Chapter 2

Literature Review*

In this chapter, we present a survey of techniques in which existing domain-knowledge is included when constructing models with deep neural networks.

In this survey, we restrict the studies on the incorporation of domain-knowledge into neural networks with one or more hidden layers. If the domain-knowledge expressed in a symbolic form (for example, logical relations that are known to hold in the domain), then the broad area of hybrid neural-symbolic systems (see for example, [GBG12, RDMM20]) is clearly relevant to the material in this chapter. However, the motivation driving the development of hybrid systems is much broader than this chapter, being concerned with general-purpose neural-based architectures for logical representation and inference. Here our goals are more modest: we are looking at the inclusion of problem-specific information into machine-learning models of a kind that will be described shortly. We refer the reader to [BGB+17] for reviews of work in the broader area of neural-symbolic modelling. More directly related to the survey in this chapter is the work on "informed machine learning", reviewed in [vRMB+21]. We share with this work the interest in prior knowledge as an important source of information that can augment existing data. However, the goals of that paper are more ambitious than here. It aims to identify categories of prior knowledge, using as dimensions: the source of the knowledge, its representation, and its point of use in a machine-learning algorithm. In this survey, we are only concerned with some of these categories. Specifically, in terms of the categories in [vRMB+21], we are interested in implicit or explicit sources of domain-knowledge, represented either as logical or numeric constraints and used in the model-construction stage by DNNs. Informal examples of what we mean by logical and numerical constraints are shown in Figure 2.1. In general, we will assume logical constraints can, in principle, be represented as statements in propositional logic or predicate logic. Numerical constraints will be representable, in

*The content of this chapter is based on the following:
T. Dash, S. Chitlangia, A. Ahuja, A. Srinivasan, "A review of some techniques for inclusion of domain-knowledge into deep neural networks", *Nature Scientific Reports*, 2022; https://doi.org/10.1038/s41598-021-04590-0.

principle, as terms in an objective function being minimised (or maximised), or prior distributions on models. We believe this covers a wide range of potential applications, including those concerned with scientific discovery.

For inhibiting this protein:	The model should follow that:
The presence of a peroxide	$p(y = 1\|\mathbf{x}) \geq 0.9$
bridge is relevant.	$p(y = 0\|\mathbf{x}) \leq 0.1$
The target site is	Initial weights should be
at most 20Å.	$3n - 2.3$ [TSN90]
(a)	**(b)**

Figure 2.1: Informal descriptions of (a) logical; and (b) numerical constraints.

2.1 Focus of this Review

We adhere to the following informal specification for constructing a deep neural network: given some data D, a structure and parameters of a deep neural network (denoted by π and $\boldsymbol{\theta}$, respectively), a learner $\mathcal{L}$ attempts to construct a neural network model M that minimises some loss function L. Figure 2.2 shows a diagrammatic representation. Note that: (a) we do not describe how the learner $\mathcal{L}$ constructs a model M given the inputs. But, it would be normal for the learner to optimise the loss L by performing an iterative estimation of the parameters $\boldsymbol{\theta}$, given the structure π; and (b) we are not concerned with how the constructed deep model M will be used. However, it suffices to say that when used, the model M would be given one or more data-instances encoded in the same way as was provided for model-construction.

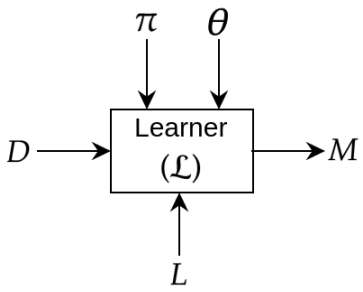


Figure 2.2: Construction of a deep neural network model M from data (D) using a learner ($\mathcal{L}$). We use π to denote the structure (organisation of various layers, their interconnections, etc.) and $\boldsymbol{\theta}$ to denote the parameters (synaptic weights) of the deep neural network. L denotes the loss function (for example, cross-entropy loss in case of classification).

In the literature, domain-knowledge–also called background knowledge–does not appear to have an accepted definition, other than that, it refers to information about the

problem. This information can be in the form of relevant features, concepts, taxonomies, rules-of-thumb, logical constraints, probability distributions, mathematical distributions, causal connections and so on. In this chapter, we use the term "domain-knowledge" to refer to problem-specific information that can directly be translated into alterations to the principal inputs of Figure 2.2. That is, by domain-knowledge we will mean problem-specific information that can change: (1) The input data to a deep neural network; (2) The loss-function used; and (3) The model (that is, the structure or parameters) of the deep neural network. In a sense, this progression reflects a graded increase in the complexity of changes involved. Figure 2.3 tabulates the principal implications of this position for commonly-used deep learning architectures.

The rest of the chapter is divided into multiple sections: We start by discussing some existing techniques for inclusion of domain-knowledge by augmenting or transforming inputs. Then we focus on some popular loss functions that are used to drive the inclusion of domain-knowledge during training of a deep neural network. We describe various biases on model parameters and changes to the structure of deep neural networks. In summary, we outline some major challenges related to the inclusion of domain-knowledge in the ways we describe and where the contributions of this lie in this area of research.

2.2 Transforming the Input Data

One of the prominent approaches to incorporate domain-knowledge into a deep neural network is by changing inputs to the network. Here, the domain-knowledge is primarily in symbolic form. The idea is simple: If a data instance could be described using a set of attributes that not only includes the raw feature-values but also includes more details from the domain, then a standard deep neural network could then be constructed from these new features. A simple block diagram in Figure 2.4 shows how domain-knowledge is introduced into the network via changes in inputs. In this survey, we discuss broadly two different ways of doing this: (a) using relational features, mostly constructed by a method called propositionalisation [LDG91] using another machine learning system (for example, Inductive Logic Programming) that deals with data and background knowledge; (b) without propositionalisation.

