)[9], are systems that enable the task of identifying specific patterns, called complex events in temporal data, consisting of simple events which are detected as an event unfolds. The stream of time-stamped simple events is processed in real time and the system observes for the occurrence of specific patterns that trigger a complex event.

There have been implementations of logic-based CER systems that combine reasoning with machine learning[3], using formalisms such as the Event Calculus[4] which allows for reasoning on events while new fluents are introduced by the time-stamped data stream and old fluents are terminated.

Event Calculus

In Event Calculus a predicate is used to determine whether or not a fact holds at a specific timepoint, in addition predicates can denote when an event is initiated and when it is terminated. One of the most important aspects of this formalization is the axiom of inertia, which proposes that an event is true at a timepoint t if it is initiated at that timepoint or if it has been initiated at a previous timepoint and it has not terminated up to timepoint t.

The axiom of inertia is supplemented by domain-specific rules about when events are initiated and when they are terminated, in order to model a specific problem. The Prolog programming language[5] and its derivatives has been used in the past to model Event Calculus interpreters along with domain specific rules.

An example of a complex event is the recognition of specific human activity from video inputs, which is the example this thesis focuses on, where patterns consisting of simple events such as a person standing close to another person or a person moving at a slow speed can lead a complex event recognition system to conclude that two people are meeting, fighting etc. Complex event recognition systems can offer significant insight into the data provided, by analyzing a stream of so called simple events and arriving at conclusion on when a complex event takes place. [1]

1.2 Neuro-symbolic AI

Machine learning and machine reasoning have been largely addressed separately and in isolation by different communities in Artificial Intelligence. Learning traditionally refers to data-driven, subsymbolic techniques for generating predictive models and it is frequently related to low-level (e.g. perception-level) tasks, especially within deep learning. Reasoning, on the other hand, refers mostly to symbolic, frequently logic- based techniques for deriving new knowledge from data and existing knowledge, and it is typically associated with higher-level inference tasks. Artificial Intelligence really needs both learning and reasoning in order to bring to life systems that combine the best of two worlds, i.e. systems capable of perceiving their environment and making sense of data, while also reasoning with what has been learnt and consulting existing knowledge about the world.

Neuro-symbolic AI is a novel field of Artificial Intelligence which aims to “integrate the ability to learn from experience and the ability to reason from what has been learned” [2]. Neuro-symbolic AI combines deep learning methods such as LSTM or CNN neural networks and their ability to provide low level perception through vast amounts of raw data with logic programming methods that excel in high-level reasoning from structured data. This has the potential of addressing many of the shortcomings of contemporary AI approaches, including the black-box nature and the brittleness of deep learning, and the difficulty to adapt knowledge representation models in the light of new data. For instance, Neuro-symbolic approaches allow to enforce symbolic constraints on neural

networks, thus allowing for a more robust and controlled behavior, while training with fewer data and allowing, in principle, to trace/explain individual predictions.

NeuroSymbolic models consist of two usually distinct components that interface with one another to create a model that combines the two modes of learning/reasoning.

The neural component is used to learn complex patterns from large amounts of unstructured data, which is unusable to symbolic reasoning methods. The network learns to transform the raw input into data that is usable by the symbolic component. The symbolic component does problem-solving through symbolic representation of rules and data in the form of facts. Structured data, if available, along with the output of the neural component are used to do inference on the input of the neurosymbolic model. The ability of the symbolic component to insert prior or expert knowledge on the domain of the problem at hand is not possible through the use of traditional Deep Learning methods.

The way the integration of the two components is done varies between approaches, from integrating the logic into the neural network[6][7][8] to training and tuning each model separately and simply piping the output of the network to the symbolic component. By integrating the two components, one can achieve the best of both worlds, using the strengths of each approach. This is especially useful since NeuroSymbolic approaches allow for seamless processing of unstructured data through the neural network and use of logic rules, background or expert knowledge through the symbolic component. This leads to more explainable and interpretable results, which are an open question when using pure Deep Learning methods[10].

This rather new academic field is being studied actively and many implementations and frameworks have been introduced, all integrating various deep learning systems with logic programming and reasoning systems[20][21].

1.3 The motivation of this thesis and related work

Many attempts at integrating neural and logic systems have been proposed and many more are in active research/development. This thesis aims to compare two of the most recent and promising frameworks in the task of detecting and

recognising human interaction in videos. For this end the CAVIAR dataset[11] is used. This thesis compares the work done on[23], where the NeurASP framework[12] is used, which combines Neural Networks and Answer Set Programming methods on the output of the NN, with a similar approach on DeepProbLog[13], which applies the ProbLog reasoning system[14][28] to the outputs of the Neural Network. Through this comparison the thesis aims to prove that Neuro-symbolic learning frameworks are becoming increasingly capable of completing tasks that were previously only completed by purely Deep Learning methods.

The main body of the thesis focuses on implementing a specific Human Activity Recognition dataset and task on DeepProbLog and comparing it to other methods. The dataset as well as the task were created in the context of [23] and experiments on a NeurASP implementation were performed. In the experimental section these results are used as they are directly comparable. This thesis is based on the task presented in [23] with the aim of broadening the number of NeSy frameworks tested in real world applications. Future work can also extend the scope of this thesis in a similar direction(e.g using DeepStochLog[15]).

1.4 Thesis structure

The rest of this thesis is structured as follows:

In section 2 a brief but comprehensive, with respect to the scope of this thesis, overview is provided on Deep Learning and Neuro-Symbolic concepts. This includes Deep Learning fundamentals as well as CNN, RNN and especially the LSTM architecture. In a separate subsection the Prolog programming language is introduced, along with basic symbolic ai and logic programming concepts.

In section 3 the task of Human Activity Recognition(HAR) is presented. Finally the CAVIAR dataset is presented along with modifications made to it for the purposes of this thesis.

In section 4 the main framework tested in the experimental section, the DeepProbLog neural probabilistic programming language is explained, the concept of neural predicates, how the inference and learning mechanisms function, along with a subsection on gradient semirings and aProbLog.

In section 5 emphasis is put on challenges and special details that need to be taken into account in order to perform HAR with DeepProbLog are explained, along with an overview of the modifications made to the DPL source code in order to perform the necessary tasks.

In section 6 the implementations that are put through testing are presented, the experimental results and the conclusions along with thoughts on future work based on this thesis can be found in **sections 7 and 8.**

In the Appendix various technical aspects of the thesis can be found, along with sections of code that is referenced in passing in the main body of the thesis

2. Deep learning and Neuro - Symbolic concepts

In this section, we define some basic concepts and terms that will be used. The first part of this section delves into some basic Deep Learning concepts, the CNN and LSTM architectures and closes with an evaluation of the advantages and drawbacks of Deep Learning methods on Human Activity Recognition. The second part delves into basic Symbolic AI concepts and the drawbacks of purely Symbolic methods and closes with some basic neuro-symbolic concepts that are useful for the reader to follow the rest of the thesis.

2.1 Brief overview of Deep Learning methods and concepts

2.1.1 Basic Deep Learning concepts

Deep learning:

Deep learning is a subset of Machine Learning that makes use of Neural Networks that are inspired from the neurons of the human brain, to identify and learn patterns in vast amounts of data. The training of the Neural Network allows for a model that has “learned” complex tasks without having been explicitly programmed to perform that task. Good examples of tasks that neural networks excel at include image classification, video activity recognition, creating Large Language Models(LLMs) that can answer questions and converse with humans.

Perceptron:

A perceptron can be thought of as the simplest cell that can be found in neural networks. It receives input in the form of a tensor, which is then processed internally by the perceptron through an activation function, which determines the output of the model. A key characteristic is that the activation function is nonlinear, allowing for layers of perceptrons in a neural network to approximate to

some degree complex nonlinear functions. In this sense a function can be thought of as any arbitrary mapping of an input to an output.

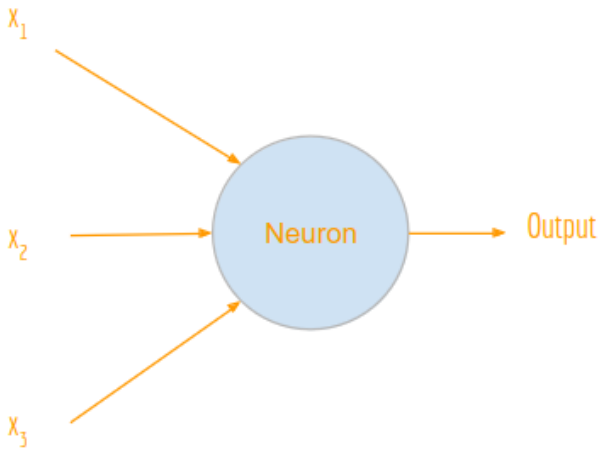


Figure 1: A simple perceptron

Neural network:

A neural network consists of layers of neurons or cells, which in their simplest form are essentially perceptrons. The neural network has three types of layers, with the first and last layer being called input and output layers respectively, and any in-between layers being called hidden layers, since the interaction between those layers is “hidden” from the user, who can only observe the input and the output of the model.

The shape of the input layer is determined by the nature of the data in the dataset. In a similar vein the shape of the output layer is determined by the nature of the label(continuous output, discrete categories etc) and the number of labels possible in the given dataset.

In contrast the shape and number of hidden layers can vary drastically from application to application, as well as between different implementations on the same dataset. In addition to fully-connected layers of neurons, the hidden layers can, and usually consist of layers that perform data transformations, such as convolutional layers in CNNs[16], the softmax transformation on the output layer among others.

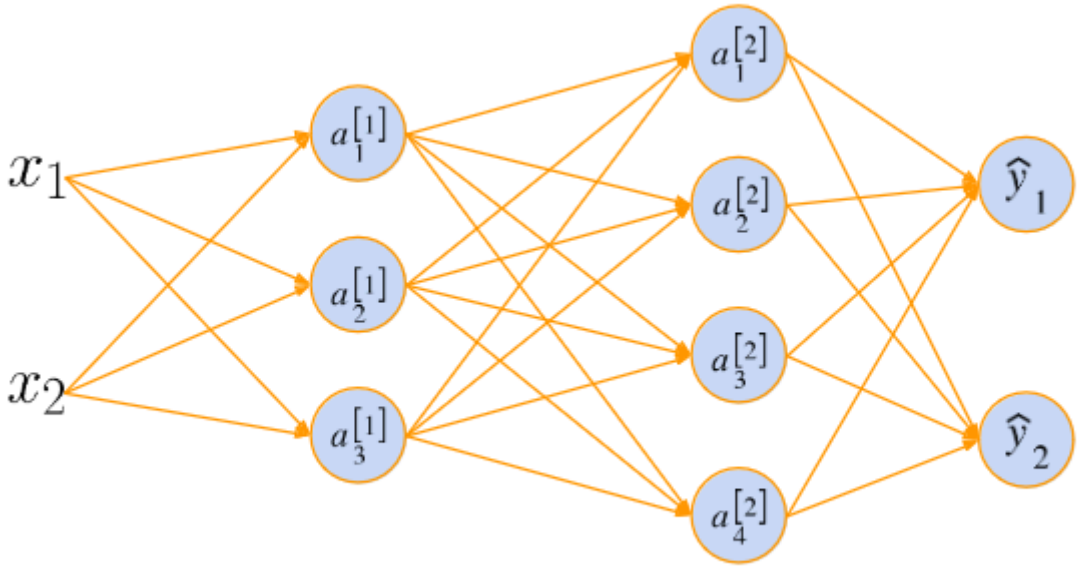


Figure 2: A feed-forward Neural Network with two layers.

