Neural Cognitive Architectures for Never-Ending Learning

Author

Emmanouil Antonios Platanios
www.platanios.org
e.a.platanios@cs.cmu.edu

Committee

Tom Mitchell[†]
Eric Horvitz[‡]
Rich Caruana[‡]
Graham Neubig[†]

Abstract

Allen Newell argued that the human mind functions as a single system and proposed the notion of a unified theory of cognition (UTC). Most existing work on UTCs has focused on symbolic approaches, such as the Soar architecture (Laird, 2012) and the ACT-R (Anderson et al., 2004) system. However, such approaches limit a system's ability to perceive information of arbitrary modalities, require a significant amount of human input, and are restrictive in terms of the learning mechanisms they support (supervised learning, semi-supervised learning, reinforcement learning, etc.). For this reason, researchers in machine learning have recently shifted their focus towards subsymbolic processing with methods such as deep learning. Deep learning systems have become a standard for solving prediction problems in multiple application areas including computer vision, natural language processing, and robotics. However, many real-world problems require integrating multiple, distinct modalities of information (e.g., image, audio, language, etc.) in ways that machine learning models cannot currently handle well. Moreover, most deep learning approaches are not able to utilize information learned from solving one problem to directly help in solving another. They are also not capable of *never-ending learning*, failing on problems that are dynamic, ever-changing, and not fixed a priori, which is true of problems in the real world due to the dynamicity of nature. In this thesis, we aim to bridge the gap between UTCs, deep learning, and never-ending learning. To that end, we propose a *neural cognitive architecture* (NCA) that is inspired by human cognition and that can learn to continuously solve multiple problems that can grow in number over time, across multiple distinct perception and action modalities, and from multiple noisy sources of supervision combined with self-supervision. Furthermore, its experience from learning to solve past problems can be leveraged to learn to solve future ones. The problems the proposed NCA is learning to solve are ever-evolving and can also be automatically generated by the system itself. In our NCA, reasoning is performed recursively in a subsymbolic latent space that is shared across all problems and modalities. The goal of this architecture is to take us a step closer towards general learning and intelligence. We have also designed, implemented, and plan to extend an artificial simulated world that allows us to test for all the aforementioned properties of the proposed architecture, in a controllable manner. We propose to perform multiple case studies—within this simulated world and with real-world applications—that will allow us to evaluate our architecture.

1 Introduction

Cognitive architectures were first introduced by Newell (1990) who argued that the human mind functions as a single system, and proposed the notion of a *unified theory of cognition* (UTC). They often consist of constructs that reflect assumptions about human cognition and that are based on facts derived from psychology experiments (e.g., problem solving, decision making, routine action, memory, learning, skill, perception, motor behavior, language, motivation, emotion, imagination, and dreaming). In fact, Newell believed that cognitive architectures are the way to answer one of the ultimate scientific questions: "How can the human mind occur in the physical universe?". Most existing work on UTCs has focused on symbolic approaches, such as the Soar architecture (Laird, 2012) and the ACT-R (Anderson et al., 2004) system. However, such approaches limit a system's ability to perceive information of arbitrary modalities, require a significant amount of human input, and are restrictive in terms of the learning mechanisms they support (supervised learning, semi-supervised learning, reinforcement learning, etc.). For this reason, researchers in machine learning (ML) have shifted their focus towards methods like deep learning.

Deep learning systems have become the de facto standard for solving prediction problems in a multitude of application areas including computer vision, natural language processing, and robotics. Driven by progress in deep learning, the machine learning community is now able to tackle increasingly more complex problems—ranging from multi-modal reasoning (Hu et al., 2017) to dexterous robotic manipulation (OpenAI et al., 2018)—many of which typically involve solving combinations of tasks. However, many real-world problems require integrating multiple, distinct modalities of information (e.g., image, audio, language) in ways that machine learning models cannot currently handle well. Furthermore, most of these approaches are also not able to utilize information learned from solving one problem to directly help in solving another—something at which human intelligence excels. There have been some limited attempts to train a single model to solve multiple problems jointly (e.g., Kaiser et al., 2017), but the resulting systems generally underperform those trained separately for each problem. Moreover, most of the existing approaches are also not capable of *never-ending learning (NEL)*; namely a machine learning paradigm in which an algorithm learns from examples continuously over time, in a largely self-supervised fashion, where its ex-