2.2.1 Propositionalisation

The pre-eminent form of symbolic machine learning based on the use of relations in first-order logic is Inductive Logic Programming (ILP) [Mug91], which has an explicit role for domain-knowledge being incorporated into learning. The simplest use of ILP [Mug91] to incorporate n-ary relations in domain-knowledge into a neural network relies on techniques that automatically "flatten" the domain-knowledge into a set of domain-specific

Arch.	Domain-Knowledge Effect		
	Transform Input Data	**Transform Loss Function**	**Transform Model**
MLP	Reformulate feature-representation	(For all architectures) Reformulate regularisation term; Addition of syntactic or semantic constraints with associated penalties; Differential costs of decisions	Changes in layers, hidden units
CNN	Reformulate spatial-representation		As with MLPs, plus changes to connections between units; changes convolution filters
RNN, Transformer	Reformulate sequence-representation		As with MLPs, plus possible changes to attention-mechanism
GNN	Reformulate graph-representation		As with MLPs, plus changes to graph-convolution

Figure 2.3: Some implications of using domain-knowledge for commonly-used deep neural network architectures. Here MLP stands for Multilayer Perceptron, CNN stands for Convolutional Neural Network, RNN stands for Recurrenural Network and GNN stands for Graph Neural Network. MLPs, CNNs and RNNs are now commonplace architectures for deep neural networks and detailed descriptions can be found in any standard textbook (for example, [BGC17, ZLLS21]). GNNs are increasingly the DNN model of choice for dealing with graph-based data, and a good description can be found in [Ham20]. In this dissertation, we will be mainly concerned with MLPs and GNNs: the details required are in Appendix A.

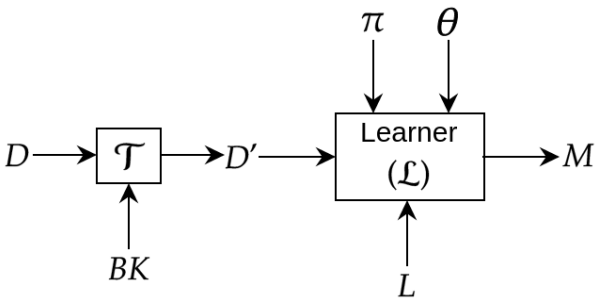


Figure 2.4: Introducing background knowledge into deep neural network by transforming data. $\mathcal{T}$ is a transformation block that takes input data D, background knowledge (BK) and outputs transformed data D' that is then used to construct a deep model using a learner $\mathcal{L}$.

relational features. Although not all DNNs require data to be a set of feature-vectors, this form of data representation is long-standing and still sufficiently prevalent. In logical terms, we categorise feature-based representations as being encodings in propositional logic. The reader would point out, correctly, that feature-values may not be Boolean. This is correct, but we can represent non-Boolean features by Boolean-valued propositions (for example, a real-valued feature f with value 4.2 would be represented by a corresponding Boolean feature f' that has the value 1 if $f = 4.2$ and 0 otherwise). With the caveat of this rephrasing, it has, of course, been possible to provide domain-knowledge to neural networks by employing domain-specific features devised by an expert. However, we focus here on ways in which domain-knowledge encoded as rules in propositional logic and first-order logic has been used to construct the input features for a deep neural network. Techniques for automatic construction of Boolean-valued features from relational domain-knowledge have a long history in the field of ILP [Md94, MDRP+12, CDM20], originating from the LINUS [LDG91]. Often called *propositionalisation*, the approach involves the construction of features that identify the conditions under which they take on the value 1 (or 0). For example, given (amongst other things) the definition of benzene rings and of fused rings, an ILP-based propositionalisation may construct the Boolean-valued feature that has the value 1 if a molecule has 3 fused benzene rings, and 0 otherwise. The values of such Boolean-valued features allow us to represent a data instance (like a molecule) as a Boolean-valued feature-vector, which can then be provided to a neural network. There is a long history of propositionalisation: see [KLF01] for a review of some of early use of this technique, and [LSR20, VSBV17] who examine the links between propositionalisation and modern-day use of embeddings in deep neural networks. More clearly, the authors examine that both propositionalisation and embedding approaches aim at transforming data into tabular data format while they are being used in different problem settings and contexts. One recent example of embedding is demonstrated in [GRS+21] where the authors use different text-embedding approaches such as sentence encoder [CYK+18] and GPT2 [RWC+19] to transform textual domain-knowledge into embedding vectors.

A direct application of propositionalisation, demonstrating its utility for deep neural networks, has been its use in Deep Relational Machines (DRMs: [Lod13]). A DRM is a deep fully-connected neural network with Boolean-valued inputs obtained from propositionalisation by an ILP engine.

The idea of propositionalisation also forms the foundation for a method known as 'Bottom Clause Propositionalisation (BCP)' to propositionalise the literals of a most-specific clause, or "bottom-clause" in ILP. Given a data instance, the bottom-clause is the most-specific first-order clause that entails the data instance, given some domain-knowledge. Loosely speaking, the most-specific clause can be thought of "enriching" the data instance with all domain relations that are true, given the data instance. The construction of such most-specific clauses and their subsequent use in ILP was introduced in [Mug95]. CILP++ [FZG14] uses bottom-clauses for data instances to construct feature-vectors for neural networks. This is an extension to CIL^{2}P in which the neural network uses recurrent connections to enforce the background knowledge during the training [GZ99].