Dataset:

The dataset used in training a Neural Network is an often large collection of annotated data used to provide a large and comprehensive -for the intents and scope of the model trained- set of pairs of raw data(e.g videos, images, text) and a label that represents a specific attribute or feature that characterizes the data provided in the specific example. Datasets need to be comprehensive enough, while also containing many instances of the object of the learning process, that the network needs to learn in order for it to contain enough information and variance in the individual datapoints, which ensure that an adequate learning signal is given to the model during training.

Training:

Through the use of the structured data provided in a dataset, a Neural Network can update its parameters in a process called “training”, during which the network is given the raw data and does a “forward-pass” through each of its layers, and then the output is compared to the label, so that the parameters of the model can be adjusted to provide better results. The course and length of the training process is influenced by design choices made by the model’s creator, called hyperparameters.

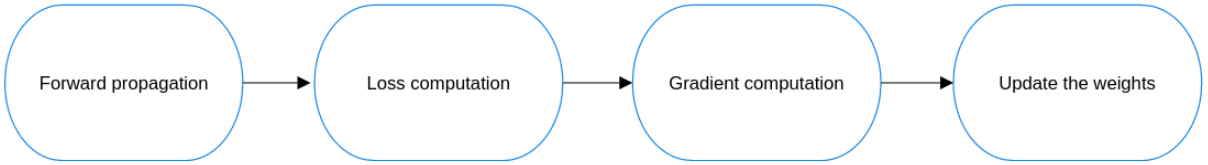


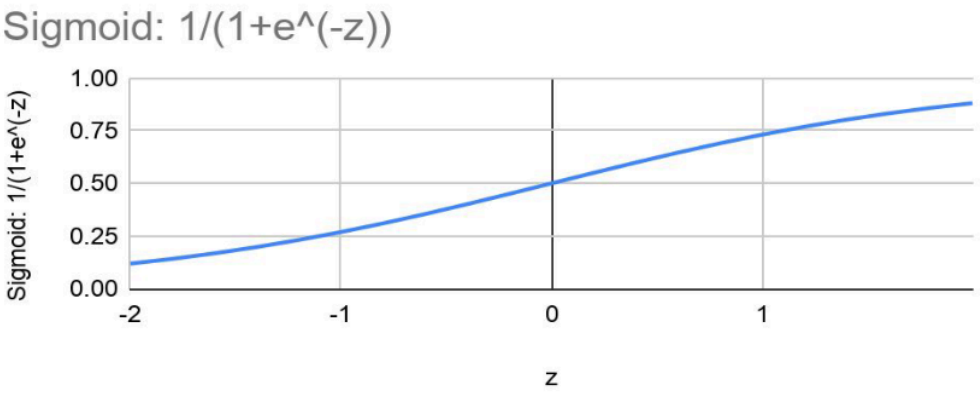
Figure 3: An overview of the main steps of the training process

Hyperparameter tuning:

In addition to the parameters that are "learned" during training, a model consists of parameters that affect the training process, such as the learning rate, the batch size, the number of epochs among others. These parameters are called hyper-parameters, since they cannot be altered during the training process. The process of testing and comparing many different hyper-parameters, either manually or in an automated fashion is called hyperparameter tuning and is an integral part of the creation of a model.

Activation functions:

Each cell of a Neural Network needs to determine whether or not to "activate" i.e, pass values to the cells that it is connected to in the following layer, based on the input of the previous cells that are themselves connected to the neuron in question. The mathematical function that takes the inputs from the neurons of the previous layers and outputs what the neuron will "pass onto" the next layer is called an **activation function**. One crucial characteristic that is common in all activation functions is the fact that they are **non-linear**, since it can be mathematically proven that a Neural Network with linear activation functions can only approximate linear functions irrespective of its complexity/depth.



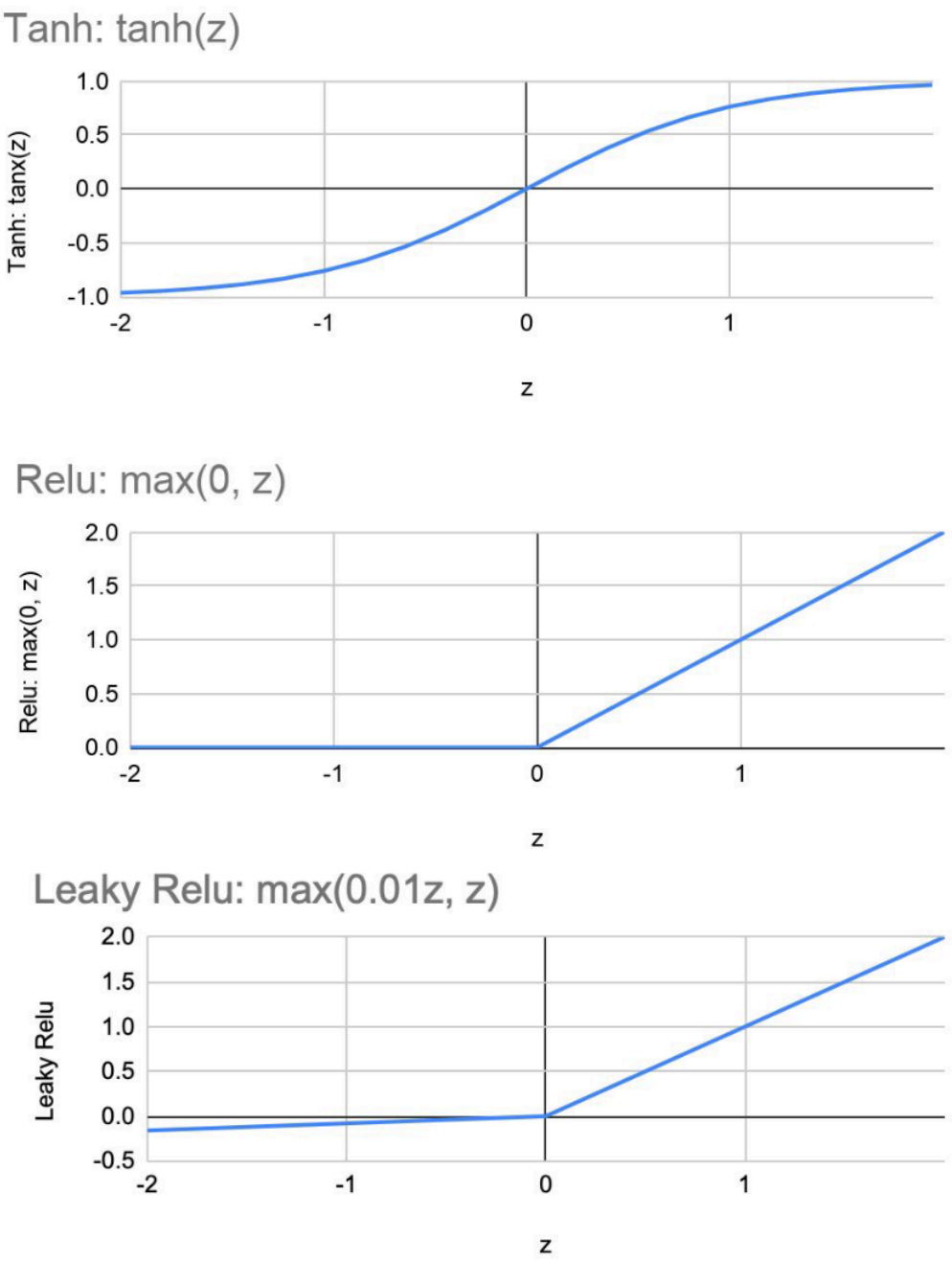


Figure 4: The most commonly used activation functions

Loss functions:

Loss functions are a type of function used during the training process of a model. A loss function is a mathematical expression, used to measure how well the model performs on a given task, providing a way to quantify the difference between predicted and actual outputs. This is in turn used to help the optimization algorithm make the needed changes to the network's parameters in order for it to

improve in the next iteration. This is done by trying to minimize the loss functions output, thus minimizing the discrepancy between predicted and actual results.

There are several types of loss functions used in deep learning, depending on the nature of the problem the model is trying to solve:

- Cross-entropy loss is commonly employed in classification problems where the model predicts a probability distribution for each class, particularly when dealing with categorical outputs.
- Mean Squared Error (MSE) is often utilized in regression tasks that involve predicting continuous values. MSE measures the average squared difference between predicted and actual output, providing us with a numerical value that can be used to guide the model's training.
- Mean Absolute Error (MAE), on the other hand, calculates the differences' absolute values instead of squaring them, making it less sensitive to outliers in the data.

Backpropagation:

A crucial component of the learning process is backpropagation, which is an optimization algorithm that allows for adjustments to the model's parameters effectively. Backpropagation is an iterative process, which works by calculating the gradient of the loss function with respect to each weight in the network and updating these weights in the opposite direction of the gradient.

The algorithm consists of two main steps: forward propagation and reverse propagation (backpropagation). Forward propagation involves feeding the input data through the neural network, calculating the output, and comparing it to the target value. The difference between the predicted and actual outputs is then used to calculate the error at each layer of the neural network.

In the second step, reverse propagation or backpropagation, the gradient of the loss function with respect to each weight in the network is computed using the chain rule. This process allows the algorithm to determine how much each weight should be adjusted to minimize the error further. Once the gradients are calculated, they're used to update the weights in the opposite direction of the gradient.

Gradient descent, SGD and the Adam optimizer

Gradient descent is an optimization algorithm that is used to find the minimum value of a function by iteratively updating its variables in the direction of steepest descent based on the calculated gradients. The gradient descent algorithm consists of two main steps, calculating gradients and updating weights.

The first step involves computing the partial derivatives of the loss function with respect to each weight in the network using the chain rule. Once the gradients are calculated, they're used to update the weights by a learning rate multiplied by the gradient. The learning rate determines the size of optimization steps taken during the process and is a hyperparameter that can be tuned for better performance.

Stochastic Gradient Descent (SGD), a variant of gradient descent that updates the model's weights after processing each individual training example rather than in batches. This approach offers several advantages, such as faster convergence and reduced memory requirements compared to its batch counterpart. In contrast to traditional Gradient descent, SGD calculates the gradients for each training sample independently before updating the weights using the calculated gradient. This results in more frequent weight updates but may result in a less stable optimization process due to noise introduced by individual samples.

The Adam optimizer is a variant of gradient descent that combines the advantages of two other optimizers - AdaGrad and RMSProp. It uses a combination of the first-order moment vector and the second-order moment matrix to adaptively adjust the learning rate for each weight in the network, resulting in faster convergence and better performance. The first-order moment vector captures the direction of steepest ascent, while the second-order moment matrix reflects the magnitude of gradients. By adaptively adjusting the learning rate based on these values, the Adam optimizer can provide better optimization performance than its counterparts.