[†]Carnegie Mellon University.
[‡]Microsoft Research.

perience from past examples can be leveraged to learn future examples (Mitchell et al., 2018). Current ML systems fail when the problems that need to be learned are not fixed a priori, but are rather dynamic and keep changing as part of the environment where the learning agents operate. For example, humans do not just learn to solve a fixed set of problems, but they rather adapt and by solving one problem, they become better able to tackle new problems that they may even have been previously unaware of[1]. Furthermore, humans are capable of creating problems to learn, on their own, something that current ML systems are not designed to achieve. Never-ending learning is thus also something at which human intelligence excels. To achieve true intelligence, a learning agent that interacts with the real world needs to be able to adapt in such a continuous fashion (i.e., due to the real world's dynamic nature). In fact, such an ability is crucial for never-ending learning, because learning *forever* only really makes sense if the learning objectives are ever-evolving.

We aim to bridge the gap between UTCs, deep learning, and never-ending learning. To that end, we propose a *neural cognitive architecture* that allows for a tighter coupling between problems, as well as a higher-level of abstraction over distinct modalities of information. We thus aim to test the following hypothesis in this thesis:

A computer system with an architecture inspired by human cognition can learn to continuously solve multiple problems that can grow in number over time, across multiple distinct perception and action modalities, and from multiple noisy sources of supervision combined with self-supervision. Furthermore, its experience from learning to solve past problems can be leveraged to learn to solve future ones.

Our main goals can be summarized as follows:

- **Formalizing** never-ending learning and the notion of a neural cognitive architecture. This includes defining the notion of an ever-evolving set of learning problems, whether the problems are provided externally or generated by the learning system itself, as well as ways to handle this setting.
- **Designing** a neural cognitive architecture that is inspired from the *Hub-and-Spoke* model of human cognition (Rogers et al., 2004; Ralph et al., 2017) and that also accounts for human *goal-priming* (Custers and Aarts, 2005; Aarts et al., 2008; Takarada and Nozaki, 2018). It is a novel modular architecture that contains perception and action spokes (i.e., modules), and a common reasoning hub for all problems, that is independent of data modalities. The reasoning hub enables human-inspired capabilities such as *associative memory* (Fanselow and Poulos, 2005; Ranganath and Ritchey, 2012) and *world simulation*. It makes use of *contextual parameter generation* (Platanios et al., 2018) to emulate goal-priming.
- **Evaluating** the capabilities of the proposed architecture using multiple case studies over different learning settings. One such setting is the artificial *Jelly Bean World* that we have created, and where we can control the kinds of problems the agent needs to solve, and their interactions. We have designed this world in a way that renders never-ending learning necessary, and plan to extend it so that it allows us to test all parts of our hypothesis, in a controllable manner. After testing our hypothesis in this artificial world, we also plan to perform experiments on real world problems, related to natural language processing, computer vision, and potentially healthcare. Healthcare applications are interesting because they present a real-world setting where such an architecture would be useful. This is due to the low amount of training data and large number of interconnected problems that underlie many healthcare applications.

This proposal is meant to describe our way of thinking about the design space for this problem as a whole. We are proposing to make progress towards confirming and exploring the aforementioned thesis statement, rather than being exhaustive. In the following section we discuss our main motivation for this thesis. Then, in Section 3 we describe the proposed approach along with background and related work for each of its components, and in Section 4 we describe our planned evaluation case studies. Finally, in Section 5 we present a tentative timeline for the proposed work.