Propositionalisation has conceptual and practical limitations. Conceptually, there is no variable-sharing between two or more first-order logic features. That is, all useful compositions have to be pre-specified. Practically, this makes the space of possible features extremely large: this has meant that the feature-selection has usually been done separately from the construction of the neural network. In this context, another work that does not employ either propositionalisation or network augmentation considers a combination of symbolic knowledge represented in first-order logic with matrix factorization techniques [RSR15]. This exploits dependencies between textual patterns to generalise to new relations.

Recent work on neural-guided program synthesis also explicitly includes domain-specific relations. Here programs attempt to construct automatically compositions of functional primitives. The primitives are represented as fragments of functional programs that are expected to be relevant. An example of neural-guided program synthesis that uses such domain-primitives is DreamCoder [EMM$^+$18, EWN$^+$20]. DreamCoder receives as inputs the partial specification of a function in the form of some inputs–output pairs, and a set of low-level primitives represented in a declarative language. Higher-level domain-concepts are then abduced as compositions of these primitives via a neurally-guided search procedure based on a version of the Bayesian "wake-sleep" algorithm [HDFN95]. The deep neural networks use a (multi-hot) Boolean-vector encoding to represent functional compositions (a binary digit is associated with each primitive function, and takes the value 1 if and only if the primitive is used in the composite function).

There are some methods that do not use an explicit propositionalisation step, nevertheless, amount to re-formulating the input feature representation. In the area of "domain-adaptation" [BDBC$^+$10], "source" problems act as a proxy for domain-knowledge

for a related "target" problem.[1] There is a form of domain-adaptation in which the target's input representation is changed based on the source model. In [DQW+21], for example, a feature-encoder ensures that the feature representation for the target domain that is the same as the one used for the source.

2.2.2 Binary and n-ary Relations

An influential form of representing relational domain-knowledge takes the form *knowledge graph*, which are labelled graphs, with vertices representing entities and edges representing binary relations between entities. A knowledge graph provides a structured representation for knowledge that is accessible to both humans and machines [PSS20]. Knowledge graphs have been used successfully in variety of problems arising in information processing domains such as search, recommendation, summarisation [SPG19]. Sometimes the formal semantics of knowledge graphs such as domain ontologies are used as sources for external domain-knowledge [YLD+21]. We refer the reader to [HBC+20] to a comprehensive survey of this form of representation for domain-knowledge.

Incorporation of the information in a knowledge-graph into deep neural models–termed "knowledge-infused learning"–is described in [KGS19, SGKW19]. This aims to incorporate binary relations contained in application-independent sources (like DBPedia, Yago, WikiData) and application-specific sources (like SNOMED-CT, DataMed). The work examines techniques for incorporating relations at various layers of deep neural networks (the authors categorise these as "shallow", "semi-deep" and "deep" infusion). In the case of shallow infusion, both the external knowledge and the method of knowledge infusion are shallow, utilising syntactic and lexical knowledge in word embedding models. In semi-deep infusion, external knowledge is involved through attention mechanisms or learnable knowledge constraints acting as a sentinel to guide model learning. Deep infusion employs a stratified representation of knowledge representing different levels of abstractions in different layers of a deep learning model to transfer the knowledge that aligns with the corresponding layer in the learning process. Fusing the information in a knowledge-graph in this way into various level of hidden representations in a deep neural network could also allow quantitative and qualitative assessment of its functioning, leading to knowledge-infused interpretability [GFS21].

There have been some recent advances in introducing external domain-knowledge into deep sequence models. For instance, in [YLD+21], the authors incorporate domain-specific knowledge into the popular deep learning framework, BERT [DCLT19] via a

[1] Superficially, this is also the setting underlying *transfer* learning. However, the principal difference is that source and target problems are closely related to domain-adaptation, but this need not be the case with transfer learning. Transfer-learning also usually involves changes in both model-parameters and model-structure. Domain-adaptation does not change the model-structure: we consider these points in a later section.

declarative knowledge source like drug-abuse ontology. The model constructed here, called Gated-K-BERT, is used jointly for extracting entities and their relationship from tweets by introducing the domain-knowledge using an entity position-aware module into the primary BERT architecture. The experimental results demonstrate that incorporating domain-knowledge in this manner leads to better relation extraction as compared to the state-of-the-art. This work could fall within the category of semi-deep infusion as described in [KGS19]. [YZQ+19], in their study on learning from electronic health records show that the adjacency information in a medical knowledge graph can be used to model the attention mechanism in an LSTM-based RNN with attention. Whenever the RNN gets an entity (a medical event) as an input, the corresponding sub-graph in the medical knowledge graph (consisting of relations such as *causes* and *is-caused-by*) is then used to compute an attention score. This method of incorporating the medical relations into the RNN falls under the category of semi-deep knowledge infusion. While the above methods use the relational knowledge from a knowledge-graph by altering or adding an attention module within the deep sequence model, a recent method called KRISP [MCP+21] introduces such external knowledge at the output (prediction) layer of BERT. This work could be considered under the category of shallow infusion of domain-knowledge as characterised by [KGS19].