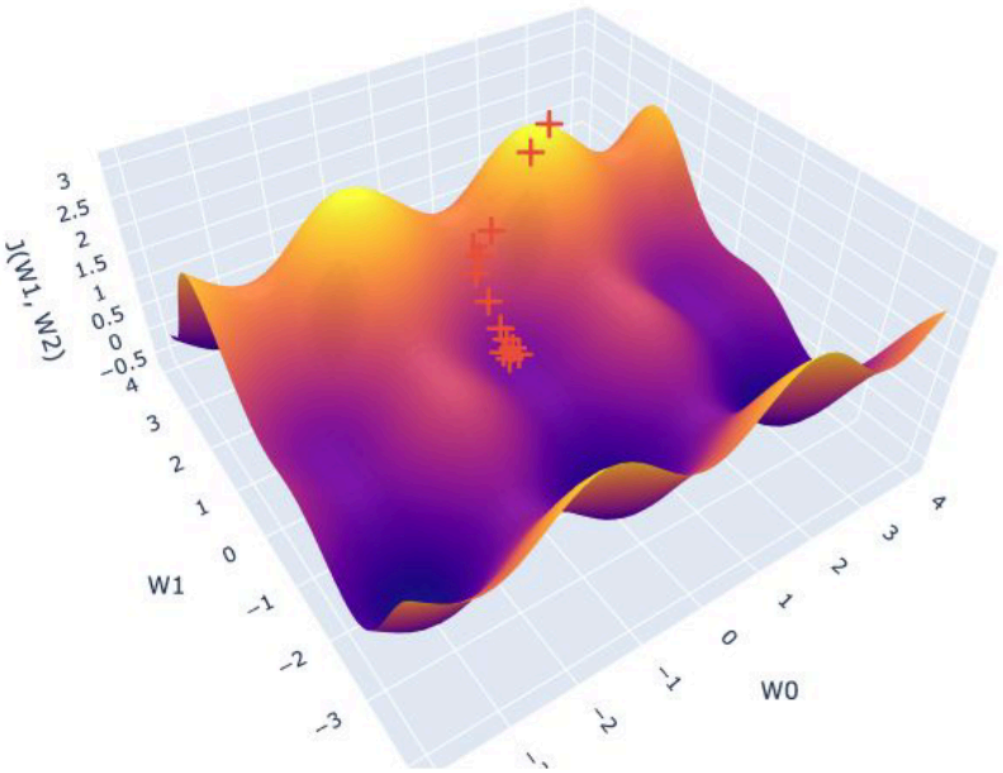


Figure 5: A gradient descent example, where each cross is a step towards minimizing the loss function

2.1.2 Deep learning architectures

There are many Neural Network architectures and models, each with its own modifications to the “vanilla” version that contains fully connected layers that consist of simple perceptrons as cells. We focus mainly on a family of neural networks called Recurrent Neural Networks and specifically the Long - Short Term Memory (LSTM) architecture, which is widely used in tasks involving sequences[17], as well as the CNN architecture which is commonly used in tasks involving images[29].

2.1.2.1 The LSTM architecture

The fundamental building block of the LSTM architecture is the LSTM cell, which consists of four “gates” that control the flow and retention of information. The three gates are:

- The forget gate, which concatenates the input with the previous state that is internally stored in the cell. It is usually implemented using the sigmoid

function, and is responsible for deciding which information is not important, thus “forgotten”.

- The input gate, which is responsible for determining which parts of the new information are important for the new “record” that is kept in the cell. It is implemented in a fashion similar to the forget gate, while using the tanh function to determine the importance of the new information.
- The output gate, which is responsible for determining which part of the current internally stored record of the input, is to be revealed in the form of the output of the cell. It uses a sigmoid function to determine what parts make it to the output and then using tanh it maps the cell state to the (-1,1) range, creating the output by multiplying pointwise the two vectors.

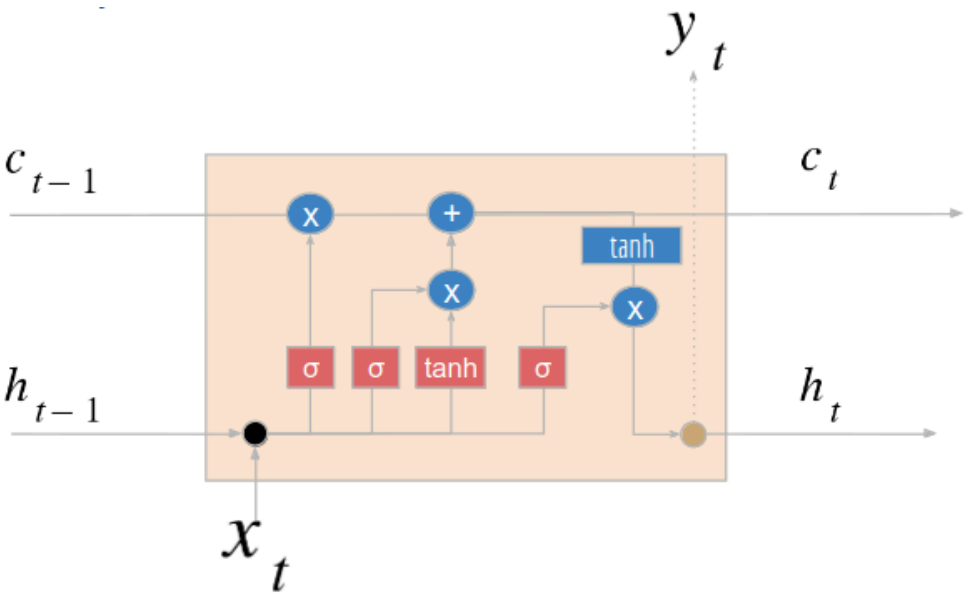


Figure 6: An LSTM cell as described in the section above

The LSTM cells can be configured in multiple different ways, from the most simple linear linking of a cell to the next one, to more complex architectures such as multi-layer LSTM networks, where each layer consists of LSTM cells and Bi-directional LSTMs, where the input is processes in both directions(front to back and vice versa).

By connecting the cells into layers and the multiple layers together the model can process sequences by passing each instance of the sequence into the LSTM, which utilizes the way the cells are connected together, along with the

specific features of each individual cell in order to retain information that is important throughout the sequence.

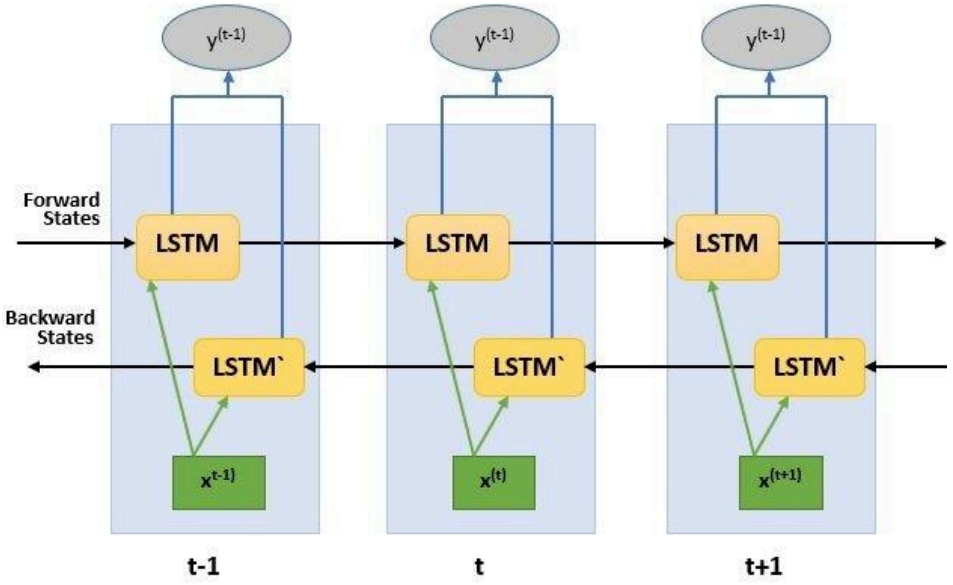


Figure 7: A bi-directional LSTM model(Bi-LSTM)

Advantages of LSTMs compared to other RNN architectures

The LSTM architecture solves many problems plaguing simpler RNN architectures, creating models that perform better, especially in longer sequences that have long term dependencies, where the gated cells mitigate the vanishing gradient effect[18].

2.1.2.2 The CNN architecture

Another very important architecture is the Convolutional Neural Network(CNN), which is used in deep learning applications involving images, such as image classification and object detection tasks.

The input of the CNN architecture is a “flattened” image of a specific size in the form of a tensor, containing all three channels(RGB). It is apparent that the nature of the input means that the CNN has an enormous amount of parameters in the input layer, meaning that a fully connected NN with even a few layers depth can become unmanageable. CNNs have one or more layers that are characteristic to the architecture itself and solve this problem by filtering the input

and extracting only relevant information for the next layers to process. These layers are called Convolutional layers.

In these layers, a mask of a given size(e.g 5x5) makes a pass on the given image and the result of the multiplication of the image tensor and the mask is the output of the layer. The parameters of the mask are learned during training and can have various effects, such as removing noise or discovering important features of an image(e.g edge detection).

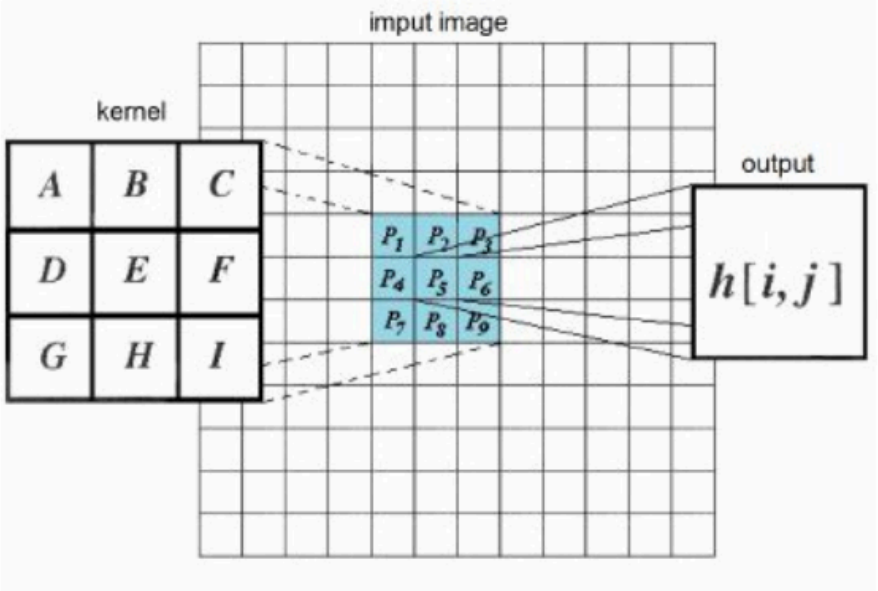


Figure 8: A mask over an input image

Due to the fact that convolutional layers shrink the input size, techniques such as padding are used to lessen the shrinking from layer to layer. During the process of training, the masks of the convolutional layer learn to distinguish features that are important to perform the task, making them do edge detection, detecting geometric shapes among other things. The result is that convolutional layers are trained to do a form of feature extraction that is relevant to the task at hand.

After passing through the convolutional layers, the output is flattened and passed onto one or more fully connected layers, with the final layer being for example a softmax. In essence the latter part of the network's architecture is identical to a traditional fully connected neural network.

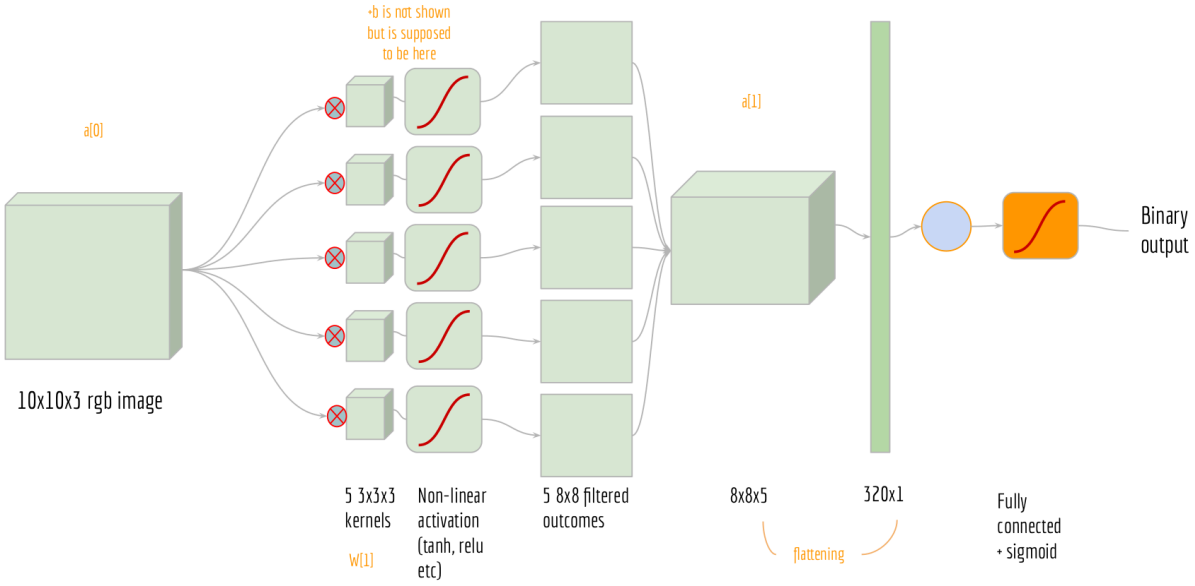


Figure 9: An example Convolutional Neural Network(CNN)

Convolutional layers are often used along with max pooling layers. The Max Pooling layer is a downsampling technique to reduce the spatial dimensions of input data while retaining important information. It does this by taking the maximum value within non-overlapping sub-regions of the input, also known as patches. This process reduces the computational requirements and makes the model more robust to small variations in input data.