2 Motivation

A long-standing goal in the fields of artificial intelligence and machine learning is to develop algorithms that can be applied across domains and that can efficiently handle multiple problems, just like the human mind does. Even though research in multi-task learning has a long history (Caruana, 1997), there has been a resurgence of interest in fundamental questions related to: (i) algorithmic frameworks for multi-task learning, such as learning-to-learn or meta-learning (Thrun and Pratt, 1998; Finn et al., 2017; Franceschi et al., 2018) and never-ending/lifelong learning (Mitchell et al., 2018), (ii) establishing best practices for building reliable systems that can handle multiple tasks at scale, such as federated learning for model personalization (Smith et al., 2017) or multi-agent coordination (Cao et al., 2013; Samarakoon et al., 2018), and (iii) learning deep representations (Bengio et al., 2013) that support multi-tasking and enable transfer learning in multiple domains, such as computer vision (Yosinski et al., 2014) or natural language processing (Collobert and Weston, 2008; Peters et al., 2018; Devlin et al., 2018).

Our interest in these questions started while working on the Never-Ending Language Learner (NELL) (Mitchell et al., 2018). NELL is a system that learns to read the web and extract knowledge from websites, in a never-ending fashion. One of the core mechanisms employed in NELL is *co-training*, which was originally proposed by Blum and Mitchell (1998). Co-training is a semi-supervised learning algorithm where multiple models are being trained together and each model can use as training examples the most confident predictions made by the other models. If any of the

[1] For example, after humans managed to build heart monitoring devices, new unsolved problems became available, such as discovering the relationship between heart rate or blood pressure and specific health problems.

models produces wrong but confident predictions, these can propagate to the other models and eventually hinder learning. This motivated us to develop several algorithms for estimating accuracies of classifiers from unlabeled data (Platanios et al., 2014; 2016; 2017). The key idea behind all these methods is that agreement among multiple models implies that the agreed upon prediction is more likely correct than wrong. However, we also observed that once we have multiple interacting tasks that are being learned jointly, we can perform accuracy estimation in a more robust manner by also accounting for inconsistencies between the tasks. For example, if one classifier predicts that `Pittsburgh` is a city and another one predicts that it is a person, and we know that something cannot be both a city and a person at the same time, then we can infer that at least one of these two classifiers must be wrong. Finally, this work pointed out an important pattern in how current machine learning systems are trained. Training data is often obtained by collecting multiple noisy labels for samples through crowdsourcing that are then aggregated to produce a single "denoised" label per sample. To this end, we adapted our accuracy estimation methods resulting in a learning framework for general machine learning systems that allows them to be trained from multiple noisy labels directly—without requiring an explicit label aggregation step (Platanios et al., 2019). Through this and other experiences from working in NELL, we observed that: (i) learning multiple tasks jointly while also accounting for their interactions, and (ii) learning from multiple noisy sources of supervision, are both crucial to building successful NEL systems.

3 Approach

We structure the proposed work in four main parts:

1. **Learning from Multiple Noisy Labels:** Mechanisms for learning from multiple noisy sources (e.g., obtained using a crowdsourcing platform), including self-supervision.
2. **Contextual Parameter Generation:** Methods that enhance the model capacity of neural networks allowing them to learn functions that are conditioned on some of their inputs (i.e., the context), thus enabling more effective multi-task learning architectures.
3. **Self-Reflection:** Mechanisms that allow a system to self-evaluate and improve without external supervision. This is an important property for never-ending learning systems as the extent of external supervision is often limited, but the system needs to keep learning.
4. **Unified Architecture:** A unified neural cognitive architecture that puts together all aforementioned components, along with several new ones, and is able to perform large scale multi-modal and multi-task learning.

3.1 Learning from Multiple Noisy Labels

Machine learning systems often rely on large amounts of annotated examples to be trained. This is especially true for never-ending learning systems. Perhaps the most common way to collect such training examples is using noisy crowdsourcing platforms like Amazon Mechanical Turk (AMT). Practitioners typically adopt the following process: (i) collect multiple annotations per example in order to reduce the amount of noise, (ii) aggregate these annotations into a single label per example that represents an estimate of the ground truth (e.g., using majority voting), and (iii) train machine learning systems using the resulting labeled examples. This results in both redundant annotations and potentially noisy ground truth labels. We propose a novel approach that enables us to merge the steps of aggregating noisy annotations and training machine learning systems, by allowing a system to be trained directly from multiple noisy annotations. Our approach also learns models of the difficulty of each example and the competence of each annotator in a generalizable manner (i.e., these models can make predictions for previously unseen examples and annotators). This enables us to more optimally assign annotators to examples, thus driving the cost of crowdsourcing down, while improving the quality of the resulting datasets. Our approach can also be used to perform ensemble learning and to estimate the accuracies of classifiers from unlabeled data. The latter has become especially relevant with recent advances in weak supervision and self-supervision (e.g., Ratner et al., 2017).