Knowledge graphs can be used directly by specialised deep neural network models that can handle graph-based data as input (graph neural networks, or GNNs). The computational machinery available in GNN then aggregates and combines the information available in the knowledge graph (an example of this kind of aggregation and pooling of relational information is in [SKB+18]). The final collected information from this computation could be used for further predictions. Some recent works are in [PKD+19, WZX+19], where a GNN is used for estimation of node importance in a knowledge-graph. The intuition is that the nodes (in a problem involving recommender systems, as in [WZX+19], a node represents an entity) in the knowledge-graph can be represented with an aggregated neighbourhood information with bias while adopting the central idea of aggregate-and-combine in GNNs. The idea of encoding a knowledge graph directly for a GNN is also used in Knowledge-Based Recommender Dialog (KBRD) framework developed for recommender systems [CLZ+19]. In this work, the authors treat an external knowledge graph, such as DBpedia [LIJ+15], as a source of domain-knowledge allowing entities to be enriched with this knowledge. The authors found that the introduction of such knowledge in the form of a knowledge-graph can strengthen the recommendation performance significantly and can enhance the consistency and diversity of the generated dialogues. In KRISP [MCP+21], a knowledge-graph is treated as an input for a GNN where each node of the graph corresponds to one specific domain-concept in the knowledge graph. This idea is a consequence of how a GNN operates: it can form more complex domain-concepts by propagating information of the basic domain-concepts along the edges in

the knowledge-graph. Further, the authors allow the network parameters to be shared across the domain-concepts with a hope to achieve better generalisation. We note that while knowledge-graphs provide an explicit representation of domain-knowledge in the data, some problems contain domain-knowledge implicitly through an inherent topological structure (like a communication network). Clearly, GNNs can accommodate such topological structure just in the same manner as any other form of graph-based relations (see for example: [ZWQ+19]).

2.3 Transforming the Loss Function

One standard way of incorporating domain-knowledge into a deep neural network is by introducing "penalty" terms into the loss (or utility) function that reflect constraints imposed by domain-knowledge. The optimiser used for model construction then minimises the overall loss that includes the penalty terms. Figure 2.5 shows a simple block diagram where a new loss term is introduced based on the background knowledge. We distinguish two kinds of domain constraints–syntactic and semantic–and describe how these have been used to introduce penalty terms into the loss function.

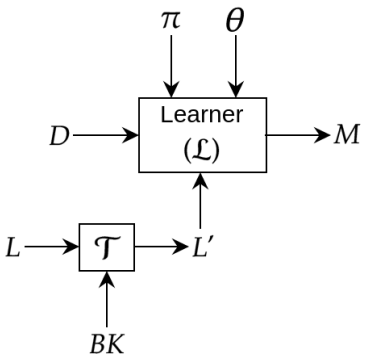


Figure 2.5: Introducing background knowledge into deep neural network by transforming the loss function L. $\mathcal{T}$ block takes an input loss L and outputs a new loss function L' by transforming (augmenting or modifying) L based on background knowledge (BK). The learner $\mathcal{L}$ then constructs a deep model using the original data D and the new loss function L'.

2.3.1 Syntactic Loss

The usual mechanism for introducing syntactic constraints is to introduce one or more *regularisation* terms into the loss function. The most common form of regularisation introduces penalties based on model complexity (number of hidden layers or number of parameters and so on: see, for example, [KGC17]).

A more elaborate form of syntactic constraints involves the concept of *embeddings*. Embeddings refer to the relatively low-dimensional learned continuous vector represen-

tations of discrete variables. Penalty terms based on "regularising embeddings" are used to encode syntactic constraints on the complexity of embeddings. [Fu95] was an early work in this direction, in which the authors proposed a strategy to establish constraints by designating each node in a Hopfield Net to represent a concept and edges to represent their relationships and learn these nets by finding the solution which maximises the greatest number of these constraints. [RBSR14] was perhaps the first method of regularising embeddings from declarative knowledge encoded in first-order logic. The proposal here is for mapping between logical statements and their embeddings, and logical inferences and matrix operations. That is, the model behaves as if it is following a complex first-order reasoning process, but operates at the level of simple vectors and matrix representations. [RSR15] extended this to regularisation by addition of differentiable loss terms to the objective-based on propositionalisation of each first-order predicate. Guo *et al.* [GWW+16] proposed a joint model, called KALE, which embeds facts from knowledge-graphs and logical rules, simultaneously. Here, the facts are represented as ground atoms with a calculated truth value in $[0, 1]$ suggesting how likely the fact holds. Logical rules (in grounded form) are then interpreted as complex formulae constructed by combining ground atoms with logical connectives, which are then modelled by fuzzy t-norm operators [Háj13]. The truth value that results from this operation is nothing but a composition of the constituent ground atoms, allowing the facts from the knowledge graph to be incorporated into the model.

[LS20] develop a method to constraint individual neural layers using soft logic based on massively available declarative rules in ConceptNet. [HBZ+18] incorporates first-order logic into low-dimensional spaces by embedding graphs nodes and represents logical operators as learned geometric relations in the space. [DRR16] proposed ordering of embedding space based on rules mined from WordNet and found it to better prior knowledge and generalisation capabilities using these relational embeddings. [LFJ+18] show that domain-based regularisation in loss function can also help in constructing deep neural networks with less amount of data in prediction problems concerned with cloud computing. In [TA18], a knowledge-based distant regularisation framework was proposed that utilises the distance information encoded in a knowledge-graph. It defines prior distributions of model parameters using knowledge-graph embeddings. They show that this results in an optimisation problem for a regularised factor analysis method.

2.3.2 Semantic Loss

Penalty terms can also be introduced on the extent to which the model's prediction satisfies semantic domain constraints. For example, the domain may impose specific restrictions on the prediction ("output prediction must be in the range $[3, 6]$"). One way in which such information is provided is in the form of domain-constraints. Penalty terms

are then introduced based on the number and importance of such constraints that are violated.