2.1.3 Advantages and drawbacks of pure Deep Learning approaches

In the task of Human Activity Recognition, which involves the classification of a sequence of frames into a category explaining an event happening in those frames, the most common approach is to use “pure” Deep Learning methods, such as implementing an LSTM model to classify the clips. This approach offers some advantages, such as the fact that there are many well established and widely used libraries that make such an implementation trivial, such as PyTorch and TensorFlow, with the developer having to only choose some parameters such as the number of cells, whether the LSTM is bidirectional or not etc. In addition to that, Deep Learning models, such as RNNs and other similar architectures can be trained by a sufficiently large and diverse dataset, as a result the patterns that

match each clip to the correct classification are “discovered” by the model itself during training, without no human input that provides prior knowledge necessary, apart from the labeling of the dataset itself. This feature of Deep Learning models is especially useful in cases where the patterns that the model needs to detect are impossible to describe in a machine readable way. An example from the field of machine vision would be classifying images of dogs and cats, a task which is trivial for a human, but cannot be easily described in a way that a model could use that information to do the classification autonomously.

Much research and academic discussion has been focused on identifying and circumventing some of the drawbacks of “pure” Deep Learning models. For the purposes of this thesis, we focus on two main weaknesses of this approach, which, although very useful in many applications, cannot be thought of as a “panacea”. Specifically, Deep Learning models require a very large amount of annotated data in order to train effectively, it is not uncommon for a dataset to involve hundreds of thousands of annotated datapoints, while machine vision datasets can be hundreds of gigabytes in size(such as ImageNet, which is 150Gb in size)[30]. It is therefore apparent that there are instances where this inherent need for large amounts of data can become an obstacle, such as when few datapoints can be sourced for a specific event, for various reasons, such as the event being a rare occurrence. Another drawback is the fact that annotation can sometimes be expensive, especially when human annotators are experts in their field and need to be compensated accordingly for doing the annotations.

One of the most important drawbacks in pure Deep Learning models is their “black box” nature, which means that the output of a model, even when this model achieves outstanding results in a specific task, cannot be explained and therefore can exist a shroud of mistrust, between the human recipient of the output and the model itself, since the patterns that it recognised in order to produce the specific output cannot be communicated in a form humans can understand. This is especially true for cases where the real world implication of a misclassification is that human lives are affected, such as in medical applications, where Deep Learning models have proven to be very capable tools for tasks such as melanoma classification[19], but what lead to the “diagnosis” is not available for the medical professional to take into account.

The final drawback which we will highlight in this thesis is the fact that no prior knowledge can be given to the model from a human expert, in order to make the training process less data hungry, while making sure that some rules are adhered to in the form of constraints, while the model is doing the classification. This last part is very useful since a large amount of data can be replaced with a smaller amount, accompanied with background knowledge, while also building trust in the model, knowing that some domain specific or common sense rules are adhered to.

2.2 Neuro - Symbolic artificial intelligence

Neuro - Symbolic AI is an approach that attempts to combine traditional Neural techniques(i.e Deep Learning models) with symbolic AI concepts such as logic programming and knowledge databases, in order to rectify many of the drawbacks pure neural approaches have, as is highlighted in the section above.

2.2.1 Basic Symbolic Artificial intelligence concepts

Symbolic Artificial Intelligence involves the use of logic and rules in order to create models that can manipulate information using the knowledge they are already given or gained in some cases[12].

Terms

Terms are structures composed of either constants, variables or expressions involving functors.

Atoms

Atoms are expressions involving a predicate p of arity n and n terms t_1. For example an atom involving the predicate *addition* could look like this: *addition(5,3,8)*, with 5, 3 and 8 being constants.

Literals

A literal is either an atom *q(t1,....,tn)* or its negation ¬*q(t1,....,tn)*, they are used to express propositions or conditions.

Rules

A rule is an expression in the form *h :- b_1,b_2,....,b_n,* where *h* is an atom and b_i*i* are literals. The head atom *h* is true if all the body literals b_i are true. Note that the comma(,) separating the literals is equivalent to conjunction(∧).

For example: addition(X,Y,Z) :- number(X), number(Y), X+Y=Z.

Facts

Rules with an empty body are called facts, since the body is empty, thus always true, which ensures that the fact is always evaluated as true.

Ground expressions

An expression that does not contain any variables is called ground. A non ground expression can be transformed into a ground expression through a substitution $\theta = \{V_1 = t_1, .\}$.

Queries

Means of retrieving information about whether a certain relation holds between objects within the logic program. This is done in the form of a question to the system, which in turn seeks a solution based on available facts and rules.

2.2.2 Problog

Prolog is a logic programming language[21]. Its name stands for “Programming in logic” and it is a representative example of declarative and logic programming languages. It is widely used in logic programming applications and has been the basis of other programming languages such as ProbLog[14].

One of the main defining characteristics of Prolog is the fact that, being a declarative language, the programmer specifies what needs to be achieved along with facts and rules that need to be taken into account, without explicitly providing the how to achieve that goal. Rules and facts are baked into the logic program and the user can then query the value of a specific term, which triggers Prolog’s inference mechanism in an attempt to evaluate the query and return a result.

Inference and execution mechanisms in Prolog

In Prolog a logic program containing facts and rules is provided for the user to query and obtain information based on the knowledge and rules inside the logic program. The purpose of the query is to get an answer on whether a term or clause is true or can be inferred to be true from the facts and rules inside the logic

program. It is important to note that Prolog works based on the closed world assumption, which means that if there is not enough evidence to prove that a relationship or a term holds, it is evaluated as false.

There are three primary component to Prolog's inference mechanism, namely:

1. **Resolution**: is a systematic process that combines two clauses (logical statements) to produce a new clause. The resolution mechanism is used to prove or disprove queries.
2. **Unification:** is the process of finding a substitution instance for variables in a query, which allows the engine to match the query with existing facts.
3. **Backtracking**: is the process of exploring different possible solutions to a query by recursively trying out different alternatives until a solution is found or it's determined that no solution exists.

Prolog's execution mechanism uses the concepts above, implementing an Selective Linear resolution for Definite clauses(SLD) with a few additional restrictions. SLD resolution is a theorem proving procedure that is complete for Horn clauses, based on the following principles:

- In linear resolution, one of two clauses is always the result of the previous step. The idea of SLD resolution is to simplify the query step by step.
- The current goal is the negation of the query and ends with the empty clause.
- Each step selects an atom from the goal and a head from a rule in the program and makes them equal with a unifier.
- The selected atom in the goal is replaced with the body of the rule.
- The literals to resolve are selected from left to right.
- The rules are selected from top to bottom.

Problog

Problog is a logic programming language that is based on Prolog, extending it with probabilistic reasoning. This is done through the introduction of probabilistic facts. This means that predicates are not evaluated as true or false, but on a spectrum based on the probability the system assigns to the predicate being true. This type of reasoning can be very helpful in dealing with noisy data,

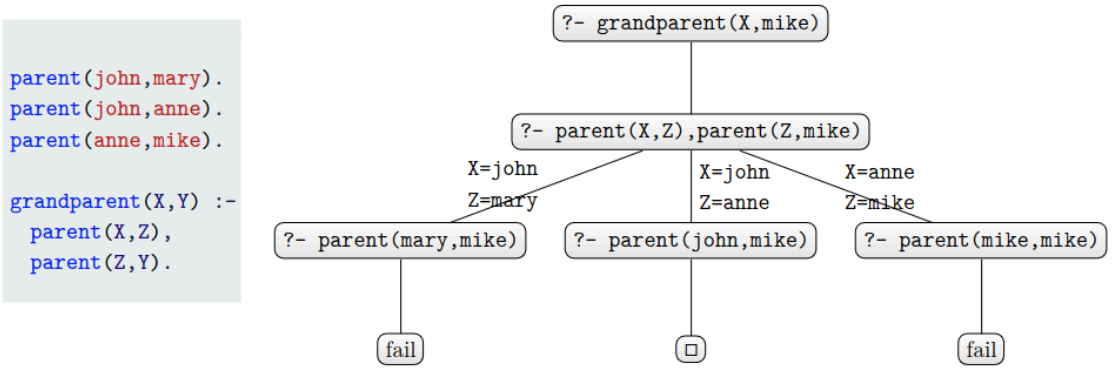


Figure 10: Prolog's execution mechanism tree on an example program

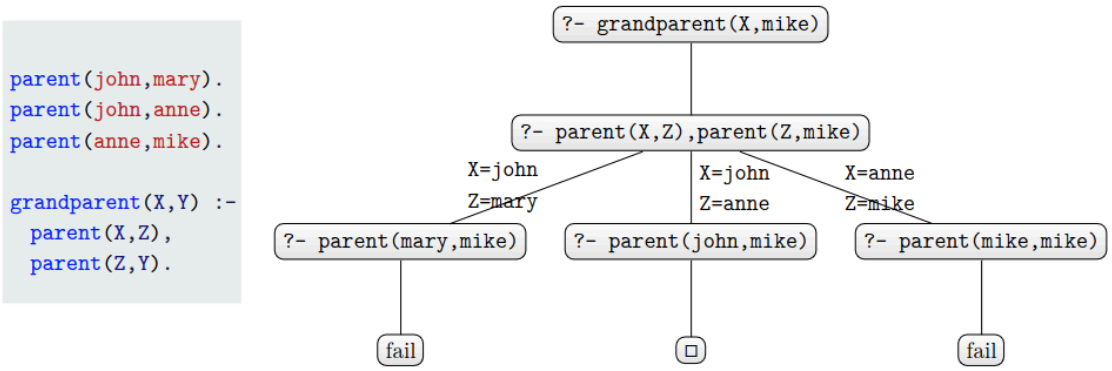


Figure 11: Prolog's execution mechanism tree on recursive rules

where each input cannot be thought of as completely accurate, as well as modeling inference under uncertainty about the truth value of some facts.

2.2.3 Drawbacks and limitations

One of the main drawbacks of symbolic Artificial Intelligence is the fact that a high-level representation of the data is needed in order for it to be manipulated through the reasoning component. This is not always possible, since an image with millions of pixels for example cannot be easily converted into high level representations of important aspects of the image(e.g the coordinates of a person in the frame) without the involvement of a human annotator, limiting the real world application of such approaches. Another drawback is the fact that knowledge needs to be explicitly inserted into the knowledge base, for example rules need to be "coded in" by an expert in order for a model to function. There has been

research involving frameworks that can generate the rules using examples and counterexamples[12], but their current state and performance of state of the art implementations are subject to ongoing research and are beyond the scope of this thesis.