The problems of ensemble learning, aggregating and denoising crowdsourced data, and estimating accuracy from unlabeled data, all share the same underlying core problem: *learning from multiple noisy labels*. More specifically, there is a common setting among all these problems where: (i) there exists an underlying ground truth, (ii) we only get to observe multiple, possibly overlapping, noisy views of that truth, and (iii) we want to be able to estimate that truth. The noisy views can have arbitrary form, such as: (i) human annotators in a crowdsourcing platform, that may make mistakes (e.g., Zhou et al., 2015), or (ii) classifiers that have already been trained (e.g., Platanios et al., 2014; 2016; 2017). To give a concrete example, consider the problem of medical pathology diagnostics, where learning-based models are becoming increasingly popular (e.g., Gulshan et al., 2016). Training models by imitating expert decisions is not as straightforward in such a scenario: the true diagnosis is unknown a priori, while the diagnostic concordance between experts is often far from perfect (Elmore et al., 2015). If we assume that the expert decisions are the ground truth, the model may overfit to their mistakes. Therefore, this practical setup requires a principled learning framework that takes into account potential discrepancies or disagreements in the observations.

3.1.1 Related Work

Learning binary classifiers from examples with noisy labels was first introduced and theoretically characterized by Angluin and Laird (1988). In that work, the noise model was based on independent random flips of the labels with some probability $\eta < 0.5$. Kearns (1998) later characterized a class of robust learning algorithms for such types of label noise. Nettleton et al. (2010) studied empirically the behavior of (at the time) popular learning algorithms under different magnitudes of noise. Natarajan et al. (2013) proposed to modify surrogate loss functions to obtain unbiased estimators and obtained performance bounds for empirical risk minimization

in the presence of noisy labels. More recently, Frénay and Verleysen (2014) surveyed several notable methods and variations of this problem. This whole line of work differs from our setting in that each example only gets a single noisy label. On the contrary, we assume that each example is labeled multiple times using independent labeling processes, which we refer to as *predictors*, each of which is characterized by an unknown confusion matrix.

This problem has also been previously framed as estimating accuracy from unlabeled data, or as aggregating worker predictions in the context of crowdsourcing. Similar settings were previously explored by Collins and Singer (1999), Dasgupta et al. (2001), Bengio and Chapados (2003), Madani et al. (2004), Schuurmans et al. (2006), Balcan et al. (2013), and Parisi et al. (2014), among others. However, none of the previous approaches considers explicitly modeling the ground truth; they rather assume some form of independence or knowledge of the true label distribution. Collins and Huynh (2014) review many methods that were proposed for estimating the accuracy of medical tests in the absence of a gold standard. Previously we proposed formulating the problem as an optimization problem that uses agreement rates between multiple noisy labelers over unlabeled data (Platanios et al., 2014). Dawid and Skene (1979), Moreno et al. (2015), and us (Platanios et al., 2016) have also previously formulated the problem in terms of probabilistic graphical models. Tian and Zhu (2015) proposed a max-margin majority voting scheme applied to crowdsourcing. More recently, we introduced a method that is able to use information provided in the form of logical constraints between the noisy labels (Platanios et al., 2017), and Khetan et al. (2017) proposed using a parametric function to model the ground truth. However, previous approaches were outperformed by Zhou et al. (2015) who formulated the problem as a form of regularized minimax conditional entropy and used their method in crowdsourcing.