A recent work that is based on the loss function is in [XZF+18]. Here the authors propose a semantic loss that signifies how well the outputs of the deep neural network matches some given constraints encoded as propositional rules. The general intuition behind this idea is that the semantic loss is proportional to a negative logarithm of the probability of generating a state that satisfies the constraint when sampling values according to some probability distribution. This kind of loss function is particularly useful for semi-supervised learning as these losses behave like self-information and are not constructed using explicit labels and can thus utilize unlabelled data. In [THM21], a compositional framework called NeuroLog is proposed that can allow plug-and-play of a neural module and a symbolic module. The domain-constraints are provided to the neural module in an abductive manner. That is, the predictions (or the outputs) from the neural module are used as input for the symbolic module that encodes some logical constraints in first-order logic. If the predictions from the neural module were all correct, then the deduction from the symbolic module would match some desired output; otherwise, the symbolic module will provide an abductive feedback to the neural module via a semantic loss. The loss function dictates how close–semantically–are the outputs of the neural net to the formula found via abduction. The neural module is then trained to reduce this semantic loss.

[HML+16] proposed a framework to incorporate first-order logic rules with the help of an iterative distillation procedure that transfers the structured information of logic rules into the weights of neural networks. This is done via a modification to the knowledge-distillation loss proposed by Hinton et al. [HVD15]. The authors show that taking this loss-based route of integrating rule-based domain-knowledge allows the flexibility of choosing a deep neural network architecture suitable for the intended task.

In [FBDC+19], authors construct a system for training a neural network with domain-knowledge encoded as logical constraints. Here the available constraints are transferred to a loss function. Specifically, each individual logic operation (such as negation, and, or, equality etc.) is translated to a loss term. The final formulation results in an optimisation problem. The authors extract constraints on inputs that capture certain kinds of convex sets and use them as optimisation constraints to make the optimisation tractable. In the developed system, it is also possible to pose queries on the model to find inputs that satisfy a set of constraints. In a similar line, [MIM+19] proposed domain-adapted neural network (DANN) that works with a balanced loss function at the intersection of models based on purely domain-based loss or purely inductive loss. Specifically, they introduce a domain-loss term that requires a functional form of approximation and monotonicity constraints. Without detailing much on the underlying equations, it suffices to say that formulating the domain loss using these constraints enforces the model to learn not only

from training data but also in accordance with certain accepted domain rules.

Another popular approach that treats domain knowledge as 'domain constraints' is semantic based regularisation [DGS17, DRG17]. It builds standard multilayered neural networks (e.g. MLP) with kernel machines at the input layer that deal with continuous-valued features. The outputs of the kernel machines are then input to the higher layers implementing a fuzzy generalisation of the domain constraints that are represented in first-order logic. The regularisation term consisting of a sum of fuzzy generalisation of constraints using t-norm operations in the cumulative loss then penalises each violation of the constraints during the training of the deep neural network. [SLM20] inject domain-knowledge at training time via an approach that combines semantic based regularisation and constraint programming [RVBW06]. This approach uses the concept of 'propagators', which is inherent in constraint programming to identify infeasible assignments of variables to values in the domain of the variables. The role of semantic-based regularisation is to then penalise these infeasible assignments weighted by a penalty parameter. This is an example of constraints on inputs. In a similar line, [LZZ21] introduce domain-knowledge into a deep LSTM-based RNN at three different levels: constraining the inputs by designing a filtering module based on the domain-specific rules, constraining the output by enforcing an output range, and also by introducing a penalty term in the loss function.

A library for integrating symbolic domain-knowledge with deep neural networks was introduced recently in [FGU+21]. It provides some effective ways of specifying domain-knowledge, albeit restricted to (binary) hierarchical concepts only, for problems arising in the domain of natural language processing and some subset of computer vision. The principle of integration involves constraint satisfaction using a primal-dual formulation of the optimisation problem. That is: the goal is to satisfy the maximum number of domain constraints while also minimising the training loss, an approach similar to the idea proposed in [FBDC+19, MIM+19, SLM20].

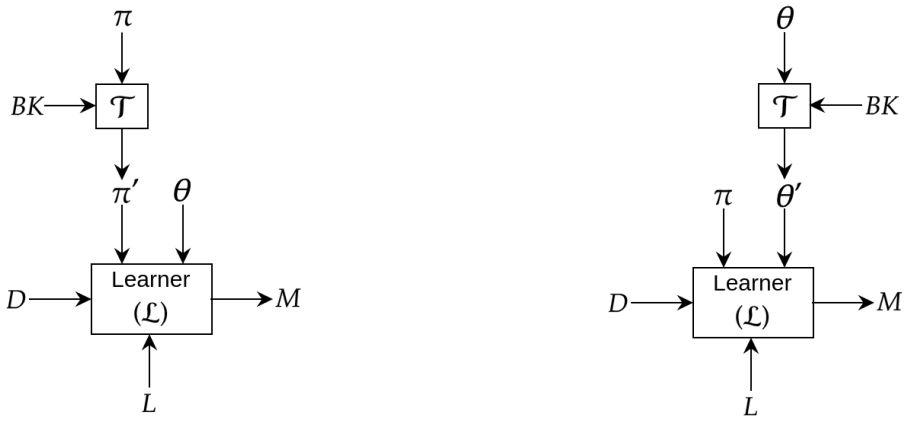
While adding a domain-based constraint term to the loss function may seem appealing, there are a few challenges. One challenge that is pointed out in a recent study [HKBG21] is that incorporating domain-knowledge in this manner (that is: adding a domain-based loss to the standard problem-specific loss) may not always be suitable while dealing with safety-critical domains where 100% constraint satisfaction is desirable. One way to guarantee 100% domain-constraint satisfaction is by directly augmenting the output layer with some transformations and then deriving a new loss function due to these transformations. These transformations are such that they guarantee the output of the network to satisfy the domain constraints. In this study, called MultiplexNet [HKBG21], the domain-knowledge is represented as a logical formula in disjunctive normal form (DNF) Here the output (or prediction) layer of a deep neural network is viewed as a multiplexor in a logical circuit that permits branching in logic. That is, the output of the network always satisfies one of the constraints specified in the domain-knowledge

(disjunctive formula).