Finally, another limitation that needs to be taken into consideration is that not all approaches factor in uncertainty and a mechanism to handle mislabeled or noisy data, since the output of many such frameworks, such as Prolog[22] is a discrete output, as opposed to the output of a Neural Network which can easily be modeled in such a way as to represent the degree of certainty that a classification is correct.

3. Neuro - Symbolic AI on Human Activity Recognition(HAR)

Neuro-symbolic AI in the form that this thesis explores is a concept that has many novel implementations and frameworks in the last years, all with different approaches as to how the symbolic and neural components interface, with some approaches doing away with the two distinct components, fusing them into one. We will briefly introduce some of the more characteristic framework philosophies and then we focus on the most characteristic parts of the approaches this thesis focuses on.

3.1 Brief overview of various framework architectures in Neuro Symbolic AI on HAR

There are lots of implementations and frameworks that intend to interface a neural and a symbolic component, the main differentiating point is how the interfacing is done, as well as the expressivity of the symbolic language.

One of the most simple architectures involves training the NN separately from the symbolic component, using labeled examples for the in-between results that the NN outputs and then processing the results through the symbolic component[27]. This approach does not take advantage of involving the symbolic component into the learning phase, thus it is not much different from a pure NN model where the results are processed before being presented. In addition, labels are needed for the latent concepts as well, since the NN needs to be trained on making predictions on those concepts, not the final label.

Another approach is to embed the logic component into the NN[27], thus solving the problem with backpropagating loss through the symbolic component, since the logic is part of the NN and backpropagation is not much different from

traditional NN approaches. The issue with this approach is that the logic component is “lost” in the black box nature of the NN. There is no easy way to insert prior knowledge and no explainability to the results, since even the logical component can not be really separated from the NN.

There are frameworks that attempt to surpass the limitations listed above, with some of the most promising being NeurASP and the DeepProbLog framework along with its variants such as DeepProbCEP, DeepStochLog. The DPL framework that this thesis focuses on has a distinct symbolic component, which is part of the model and is treated as such from the training phase all the way through.

3.2 Human Activity Recognition with NeurASP and DPL on the CAVIAR dataset

The task that is used in the context of this thesis in order to evaluate the capabilities of Neuro - Symbolic approaches compared to traditional Deep Learning architectures in complex, real-world scenarios is classifying interactions between two people from a video feed, based on a subset of the CAVIAR dataset.

There has been work done on this exact scenario in NeurASP[23]. This task is chosen as opposed to a more traditional, task such as MNIST addition, which has been widely used as a benchmark on similar frameworks, as well as NeurASP, DeepProbLog and DeepStochLog[15], in order to evaluate the ability of those frameworks in more complex tasks, that have real world applications.

We will briefly describe the task at hand, as well as the dataset creation and the prior-knowledge that is introduced in the form of the logic program, based on the work done on evaluating NeurASP in this exact scenario.

3.2.1 The CAVIAR dataset

A subset of the CAVIAR dataset is used in this task, namely the section of video clips recorded in the entrance of the INRIA Labs at Grenoble, France, in the exact same fashion as the work in [23] on NeurASP.

Each datapoint consists of a clip that is 24 frames in length as well as the annotated actions of each person. The actions that involve each individual are referred to as simple events and they include one of the four following categories: inactive, active, walking, running. Each pair of people performs one of the following complex events at any point in time: meeting, moving and no interaction. For each frame, the width, height and coordinates of each bounding box are given, as well as the orientation of each of the persons in frame.

3.2.2 Transforming the dataset

The dataset is created by combining each clip of 24 frames with a complex event label for a pair of people in the frame. For each frame the xml annotation is used to create pairs of people and their corresponding interactions(or lack thereof).

Figure 12: A sample frame along with bounding box and orientation data visualized on top of the image.

We create three similar datasets from this point on, in order to test both an LSTM neural component in the task of sequence classification, as well as a

separate CNN component in the task of image classification, both in the context of Human Activity Recognition.

The dataset for the sequence classification case(LSTM)

The xml annotation and pairs of people, along with their interactions is used, while also taking into account other facts that are provided such as the bounding box and coordinates of each person, as well as their orientation.

The preprocessing of the data that results in the final form of the dataset used is based on the work done on [24], where the exact same dataset is used in an implementation that uses the NeurASP framework.

The simple events, the coordinates and the orientation are used to produce the complex event label that is used as the ground truth for the dataset. It is important to note that the intermediary labels are not used to train the model.

Ground truth is generated using NeurASP according to rules created for the CAVIAR dataset that were proposed in WOLED[25]. The equivalent complex event definitions in Prolog are used as part of the symbolic component in DeepProbLog and are part of Appendix A, as well as basic Event Calculus concepts such as the law of inertia that play a pivotal role in the logic of the programs.

A variant of this dataset for binary classification is created by combining the labels for meeting and moving into one “interaction” label. The Prolog rules of the original variant of the dataset are modified in order to create rules for the “interaction” and the “no_interaction” predicates.

The dataset for the image classification case(CNN)

The annotated interactions for each pair are used exactly like the example above, but the input of each datapoint is not a list of features, but the raw images themselves. The images are preprocessed and resized in order to be used.

The complex events are generated using a simple Python script that is logically equivalent to the Problog definitions of this task, which can be found on Appendix A. The resulting dataset is exactly like a normal CNN classification dataset with complex events.

Both datasets are used in the versions described above, as well as modified versions that are tailored to training with pure neural network

approaches, in order to use the results on Deep Learning model's performance on simple event classification and complex event classification as a baseline with which to compare the NeuroSymbolic models that are tested.

3.2.3 Dataset structure, simple and complex event counts

In the sequence classification case the input for each frame looks like the table below, while in the image classification case the input for each frame is an 80x80 image of the original caviar dataset videos.

Frame number	Person 1		Person 2		Shared	
	xcoordinate	y_coordinate	x_coordinate	y_coordinate	euclidean distance	orientation
1	0.8	0.61	0.81	1	0.55	0.95
2	0.8	0.62	0.81	1	0.54	0.95
3	0.81	0.61	0.82	1	0.55	0.97
4	0.83	0.61	0.83	1	0.56	0.97
5	0.84	0.62	0.83	1	0.56	1

In the table below the counts for the simple events for the image classification case, as well as the counts for the complex events are presented, in order to visualize the change in labels due to the change in the complex event definitions that produce the ground truth for each of the two datasets:

Integer encoding	Simple events		Complex events		
	Labels	Counts	Labels	Image classification(CNN)	Sequence classification(LSTM)
0	active	1642	no_interaction	8434	7824
1	inactive	3877	meeting	1172	2396
2	walking	8264	moving	4554	3124
3	running	377			

In the LSTM case, the dataset is split into 3 folds that are used to train and test the model. Each fold is stratified, meaning that the distribution between the classes stays the same. Each fold is split into a training(80% of the fold) and testing(20% of the fold) part.

For the CNN case the entirety of the dataset is used in one single fold, while retaining the 80-20 split between training and testing.

4. The DeepProbLog neural probabilistic programming language

The main characteristics of various neuro - symbolic architectures have been described in the previous sections. It is important to note that many architectures focus on various ways to integrate the symbolic component into the neural component, such as with fusing the equivalent logic rules inside the neural network architecture[7][8].

The framework this thesis is mainly focused on, namely DeepProbLog(DPL), approaches this problem the other way around, by finding a way to integrate the low-level subsymbolic component that is represented by neural network architectures inside an already established probabilistic logic programming language, thus inheriting the semantics and reasoning from the already established language with few modifications.

4.1 DeepProbLog architecture overview

The DeepProbLog neural probabilistic logic programming language is an extension of the well studied probabilistic logic programming language ProbLog which is itself based on the famous Problog programming language. The DPL language incorporates deep-learning into the logic programming language through neural predicates, thus it inherits the semantics and inference of the Problog programming language. This is in stark contrast to most similar approaches that attempt to incorporate the symbolic component into the neural one, or by keeping the two completely distinct, though tightly integrated.

One of the main advantages of this approach is that essentially the entire Problog programming language and its semantics is available to the model architecture, giving it a very powerful tool to express complex relationships and rules, as well as model noisy data and uncertainty through the use of probabilities. While most frameworks use less expressive languages like Datalog,DPL is Turing equivalent to Problog.

4.1.2 Neural predicates in DPL

In order to interface the sub-symbolic with the symbolic reasoner in a way that inherits all the features of the logic programming language, there is the need to interface the neural component in a seamless way with the symbolic one. This is achieved through the use of neural predicates.

Essentially after doing a forward pass through the neural network, the output layer is treated as if each of the output neurons was a predicate with probability equal to the output of the network at that neuron.

Example: a CNN that is incorporated into a DPL model that processes two images from the MNIST dataset and evaluates their sum, could be created in such a way so that it receives each image as an input and outputs a probability distribution on the possible digits, through a softmax output layer. Each of the neurons in the output is treated as a neural predicate, which for the purposes of inference and reasoning within the logic program is equivalent to a standard predicate with probability equal to that of the normalized output of the corresponding neuron.

```
nn(m_digit, [X], Y, [0...9]) :: digit(X,Y).
addition(X,Y,Z) :- digit(X,N1), digit(Y,N2), Z is N1+N2.
```

(a) The DeepProbLog program.

```
nn(m_digit,[0],0)::digit(0,0);nn(m_digit,[0], 1)::digit(0,1).
nn(m_digit,[1],0)::digit(1,0);nn(m_digit,[1], 1)::digit(1,1).
addition(0,1,1) :- digit(0,0), digit(1,1).
addition(0,1,1) :- digit(0,1), digit(1,0).
```

(b) The ground DeepProbLog program.

```
0.8 :: digit(0,0); 0.1 :: digit(0,1).
0.2 :: digit(1,0); 0.6 :: digit(1,1).
addition(0,1,1) :- digit(0,0), digit(1,1).
addition(0,1,1) :- digit(0,1), digit(1,0).
```

(c) The ground ProbLog program.

Figure 13: The example above shown in ground state.

This structure allows for the logic program to do inference using rules that are included in a program file, while also taking into account additional contextual facts that are dependent on the example that is being processed, along with the

output of the neural component, which can be directly translated into probabilistic facts. This is a very powerful tool that allows for more robust, data-efficient training of the neural network and as this thesis claims, a more fine tuned model, compared to a traditional neural network in the same task. This has already been demonstrated in the MNIST addition example above, the experiments of this thesis focus on proving the same point in a more complex, real-world scenario.

4.1.2 Integrating the neural and symbolic components in DPL

The input data is a combination of sub-symbolic and symbolic features, such as an image and some probabilistic facts, note that it is not necessary for symbolic input to be given. The probabilistic facts that are given as input are incorporated into the logic program, the technical details of the specific method that was used during the experimental phase are laid out in Appendix B. The logic program is given a query that needs to be evaluated, according to the rules given, the symbolic input and the subsymbolic input. During this process, when the logic program encounters a neural predicate, a forward pass is done through the corresponding network, and the output of the network is used as the probability of atomic probabilistic facts as defined in the logic program.

It is important to note that the forward pass is initiated when the logic program encounters a neural predicate that needs to be evaluated. Another fact that needs to be emphasized is that the targets are dependent on the logic program output directly, and only indirectly are dependent on the neural component, to the extent that the neural predicate outputs influence the logic program output.

The semantic loss of the logic program is given as gradients to the neural component during training.

4.2 Inference in DeepProbLog

Inference in DPL is similar to that in ProbLog, with the exception of the neural predicates which introduce the output of the neural component into the logic program. We can describe the process in four steps:

1. The program is given the symbolic and subsymbolic input and is given a query. The evaluation of this query is the output of the model. The first step involves grounding the logic program with respect to the query. During this phase if a neural predicate needs to be evaluated, the probability of the predicate is determined by the output of the neural network.
2. After the first step the query is reframed as an equivalent in terms of truth value formula in propositional logic that directly connects the value of the query to the truth values of the probabilistic facts.
3. The formula is then compiled to a Sentential Decision Diagram(SDD)[25], which is a representation of the propositional logic formula that allows for the use of knowledge compilation technology for more efficient evaluation.
4. After the logic formula is compiled into an SDD, the evaluation is performed from the bottom of the diagram towards the top to calculate the truth value of the query, expressed as a probability. This is done by evaluating each leaf of the diagram and multiplying if an “AND” node is encountered, or summing the probabilities if an “OR” node is encountered. The regular addition and multiplication during the traversal of the SDD are operations performed on the corresponding probabilities of probabilistic facts. The two operations along with the mapping of each fact to its corresponding probability defines a semiring.

Inference example

The inference mechanism can be better demonstrated using the following program as an example:

```
0.2 :: earthquake.
0.1 :: burglary.
alarm :− earthquake.
alarm :− burglary.
0.5 :: hearsalarm(mary).
calls(mary) :− alarm, hearsalarm(mary).
```

The SDD table which is used during ProbLog inference can be seen in the figure below:

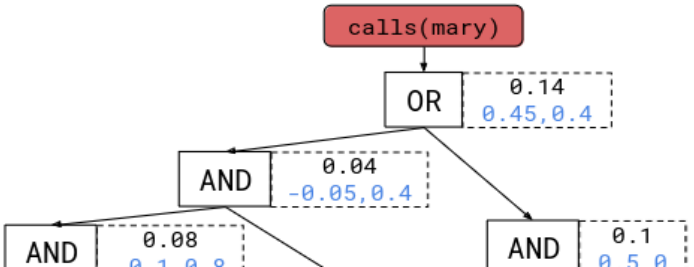


Figure 14: The SDD diagram for the program in the “Calls Mary” example

4.3 Learning in DeepProbLog

The DPL approach to training involves interpreting the difference of the output of the logic program compared to what the ground truth probability value of the query is, as a loss that is represented by a gradient that is backpropagated to tune the parameters of the logic program as well as the parameters of the neural network. To be exact: Given a DPL program with parameters X and a set Q of pairs(q,p) with q being the query and p being the ground truth probability of the query, a loss function L is used to compute: $argmin_{\vec{x}1/|Q|Sum_(q,p)L(P_(x=\vec{x},p)}$.

The problem of backpropagating seamlessly from the symbolic component to the neural, in order to tune the parameters in both components with one pass, gradient descent is used. The process is similar to inference in the initial steps, in the sense that the program is grounded with respect to a query that is used as a training example, the current parameters of the neural predicates, which are represented in the program in the form of neural Annotated Disjunctions. The loss is computed from the formula above, as well as the gradients which are finally used to update the parameters of the model.

It is important to note that this architecture allows the neural network training to be constrained by the logic in the symbolic component, due to the fact that the two are connected and loss/gradient calculations start from the symbolic component and then propagate backwards.

4.4 Gradient Semirings and aProbLog

Thus far no mechanism for the computation of the gradient is presented. There is no trivial way to convert the output of the logic program to a gradient, DPL solves the problem by using arithmetic circuits, turning the semantic loss to a gradient that can be backpropagated to train the NN

In order to compute the gradient an extension of ProbLog is used, called Algebraic ProbLog(aProbLog), which allows for inference to arbitrary commutative semirings, in our case the gradient semiring, as shown in [26].

4.4.1 Representing and calculating the gradient semiring in aProbLog

The traversal of the SDD is equivalent to doing multiplication and addition operations on the probability semiring, as described in the inference stage. This is a characteristic of the ProbLog programming language. The aProbLog variant generalizes this idea to arbitrary commutative semirings. Specifically, to compute gradients on the gradient semiring, each value from the semiring is associated with both facts and their negations and the multiplication and addition operations are defined in a more complex manner as follows:

$$(a_1, \vec{a_2}) + (b_1, \vec{b_2}) = (a_1 + a_2, b_1 + b_2) \qquad e^{\oplus} = (0, 0)$$

$$(a_1, \vec{a_2}) * (b_1, \vec{b_2}) = (a_1 b_1, b_1 a_2 + b_2 a_1) \qquad e^{\otimes} = (1, 0)$$

4.4.2 Gradient descent for DPL, based on aProbLog

Based on the gradient semiring the logic program can compute the loss gradient with respect to the output of the neural network, while treating it like a black box. The gradient is used to start the backpropagation process, which is then fed into the neural network, where using the loss gradient can be used in a standard way to tune the parameters of the network.

Example

```
flip(coin1).
flip(coin2).
nn(m_side, C, [heads, tails]) :: side(C, heads); side(C, tails).
t(0.5) :: red ; t(0.5) :: blue.
heads :- flip(X), side(X, heads).
win :- heads.
win :- \+heads, red.
query(win).
```

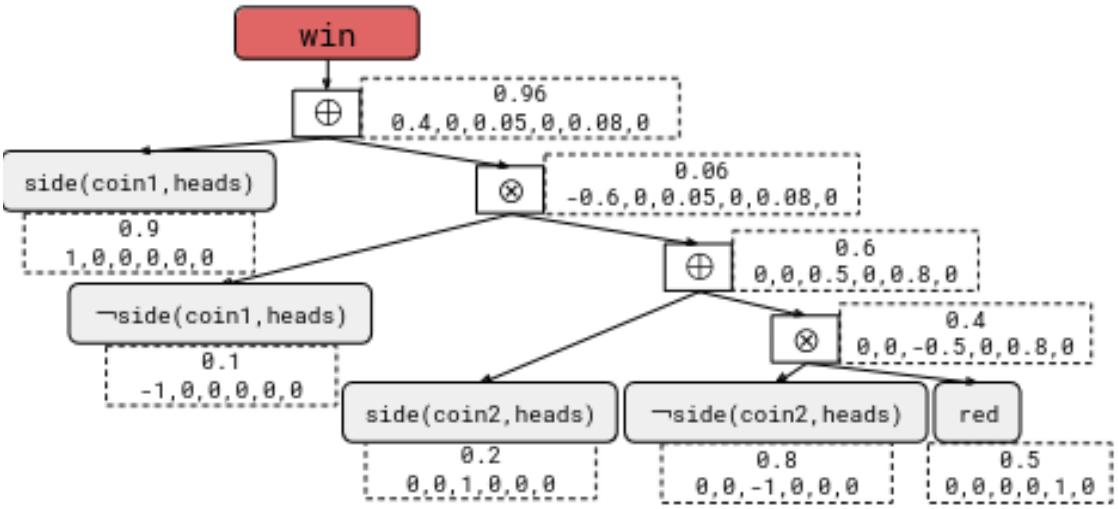


Figure 15: The SDD diagram, along with the computation in the coins example

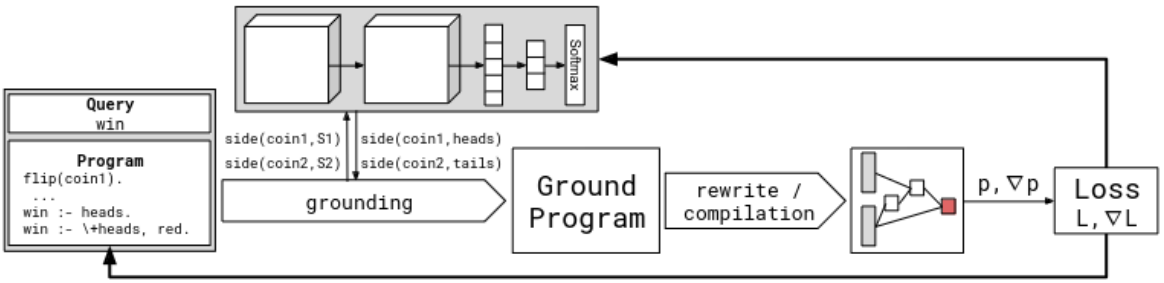


Figure 16: The learning pipeline as demonstrated through the coins example

5. Human Activity Recognition with DeepProbLog

In order to use DPL for human activity recognition, we need to ensure that the logic program describes the task at hand as specified, encoding the right logic rules. This is straightforward as the rules that define the complex events are known and defining the neural predicates is easy as well since the domain for the

simple event definitions is also well defined. The logic program can be found in the Appendix. In the following sections we focus on some aspects of the DPL implementation that need to be taken into account.

5.1 Solving the challenge of scalability

The definition of the law of inertia in the context of the task at hand is shown below:

```
holdsAt(Video, ComplexEvent, T) :-
    previous(T, T1),
    initiatedAt(Video, ComplexEvent, T1).

holdsAt(Video, ComplexEvent, T) :-
    holdsAt(Video, ComplexEvent, T1),
    previous(T, T1),
    \+terminatedAt(Video, ComplexEvent, T1).
```

The fact that the definition of the law of inertia is inherently recursive does not pose a problem for very small sequences(e.g. of length 3 or 4), but becomes a serious hurdle in the case of the CAVIAR clips which consist of 24 frames each. In this case the most straightforward formulation of the law of inertia results in unscalable inference and learning. This makes the most straightforward encoding of an Event Calculus interpreter in ProbLog unusable.

This is due to the fact that when the ProbLog compiler is queried about a specific timestep, it will start recursively building arithmetic circuits to solve the query. This means that if the compiler is queried about the fourth frame of a clip, it cannot assess if the holdsAt predicate is true for that frame, unless it assesses the same predicate for the preceding frames in reverse order, i.e frame 3, then 2, then 1. The number of arithmetic circuits becomes unbearably large very fast, since each circuit contains the previous one as a subcircuit.

Time	1	2	3	4	5	6	7	8	9	10	11	12	13
holdsAt(tensor, complex_event, T)	0	1	24	112	245	604	1304	3590	7955	25774	67106	199023	462658

The original DPL framework does not provide a way to solve this problem. In the setup tested a modification is made so that the recursive definition can be assessed without traversing the entire subsequence backwards, but by only visiting the previous timestep's assessment of the query. This means that we need to extend the definition of the Event Calculus interpreter.

5.2 Preserving previous inference and injecting facts at runtime in DPL

In order to solve the problem of scalability, we take advantage of the fact that the sequence is processed in chronological order by the logic program, which means that initial inference when it comes to the law of inertia is very fast, while later instances run into the problem presented in the previous section.

The solution to the problem is provided by storing the holdsAt predicate at each timestep and then injecting that assessment in the form of a fact into the next iteration through the sequence, thus eliminating the need for recursively recreating all the previous circuits at each timestep.

The solution itself is obvious, but implementing the solution while preserving the core functions of the framework is not so trivial, since in the standard implementation of DPL, injecting facts at runtime or in other words modifying the logic program while using it for inference is not possible. The technical details of the injection are provided in the Appendix.

After injecting the previous assessment of the holdsAt predicates are modified as shown below:

```
holdsAt(Video, T, F) :-
    previous(T1, T),
    initiatedAt(F, Video, T1).

holdsAt(Video, T, F) :-
    previous(T1, T),
    cached(Video, F, T1),
    \+terminatedAt(F, Video, T1).
```

Where the cached(Video, F, T1) predicate is injected only if the predicate holdsAt(Video, T1, F) is evaluated as true in the previous step of the sequence.

5.3 Adding the distance and orientation contextual facts

In the definitions of the complex events both the distance between the pair of persons and their orientation with respect to one another are taken into account. For example the meeting complex event is thought to take place if -among other things- the two people meeting are relatively close to one another. The orientation of each person in the frame, as well as their positions are provided by the CAVIAR dataset as input.

The issue is that there is no intuitive way to input the distances and orientation in the form of input to the logic program. At the same time it is trivial to give the above values as input to the neural network but there is no apparent way to translate that input as a neural predicate.