The other form of semantic loss could be one that involves a human for post-hoc evaluation of a deep model constructed from a set of first-order rules. In this line, [HH21] proposed an analogical reasoning system intended for discovering rules by training a sequence-to-sequence model using a training set of rules represented in first-order logic. Here the role of domain-knowledge comes post training of the deep sequence model; that is, an evaluator (a human expert) tests each discovered rule from the model by unifying them against the (domain) knowledge base. The domain-knowledge here serves as some kind of a validation set where if the ratio of successful rule unification crosses a certain threshold, then the set of discovered rules are accepted.

2.4 Transforming the Model

Over the years, many studies have shown that domain-knowledge can be incorporated into a deep neural network by introducing constraints on the model parameters (weights) or by making a design choice of its structure. Figure 2.6 shows a simple block diagram showing domain-knowledge incorporation at the design stage of the deep neural network.



(a) Transforming structure

(b) Transforming parameters

Figure 2.6: Introducing background knowledge into deep neural network by transforming the model (structure and parameter). In (a), the transformation block $\mathcal{T}$ takes a input structure of a model π and outputs a transformed structure π' based on background knowledge (BK). In (b), the transformation block $\mathcal{T}$ takes a set of parameters $\boldsymbol{\theta}$ of a model and outputs a transformed set of parameters π' based on background knowledge (BK).

2.4.1 Constraints on Parameters

In a Bayesian formulation, there is an explicit mechanism for the inclusion of domain-knowledge through the use of priors. The regularisation terms in loss-functions, for

example, can be seen as an encoding of such prior information, usually on the network's structure. Priors can also be introduced on the parameters (weights) of a network. Explicitly, these would take the form of a prior distribution over the values of the weights in the network. The priors on networks and network weights represent our expectations about networks before receiving any data, and correspond to penalty terms or regularisers. Buntine and Weigend [BW91] extensively study how Bayesian theory can be highly relevant to the problem of training feedforward neural networks. This work is explicitly concerned with choosing an appropriate network structure and size based on prior domain-knowledge and with selecting a prior for the weights.

The work by [Nea95] on Bayesian learning for neural networks also showed how domain-knowledge could help build a prior probability distribution over neural network parameters. In this, the shown priors allow networks to be "self-regularised" to not overfit even when the complexity of the neural network is increased. In a similar spirit, [KT07] showed how prior domain knowledge could be used to define 'meta-features' that can aid in defining the prior distribution of weights. These meta-features are additional information about each of the features in the available data. For instance, for an image recognition task, the meta-feature could be the relative position of a pixel (x, y) in the image. This meta information can be used to construct a prior over the weights for the original features.

There have been some studies that involve representing domain-knowledge as qualitative statements, called monotonicity constraints [ARD05]. These studies have focused primarily on estimating parameters for Bayesian networks [YN13]. Since monotonic constraints are intuitive to the domain-experts in many real-world scenarios, there have been decent attempts in constructing monotonic deep neural networks. The main idea is again the same as discussed before: Constraining the parameters of the deep neural network to be non-negative by imposing a non-negative activation function at the internal neurons [MZB+21, RS22].

Transfer Learning

Transfer Learning is a mechanism to introduce priors on weights when data is scarce for a problem (usually called the "target" domain). Transfer learning relies on data availability for a problem similar to the target domain (usually called the "source" domain). From the position taken in this chapter, domain-knowledge for transfer learning is used to change the structure or the parameter values (or both) for a model for the target problem. The nature of this domain-knowledge can be seen prior distributions on the structure and/or parameter-values (weights) of models for the target problem. The prior distributions for the target model are obtained from the models constructed for the source problem.

In practice, transfer learning from a source domain to a target domain usually in-

volves a transfer of weights from models constructed for the source domain to the network in the target domain. This has been shown to boost performance significantly. From the Bayesian perspective, transfer learning allows the construction of the prior over the weights of a neural network for the target domain based on the posterior constructed in the source domain. Transfer learning is not limited by the kind of task (such as classification, regression, etc.) but rather by the availability of related problems. Language models are some of the very successful examples of the use of transfer learning, where the models are initially learnt on a huge corpus of data and fine-tuned for other more specialised tasks. [ZQD+20] provides an in-depth review of some of the mechanisms and the strategies of transfer learning. Transfer learning need not be restricted to deep neural networks only: in a recent study, [LWM18] proposes a model that transfers knowledge from a deep neural network to a decision tree using knowledge distillation framework. The symbolic knowledge encoded in the decision tree could further be utilised for a variety of tasks. A recently available learning technique that is closely related to transfer learning is Transformational Machine Learning (TML [OOD+21]), albeit with a major difference: While in transfer learning the parameters of a trained model built for one problem serve as priors for a (related) target problem, in TML, the predictions of a (or a set of) model(s) serve as input features for a related target problem. This technique allows indirect inclusion of knowledge from one or multiple problems into a machine learning (including deep neural network) model.

A subcategory of transfer learning is one in which the problem (or task) remains the same, but there is a change in the distribution over the input data from the source and the target. This form of learning is viewed as an instance of domain-adaptation [BDBC+10]. Similar to transfer learning, the knowledge is transferred from a source domain to a target domain in the form of a prior distribution over the model parameters. This form of domain-adaptation uses the same model structure as the source, along with an initial set of parameter values obtained from the source model. The parameter values are then fine-tuned using labelled and unlabelled data from the target data [THDS15]. An example of this kind of learning is in [DSW+20] where a BERT model is fine-tuned with data from multiple domains. There are some recent surveys along these lines: [WD18, RP20].

2.4.2 Specialised Structures

DNN-based methods arguably work best if the domain-knowledge is used to inspire their architecture choices [BGKP21]. There are reports on incorporating first-order logic constructs into the structure of the network. This allows neural networks to operate directly on the logical sentences comprising domain-knowledge.

Domain-knowledge encoded as a set of propositional rules is used to constrain the structure of the neural network. Parameter-learning (updating of the network weights)

then proceeds as normal, using the structure. The result could be thought of as learning weighted forms of the antecedents present in the rules. The most popular and oldest work along this line is Knowledge-Based Artificial Neural Network (KBANN) [TSN90] that incorporates knowledge into neural networks. In KBANN, the domain-knowledge is represented as a set of hierarchically structured propositional rules that directly determines a fixed topological structure of a neural network [TS94]. KBANN was successful in many real-world applications; but, its representational power was bounded by pre-existing set of rules which restricted it to refine these existing rules rather than discovering new rules. A similar study is KBCNN [Fu93], which first identifies and links domain attributes and concepts consistent with initial domain-knowledge. Further, KBCNN introduces additional hidden units into the network, and most importantly, it allowed decoding of the learned rules from the network in symbolic form. However, both KBANN and KBCNN were not appropriate for learning new rules because of the way the initial structure was constructed using the initial domain-knowledge base.

Some of the limitations described above could be overcome with the proposal of a hybrid system by Fletcher and Obradovic [FO93]. The system was able to learn a neural network structure that could construct new rules from an initial set of rules. Here, the domain-knowledge is transformed into an initial network through an extended version of KBANN's symbolic knowledge encoding. It performed incremental hidden unit generation, thereby allowing construction or extension of the initial rule-base. In a similar manner, there was a proposal for using Cascade ARTMAP [Tan97] which could not only construct a neural network structure from rules but also perform explicit cascading of rules and multistep inferencing. It was found that the rules extracted from Cascade ARTMAP are more accurate and much cleaner than the rules extracted from KBANN [TS93].

In the late 1990s, Garcez and Zaverucha proposed a massively parallel computational model called CIL^{2}P based on feedforward neural network that integrates inductive learning from examples and domain-knowledge, expressed as a propositional logic program [GZ99]. A translation algorithm generates a neural network. Unlike KBANN, the approach uses the notion of "bipolar semi-linear" neurons. This allows the proof of a form of correctness, showing the existence of a neural-network structure that can compute the logical consequences of the domain-knowledge. The output of such a network, when combined into subsequent processing naturally incorporates the intended interpretation of the domain predicates. The authors extend this to the use of first-order logic programs: we have already considered this in section 2.2.

A recent proposal called LENSR [XXK+19] focuses on embedding symbolic knowledge expressed as logical rules. It considers two languages of representation: Conjunctive Normal Form (CNF) and decision-Deterministic Decomposable Negation Normal form (d-DNNF), which can naturally be represented as graph structures. The graph structures can be provided to a graph neural network (GNN) to learn an embedding suitable for

further task-specific implementations.

Somewhat in a similar vein to the work by [GZ99], the work reported in [XZF+18] considers as a set of propositional statements representing domain constraints. A deep neural network is then trained to find satisfying assignments for the constraints. Again, once such a network is constructed, it can clearly be used in subsequent processing, capturing the effect of the domain constraints. The network is trained using a semantic loss that we have described in subsection 2.3.2.

In [LS20], it is proposed to augment a language model that uses a deep net architecture with additional statements in first-order logic. Thus, given domain-knowledge encoded as first-order relations, connections are introduced into the network based on the logical constraints enforced by the domain-relations. The approach is related somewhat to the work in [SAZ+18] that does not explicitly consider the incorporation of domain-knowledge but does constrain a deep neural network's structure by first grounding a set of weighted first-order definite clauses and then turning them into propositional programs.

We note that newer areas are emerging that use representations for domain-knowledge that go beyond first-order logic relations. This includes probabilistic first-order logic as a way of including uncertain domain-knowledge [MDK+18]. One interesting way this is being used is to constrain the training of "neural predicates", which represent probabilistic relations that are implemented by neural networks, and the framework can be trained in an end-to-end fashion [MDK+18, WMMR21]. In DeepProbLog [MDK+18], for example, high-level logical reasoning can be combined with the sub-symbolic discriminative power of deep networks. For instance, a logic program for adding two digits and producing the output sum is straightforward. However, what if the inputs are images of the corresponding digits? Here, a deep neural network is used to map an image to a digit, while a (weighted) logic program, written in ProbLog [DRKT07], for the addition operation, is treated as the symbolic domain-knowledge. The ProbLog program is extended with a set of ground neural predicates for which the weights correspond to the probability distribution of classes of digits $(0, \ldots, 9)$. The parameters (weights of predicates and weights of neural network) are learned in an end-to-end fashion. A recent approach called DeepStochLog [WMMR21] is a framework that extends the idea of neural predicates in DeepProbLog to definite clause grammars [PW80]. The reader may note that although DeepProbLog and DeepStochLog do not really transform the structure of the deep neural network, we are still considering these methods under the heading of specialised structures. This is because of the fact that the hybrid architecture is a tightly coupled approach combining probabilistic logic and deep neural networks.