The solution that we resort to is to add the orientation and distance values to each tensor for the input of the neural network, to ensure that the correct values are being handled each time the neural predicates are evaluated through the neural network. The NN itself is modified to ignore those inputs and a modification to the DPL framework takes those values and injects them as facts in the logic program along with the cached holdsAt predicate, in order to be used once during inference without modifying the core of the logic program.

5.4 Input tensors and labels

The input tensors are formed using the original CAVIAR dataset tensors provided for each person in frame, which include the bounding box size, coordinates and the orientation of each person. The height, width and coordinates for each person are concatenated, and two features are concatenated in the end, the difference of the two person’s orientations and the euclidean distance between the two boxes.

5.5 DPL learning and inference in HAR

The input tensor is passed through the LSTM network which then outputs one simple event label for each person's activity for each given frame. For each frame the output of the neural network is used to produce neural predicates that are used in the logic program to do inference, for example, the following neural predicates are produced:

```
0.06 :: happensAt(0, person1, 0, active).
0.09 :: happensAt(0, person1, 0, inactive).
0.73 :: happensAt(0, person1, 0,running).
0.12 :: happensAt(0, person1, 0,walking).
```

The output for each person along with the orientation and distance features are passed in the form of predicates to the logic program, in order for inference in problog to take place. In the example above, the final predicates that are passed on to the program, for timestep 0 are:

```
0.06 :: happensAt(0, person1, 0, active).
0.09 :: happensAt(0, person1, 0, inactive).
0.73 :: happensAt(0, person1, 0,running).
0.12 :: happensAt(0, person2, 0,walking).
0.08 :: happensAt(0, person2, 0, active).
0.13 :: happensAt(0, person2, 0, inactive).
0.57 :: happensAt(0, person2, 0,running).
0.22 :: happensAt(0, person2, 0,walking).
is_close(0, person1, person2, 0, 34),
orientation(0, person1, person2, 0).
```

The logic program does inference according to the aProbLog implementation described above. Each of the predicates that correspond to the three possible complex events for each timepoint are assigned a probability, with the sum of all three being 1, thus the output of the logic program is a probability distribution over the possible outcomes, which can be interpreted as the certainty of the model that each outcome holds, with the one with the highest probability assigned to it being the one that is considered to be the output of the model for that timepoint.

6. Experimental implementations

In this section the implementations of the models that are used are explained, as well as the results each implementation yields in the test dataset. The purpose of this comparison is to prove that NeuroSymbolic approaches can have an advantage over traditional Deep Learning models in complex, real world applications, such as the task of Complex Event Recognition in HAR.

For the LSTM experimental setup the two approaches used are as follows:

- A pure Deep Learning approach, where a bi-LSTM model is trained on the sequences of features that are contained in the dataset in order to directly classify which complex event is taking place each time.
- A DeepProblog approach, where a bi-LSTM model is trained on the latent simple events that take place in each frame and the output is passed onto the Problog component in the form of neural predicates in order to perform inference and output the final complex event classification for each frame in the sequence.

An identical setup is used for the binary classification variant.

For the CNN experimental setup the two approaches used are as follows:

- A pure Deep Learning approach, where a CNN model is trained on the images that are contained in the dataset in order to directly classify which complex event is taking place each time.
- A DeepProblog approach, where a CNN model is trained on the latent simple events that take place in each frame and the output is passed onto the Problog component in the form of neural predicates in order to perform inference and output the final complex event classification for each frame in the sequence.

6.1 LSTM experimental setup for the multilabel case

6.1.1 Deep Learning approach

The hyperparameters of the bi-LSTM model are the same as the ones used in [24], in order for it to play the role of a “benchmark”, with which the different models are compared.

6.1.1.1 LSTM network architecture and hyperparameters

The hyperparameters are as follows:

- Loss function: Cross Entropy loss
- Learning rate: 1.5E-06
- Hidden layers: 2
- Neurons per layer: 256
- Dropout: 0.3
- Epochs: 1000
- Batch size: 8

6.1.1.2 Results

The results for the three folds as well as the average performance of the model across the entire test dataset are as follows:

Fold 1			
	Precision	**Recall**	**F1-score**
no_interaction	0.66	0.84	0.74
meeting	0.49	0.22	0.3
moving	0.48	0.35	0.35
accuracy			0.6
macro avg			0.48
weighted avg			0.56
Fold 2			
	Precision	**Recall**	**F1-score**
no_interaction	0.63	0.6	0.61
meeting	0.41	0.71	0.52
moving	0.53	0.44	0.48
accuracy			0.56
macro avg			0.54
weighted avg			0.56

Fold 3			
	Precision	**Recall**	**F1-score**
no_interaction	0.69	0.84	0.76
meeting	0.72	0.75	0.73
moving	0.65	0.41	0.5
accuracy			0.69
macro avg			0.69
weighted avg			0.67
Average f-1 score			0.56
Weighted average f-1 score			0.6

6.1.2 DeepProbLog approach

For the DPL model the parameters were as close to the original ones as possible, the number of epochs is reduced, since further experimentation showed no improvement after the first 5 epochs, while the internal workings of the DPL framework made necessary to change the batch number.

6.1.2.1 Neural component architecture and hyperparameters

The hyperparameters are as follows:

- Loss function: Cross Entropy loss
- Learning rate: 1.00E-06
- Hidden layers: 2
- Neurons per layer: 256
- Dropout: 0.3
- Epochs: 5
- Batch size: 1

6.1.2.2 Results

The results for the three folds as well as the average performance of the model across the entire test dataset are as follows:

Fold 1			
	Precision	**Recall**	**F1-score**
no_interaction	0.88	1	0.93

meeting	0.28	0.77	0.42
moving	0.79	0.15	0.25
accuracy			0.69
macro avg			0.53
weighted avg			0.65
Fold 2			
	Precision	**Recall**	**F1-score**
no_interaction	0.82	0.97	0.89
meeting	0.24	0.6	0.34
moving	0.79	0.15	0.26
accuracy			0.65
macro avg			0.5
weighted avg			0.61
Fold 3			
	Precision	**Recall**	**F1-score**
no_interaction	0.85	0.98	0.91
meeting	0.31	0.8	0.44
moving	0.79	0.16	0.27
accuracy			0.68
macro avg			0.54
weighted avg			0.64
Average f-1 score			0.53
Weighted average f-1 score			0.63

6.2 LSTM experimental setup for the binary case

6.2.1 LSTM network architecture and hyperparameters

The hyperparameters are as follows:

- Loss function: Cross Entropy loss
- Learning rate: 1.5E-06
- Hidden layers: 2
- Neurons per layer: 256
- Dropout: 0.3
- Epochs: 1000

- Batch size: 8

Results

The results for the three folds as well as the average performance of the model across the entire test dataset are as follows:

Fold 1			
	Precision	**Recall**	**F1-score**
no_interaction	0.70	0.78	0.74
interaction	0.27	0.91	0.41
accuracy			0.69
macro avg			0.53
weighted avg			0.65
Fold 2			
	Precision	**Recall**	**F1-score**
no_interaction	0.82	0.97	0.89
interaction	0.79	0.15	0.26
accuracy			0.65
macro avg			0.5
weighted avg			0.61
Fold 3			
	Precision	**Recall**	**F1-score**
no_interaction	0.85	0.98	0.91
interaction	0.79	0.16	0.27
accuracy			0.68
macro avg			0.54
weighted avg			0.64
Average f-1 score			0.53
Weighted average f-1 score			0.63

6.2.2 DeepProbLog approach

The hyperparameters are as follows:

- Loss function: Cross Entropy loss
- Learning rate: 0.001
- Hidden layers: 2
- Neurons per layer: 256
- Dropout: 0.3
- Epochs: 2
- Batch size: 1

Results

The results for the three folds as well as the average performance of the model across the entire test dataset are as follows:

Fold 1			
	Precision	**Recall**	**F1-score**
no_interaction	1.00	0.93	0.96
interaction	0.94	1	0.97
accuracy			0.97
macro avg			0.97
weighted avg			0.97
Fold 2			
	Precision	**Recall**	**F1-score**
no_interaction	0.91	0.96	0.93
interaction	0.94	0.88	0.91
accuracy			0.92
macro avg			0.92
weighted avg			0.92
Fold 3			
	Precision	**Recall**	**F1-score**
no_interaction	0.91	0.98	0.94
interaction	0.97	0.88	0.93
accuracy			0.94
macro avg			0.93
weighted avg			0.94
Average f-1 score			0.94

Weighted average f-1 score	0.94

6.3 CNN experimental setup

6.3.1 Deep Learning approach

The hyperparameters of the CNN model in both approaches are similar, with the intention that the needs in convolutional layers are similar in both cases, while the logical component is substituted by the fully connected layers in the second part of the model.

6.3.1.1 CNN network architecture and hyperparameters

- Loss function: Cross Entropy loss
- Learning rate: 1E-03
- Hidden layers: 5
- Convolutional layers: 3
- Neurons per layer: 128
- Dropout: 0.5
- Epochs: 10
- Batch size: 256

Results

The result of the model is as follows:

	Precision	Recall	F1-score
no_interaction	0.94	0.79	0.86
meeting	0.0	0.0	0.0
moving	0.77	0.85	0.80
accuracy			0.81
macro avg			0.55
weighted avg			0.77

6.3.2 DeepProbLog approach

The results for the DeepProbLog approach, as well as the neural component architecture are listed below.

6.3.2.1 Neural component architecture and hyperparameters

- Loss function: Cross Entropy loss

- Learning rate: 1E-03
- Hidden layers: 5
- Convolutional layers: 3
- Neurons per layer: 128
- Dropout: 0.5
- Epochs: 10
- Batch size: 256

Results

The results for the three folds as well as the average performance of the model across the entire test dataset are as follows:

	Precision	Recall	F1-score
no_interaction	0.59	1.00	0.74
meeting	0.00	0.00	0.00
moving	0.00	0.00	0.00
accuracy			0.59
macro avg			0.25
weighted avg			0.44

7. Conclusion

7.1 Evaluating the results

The results show that the DeepProbLog model managed to surpass the bi-LSTM by a small margin in the weighted macro F-1 score metric, while managing to train the neural component in only 5 epochs compared to 1000 used in the bi-LSTM approach. This is an important result, showing that the DPL model is marginally better than the traditional Deep Learning model, while being much more data efficient, managing to converge in only 5 epochs.

The results of the binary variant show a significant improvement of the DeepProbLog model compared to the traditional Deep Learning approach. Not only is the f-1 score of the model significantly better(0.94 versus 0.63 for the neural network), but the model also manages to train in just two epochs compared to the 1000 epochs needed for the baseline. It is important to note that the neural component is the same for both cases, with the only difference being that the DeepProbLog model has the logical component and the complex event rules. This result highlights the advantages of Neuro-Symbolic approaches compared to traditional Deep Learning methods.

The results of the CNN model show that the DeepProbLog framework, being an experimental implementation that lacks polishing, fails to provide the required learning signal in this task. The author of the thesis attributes this result to bugs in the DPL code, a hypothesis that is supported to some extent by the large number of issues that arose during the writing of the code for this thesis and the modifications that were required, as shown in section 5.

7.2 What this thesis aimed to prove

The aim of this thesis was to prove that NeuroSymbolic frameworks are capable of creating models that can perform in real world, demanding applications, without altering the dataset structure or requiring further annotation than what is required for deep learning approaches to be applied.

Another goal of this thesis was to prove that NeuroSymbolic frameworks can create models that compete with pure deep learning approaches in complex tasks such as the Human Activity Recognition task tested in the experimental section and even surpass pure deep learning approaches in some metrics.