One of the approaches involves transformation of a probabilistic logic program to graph-structured representation. For instance, in kLog [FCDRDG14] the transformed representation is an undirected bipartite graph in the form of 'Probabilistic Entity-Relationship model' [HMK07] which allows the use of a graph-kernel [VSKB10] for data

classification purpose, where each data instance is represented as a logic program constructed from data and background-knowledge. Another approach uses weighted logic programs or *templates* with GNNs [ŠŽK21] demonstrating how simple relational logic programs can capture advanced graph convolution operations in a tightly integrated manner. However, it requires the use of a language of Lifted Relational Neural Networks (LRNNs) [SAZ+18]. Template-based construction of deep neural network structure can be also seen in Logical Neural Networks (LNNs: [RGL+20]). LNNs resemble a tree structure where the leaf nodes represent the facts in the data and background knowledge, the internal nodes implement logical connectives (and, or, implication, etc.) using t-norm operators derived from real-valued logic, and the outputs represent the rules [SdCRG21]. The inputs to LNNs are instantiated facts (Boolean), and the network is trained by minimising a constrained loss function which is a consequence of such a specialised structure.

An interesting proposal is to transform facts and rules, all represented in (weighted) first-order logic into matrix (or tensor) representations. Learning and inference can then be conducted on these matrices (or tensors) [SG16, CYM20] allowing faster computation. NeuralLog [GC21], for example, extends this idea and constructs a multilayered neural network, to some extent, similar to the ones in LRNN consisting of fact layer, rule layer and literal layer etc. The learning here refers to the updates of the weights of the rules. Another work that translates domain-knowledge in first-order logic into a deep neural network architecture consisting of the input layer (grounded atoms), propositional layer, quantifier layer and output layer is [DGS17]. Similar to LRNN, it uses the fuzzy t-norm operators for translating logical OR and AND operations.

Further emerging areas look forward to providing domain-knowledge as higher-order logic templates (or "meta-rules": see [CDM20] for pointers to this area). To the best of our knowledge, there are, as yet, no reports in the literature on how such higher-order statements can be incorporated into deep neural networks.

2.5 Summary of the Review

Our primary discussion in this review is about the techniques for inclusion of some form of domain-knowledge into deep neural networks. We classify these techniques of into 3 broad categories where the domain-knowledge (1) changes the input data representation, (2) changes the loss function used for training the deep neural network, and (3) changes the parameters or structure of the deep neural network. Under each of these categories, we studied several techniques of inclusion of domain-knowledge. We summarise these techniques in Figure 2.7. We discuss some practical challenges associated with these techniques next.

We first outline some practical challenges in incorporating domain-knowledge encoded as logical or numerical constraints into a deep neural network. Below, we outline some

Principal Approach	Work (Reference)	Type of Learner
Transforming Data	DRM [Lod13]	MLP
	CILP++ [FZG14]	MLP
	R-GCN [SKB+18]	GNN
	KGCN [WZX+19]	GNN
	KBRD [CLZ+19]	GNN
	DG-RNN [YZQ+19]	RNN
	DreamCoder [EWN+20]	DNN*
	Gated-K-BERT [YLD+21]	Transformer
	KRISP [MCP+21]	GNN, Transformer
Transforming Loss	IPKFL [KT07]	CNN
	ILBKRME [RSR15]	MLP
	HDNNLR [HML+16]	CNN, RNN
	SBR [DGS17]	MLP
	SBR [DRG17]	CNN
	DL2 [FBDC+19]	CNN
	Semantic Loss [XZF+18]	CNN
	LENSR [XXK+19]	GNN
	DANN [MIM+19]	MLP
	PC-LSTM [LZZ21]	RNN
	DomiKnowS [FGU+21]	DNN*
	MultiplexNet [HKBG21]	MLP, CNN
	Analogy Model [HH21]	RNN
Transforming Model	KBANN [TS94]	MLP
	Cascade-ARTMAP [Tan97]	ARTMAP
	CIL^{2}P [GZ99]	RNN
	DeepProbLog [MDK+18]	CNN
	LRNN [SAZ+18]	MLP
	TensorLog [CYM20]	MLP
	Domain-Aware BERT [DSW+20]	Transformer
	NeuralLog [GC21]	MLP
	DeepStochLog [WMMR21]	DNN*
	LNN [SdCRG21]	MLP

Figure 2.7: Some selected works, in no particular order, showing the principal approach of domain-knowledge inclusion into deep neural networks. DNN* refers to a DNN structure dependent on intended task. We use 'MLP' here to represent any neural network, that conforms to a layered-structure that may or may not be fully-connected. RNN also refers to sequence models constructed using Long Short-Term Memory (LSTM) or Gated Recurrent Unit (GRU) cells.

immediate practical challenges concerning the logical constraints:

- There is no standard framework for translating logical constraints to neural networks. While there are simplification methods which first, construct a representation of the logical constraint that a standard deep neural network can consume,

this process has its limitations as described in the relevant section above.

- Logic is not differentiable. This does not allow using standard training of deep neural network using gradient-based methods in an end-to-end fashion. Propagating gradients via logic has now been looked at in [EG18], but the solution is intractable and does not allow day-to-day use.

- Many neural network structures are directed acyclic graphs (DAGs). However, transforming logical formulae directly into neural network structures in the manner described in some of the discussed works can introduce cyclic dependencies, which may need a separate form of translations.

- A deep neural network, with its structure constrained via logical domain-knowledge may not always be scalable to large datasets.

There are also the following practical challenges concerning the numerical constraints:

- We have seen that the numerical constraints are often provided with the help of modification to a loss function. Given some domain-knowledge in a logical representation, constructing a term in loss function is not straightforward.

- Incorporating domain-knowledge via domain-based loss may not be suitable for some safety-critical applications.

- The process of introducing a loss term often results in a difficult optimisation problem (sometimes constrained) to be solved. This may require additional mathematical tools for a solution that can be implemented practically.

Furthermore, in this review, we have not discussed how the domain-knowledge is to be acquired from domain experts.