7.3 Conclusions on the capabilities of DeepProbLog in HAR

By comparing the results of the experimental evaluation chapter it is obvious that the DeepProbLog framework, despite the fact that it is an experimental implementation that cannot compare to the amount of optimization and polishing that a framework like TensorFlow and Keras represent, can produce models that compare to pure deep learning approaches in some cases.

This thesis proved the hypothesis that was based on the results of work done on[12][13] among others, that showed the capabilities of NeuroSymbolic approaches on simpler tasks, such as MNIST addition, while being capable of outperforming pure deep learning approaches on some metrics, such as performance and data efficiency.

The experimental results stand as proof that the hypotheses this thesis aimed to test are true, while also highlighting the fact that NeuroSymbolic frameworks have reached a state of maturity that allows for their applications beyond simple tasks and toy models as was the case in the past. This conclusion does not negate the fact that there are many improvements to be made in the implementation and design of such frameworks in order for them to be more widely used and accepted.

7.4 Difficulties encountered and future work

One of the main difficulties encountered during the course of this thesis was the fact that the DPL framework, like similar other frameworks that are based on recent publications and approach in a novel way the task of coupling subsymbolic(neural) and symbolic reasoning, lacks proper documentation and examples of applications in real world examples apart from the ones used in the papers. In addition the implementations understandably lack refining in some areas, which makes it difficult to expand the scenarios that the framework has

been tested on, due to the fact that modifications to the source code and the way the framework works are sometimes necessary, as evidenced by the fact that the original framework had no apparent mechanism for passing though contextual facts and blocking recursive calls ballooning in the later stages of each sequence.

This difficulty could be the cause of lacking performance in the multilabel sequence classification test, compared to other approaches[24], while also limiting the scope of this thesis from trying more datasets and different settings due to time constraints.

Future work could focus on using the framework on other tasks and datasets and comparing it to pure deep learning approaches or in implementing and comparing other frameworks and approaches that have been recently introduced such as DeepStochLog among others.

References

1. Nikos Katzouris Scalable Relational Learning for Event Recognition, PhD Thesis, National and Kapodistrian University of Athens (2017).
2. Garcez, Artur & Gori, Marco & Lamb, Luís & Serafini, Luciano & Spranger, Michael & Tran Neural-Symbolic Computing: An Effective Methodology for Principled Integration of Machine Learning and Reasoning, Son. (2019)
3. Artikis et al. 2012
4. Artikis, Alexander, Marek Sergot, and Georgios Paliouras. "Reactive reasoning with the event calculus." (2015).
5. Giannesini, F., Kanoui, H., Pasero, R., & Van Caneghem, M. (1986). *Prolog*. Addison-Wesley Longman Publishing Co., Inc.
6. M. M. Gupta, "Fuzzy logic and neural networks," *[Proceedings 1992] IEEE International Conference on Systems Engineering*, Kobe, Japan, 1992, pp. 636-639, doi: 10.1109/ICSYSE.1992.236895.
7. James M. Keller, Ronald R. Yager, Hossein Tahani, “Neural network implementation of fuzzy logic”, Fuzzy Sets and Systems, Volume 45, Issue 1, 1992, Pages 1-12, ISSN 0165-0114
8. Zhiting Hu, Xuezhe Ma, Zhengzhong Liu, Eduard Hovy, Eric Xing, “Harnessing Deep Neural Networks with Logic Rules”
9. Gianpaolo Cugola and Alessandro Margara. 2012. Processing flows of information: From data stream to complex event processing. ACM Comput. Surv. 44, 3, Article 15 (June 2012), 62 pages. https://doi.org/10.1145/2187671.2187677
10. Jaykumar Kakkad, Jaspal Jannu, Kartik Sharma, Charu Aggarwal, Sourav Medya, “A Survey on Explainability of Graph Neural Networks”
11. http://homepages.inf.ed.ac.uk/rbf/CAVIARDATA1
12. Yang, Zhun, Adam Ishay, and Joohyung Lee. "Neurasp: Embracing neural networks into answer set programming." *arXiv preprint arXiv:2307.07700* (2023).

13. Manhaeve, Robin, et al. "Deepproblog: Neural probabilistic logic programming." *Advances in neural information processing systems* 31 (2018).
14. Kimmig, Angelika, et al. "On the implementation of the probabilistic logic programming language ProbLog." *Theory and Practice of Logic Programming* 11.2-3 (2011): 235-262.
15. Winters, Thomas, et al. "Deepstochlog: Neural stochastic logic programming." *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 36. No. 9. 2022.
16. O'shea, Keiron, and Ryan Nash. "An introduction to convolutional neural networks." *arXiv preprint arXiv:1511.08458* (2015).
17. Yu, Yong, et al. "A review of recurrent neural networks: LSTM cells and network architectures." *Neural computation* 31.7 (2019): 1235-1270.
18. Hochreiter, Sepp. "The vanishing gradient problem during learning recurrent neural nets and problem solutions." *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* 6.02 (1998): 107-116.
19. Ha, Qishen, Bo Liu, and Fuxu Liu. "Identifying melanoma images using efficientnet ensemble: Winning solution to the siim-isic melanoma classification challenge." *arXiv preprint arXiv:2010.05351* (2020).
20. Garcez, Artur d'Avila, et al. "Neural-symbolic computing: An effective methodology for principled integration of machine learning and reasoning." *arXiv preprint arXiv:1905.06088* (2019).
21. Garcez, Artur d'Avila, and Luis C. Lamb. "Neurosymbolic AI: The 3rd wave." *Artificial Intelligence Review* 56.11 (2023): 12387-12406.
22. Sterling, Leon, and Ehud Y. Shapiro. *The art of Prolog: advanced programming techniques*. MIT press, 1994.
23. Oikonomakis, Andreas. *Neuro-symbolic answer set programming for human activity recognition in videos*. MS thesis. Πανεπιστήμιο Πειραιώς, 2023.
24. Katzouris, Nikos, and Alexander Artikis. "WOLED: a tool for online learning weighted answer set rules for temporal reasoning under uncertainty." *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning*. Vol. 17. No. 1. 2020.

25. A. Darwiche, SDD: A new canonical representation of propositional knowledge bases, in: Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence, IJCAI-11, 2011, pp. 819–826.
26. Kimmig, Angelika, Guy Van den Broeck, and Luc De Raedt. "An algebraic Prolog for reasoning about possible worlds." *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 25. No. 1. 2011.
27. Sarker, Md Kamruzzaman et al. 'Neuro-symbolic Artificial Intelligence'. 1 Jan. 2021 : 197 – 209.
28. ProbLog reasoning system: Hahn, S., Janhunen, T., Kaminski, R., Romero, J., Rühling, N., & Schaub, T. (2022). plingo: A system for probabilistic reasoning in clingo based on lpmln. RuleML+RR, 54-62.
29. Lecun, Yann; Boser, B.; Denker, J. S. et al, Handwritten digit recognition with a back-propagation network, Advances in Neural Information Processing Systems (NIPS 1989), Denver, CO. ed, David Touretzky. Vol. 2 Morgan Kaufmann, 1990.
30. Krizhevsky, Alex; Sutskever, Ilya; Hinton, Geoffrey E. (2017-05-24). "ImageNet classification with deep convolutional neural networks", Communications of the ACM. 60 (6): 84–90.

Appendix A

The logic program in case 1

```
nn(caviar_net, [Video, P, T], SE, [active, inactive, running, walking]) :: happensAt(Video, P, T, SE).

holdsAt(Video, F, T) :-
    previous(T1, T),
    initiatedAt(F, Video, T1).

holdsAt(Video, F, T) :-
    previous(T1, T),
    previous_step(Video, F, T1),
    \+terminatedAt(F, Video, T1).

%moving complex event
initiatedAt(moving(P1, P2), Video, T):-
        happensAt(Video, P1, T, walking),
        happensAt(Video, P2, T, walking),
        is_close(Video, P1, P2, T, 34),
        orientation(Video, P1, P2, T).

terminatedAt(moving(P1,P2), Video, T) :-
        happensAt(Video, P1, T, walking),
        far(Video, P1, P2, T, 34).

terminatedAt(moving(P1,P2), Video, T) :-
        happensAt(Video, P2, T, walking),
        far(Video, P1, P2, T, 34).

terminatedAt(moving(P1,P2),T) :-
        happensAt(Video, P1, T, active),
        happensAt(Video, P2, T, active).

terminatedAt(moving(P1,P2),T) :-
        happensAt(Video, P1, T, active),
        happensAt(Video, P2, T, inactive).

terminatedAt(moving(P1,P2),T) :-
```

```
        happensAt(Video, P1, T, inactive),
        happensAt(Video, P2, T, active).

terminatedAt(moving(P1,P2),T) :-
        happensAt(Video, P1, T, running).

terminatedAt(moving(P1,P2),T) :-
        happensAt(Video, P2, T, running).

%meeting complex event
initiatedAt(meeting(P1, P2), Video, T) :-
        happensAt(Video, P1, T, active),
        happensAt(Video, P2, T, active),
        is_close(Video, P1, P2, T, 25).

initiatedAt(meeting(P1, P2), Video, T) :-
        happensAt(Video, P1, T, active),
        happensAt(Video, P2, T, inactive),
        is_close(Video, P1, P2, T, 25).

initiatedAt(meeting(P1, P2), Video, T) :-
        happensAt(Video, P1, T, inactive),
        happensAt(Video, P2, T, active),
        is_close(Video, P1, P2, T, 25).

initiatedAt(meeting(P1, P2), Video, T) :-
        happensAt(Video, P1, T, inactive),
        happensAt(Video, P2, T, inactive),
        is_close(Video, P1, P2, T, 25).

terminatedAt(meeting(P1, P2), Video, T) :-
        happensAt(Video, P1, T, running).

terminatedAt(meeting(P1, P2), Video, T) :-
        happensAt(Video, P2, T, running).

terminatedAt(meeting(P1, P2), Video, T) :-
        happensAt(Video, P1, T, walking),
        far(Video, P1, P2, T, 25).

terminatedAt(meeting(P1, P2), Video, T) :-
        happensAt(Video, P2, T, walking),
        far(Video, P1, P2, T, 25).
```

```
%nointeraction complex event
holdsAt(Video, nointeraction(P1,P2), T) :-
        \+holdsAt(Video, meeting(P1,P2), T),
        \+holdsAt(Video, moving(P1,P2),T).

terminatedAt(nointeraction(P1,P2), Video, T) :-
        holdsAt(Video, nointeraction(P1,P2), T).

previous(T1, T) :-
        T >= 0,
        T1 is T-1,
        T1 >= 0.

is_close(Video, P1, P2, T, D) :- distance(Video, P1, P2, T, D1), D1 =< D.
far(Video, P1, P2, T, D) :- distance(Video, P1, P2, T, D1), D1 > D.
```

The logic program in case 2

```
nn(caviar_cnn, [Video, P, T], SE, [active, inactive, walking, running]) :: happensAt(Video,
P, T, SE).

holdsAt(Video, meeting(P1, P2), T) :-
        happensAt(Video, P1, T, active),
        happensAt(Video, P2, T, active).

holdsAt(Video, meeting(P1, P2), T) :-
        happensAt(Video, P1, T, inactive),
        happensAt(Video, P2, T, active).

holdsAt(Video, meeting(P1, P2), T) :-
        happensAt(Video, P1, T, active),
        happensAt(Video, P2, T, inactive).

holdsAt(Video, meeting(P1, P2), T) :-
        happensAt(Video, P1, T, inactive),
        happensAt(Video, P2, T, inactive).

holdsAt(Video, moving(P1, P2), T) :-
        happensAt(Video, P1, T, walking),
        happensAt(Video, P2, T, walking).
```

```
holdsAt(Video, nointeraction(P1, P2), T) :-
    \+holdsAt(Video, moving(P1, P2), T),
    \+holdsAt(Video, meeting(P1, P2), T).
```

List of Figures