Our approach is a generalization of the approaches proposed by Zhou et al. (2015), Platanios et al. (2016), and Khetan et al. (2017). Similar to our prior work (Platanios et al., 2016) we define a generative process for our observations. However, our approach is also able to handle categorical labels, as opposed to just binary labels. Also, similar to Zhou et al. (2015) we define the confusion matrix for each instance-predictor pair as a function of instance difficulty and predictor competence. However, in our approach we explicitly learn the difficulty and competence functions, allowing us to generalize to previously unseen instances and predictors. Interestingly, the inference algorithm for our generative probabilistic model has a similar form to that of Zhou et al. (2015) (except for the explicit learning of a ground truth function, as well as of difficulty and competence functions). In fact, the algorithm of Zhou et al. (2015) can be derived as an Expectation-Maximization (EM) inference algorithm for a generative model, that is a simplified version of the one that we are proposing. Finally, similar to Khetan et al. (2017) we propose to use a parametric function to model the ground truth, but we go a step further and also propose to use parametric functions to model the instance difficulties and predictor competences. Thus, our approach allows us to predict which predictors are likely to perform better for specific instances, enabling us to allocate predictors more optimally and reduce costs.

3.1.2 Proposed Method

Let us denote the observed data by $\mathcal{D} = \{\boldsymbol{x}_i, \hat{\mathcal{Y}}_i\}_{i=1}^N$, where $\hat{\mathcal{Y}}_i = \{\mathcal{M}_i, \{\hat{y}_{ij}\}_{j \in \mathcal{M}_i}\}$, $\mathcal{M}_i$ is the set of predictors that made predictions for instance $\boldsymbol{x}_i$, and $\hat{y}_{ij}$ is the output of predictor $\hat{f}_j$ for instance $\boldsymbol{x}_i$. Our goal is to learn functions representing the underlying ground truth and predictor qualities, given our observations $\mathcal{D}$.

Ground Truth. We define the ground truth as a function $h_\theta(\boldsymbol{x}_i)$ that is parameterized by θ and that approximates the true distribution of the label given $\boldsymbol{x}_i$. In our setting, $h_\theta(\boldsymbol{x}_i) \in \mathbb{R}^C_{\geq 0}$ and $\sum_j [h_\theta(\boldsymbol{x}_i)]_j = 1$, where C is number of values the label can take (i.e., assuming categorical labels). More specifically, $[h_\theta(\boldsymbol{x}_i)]_k \triangleq \mathbb{P}(y_i = k \mid \boldsymbol{x}_i)$, where we use square brackets and subscripts to denote indexing of vectors, matrices, and tensors. For example, h_θ could be a deep neural network that would normally be trained in isolation using the cross-entropy loss function. In our method the network is trained using the Expectation-Maximization algorithm, as described in the next section.

Predictor Qualities. We define the predictor qualities as the confusion matrices $\mathbf{Q}_{ij} \in \mathbb{R}^{C \times C}_{\geq 0}$, for each instance $\boldsymbol{x}_i$ and predictor $\hat{f}_j$, where $\sum_l [\mathbf{Q}_{ij}]_{kl} = 1$, for all $k \in \{1, \ldots, C\}$. $[\mathbf{Q}_{ij}]_{kl}$ represents the probability that predictor $\hat{f}_j$ outputs label l given that the true label of instance $\boldsymbol{x}_i$ is k. We define these confusion matrix in a way that generalizes the successful approach of Zhou et al. (2015)[2]:

$$\mathbf{Q}_{ij} = \mathbf{D}_i \bullet_3 \mathbf{C}_j, \tag{1}$$

where $\bullet_i$ represents an inner product along the i^{th} dimension of the two tensors, and:

- $\mathbf{D}_i = d_\phi(\boldsymbol{x}_i)$ represents the *difficulty* tensor for instance $\boldsymbol{x}_i$, where d is a function parameterized by ϕ, $\mathbf{D}_i \in \mathbb{R}^{C \times C \times L}$, and L is a latent dimension (it is a hyperparameter of our model). $[\mathbf{D}_i]_{kl-}$ is an L-dimensional embedding representing the likelihood of confusing $\boldsymbol{x}_i$ as having label l instead of k, when k is its true label.
- $\mathbf{C}_j = c_\psi(\boldsymbol{r}_j)$ represents the *competence* tensor for predictor $\hat{f}_j$, where c is a function parameterized by ψ, $\boldsymbol{r}_j$ is some representation of $\hat{f}_j$ (e.g., could be a one-hot encoding of the predictor, in the simplest case), and $\mathbf{C}_j \in \mathbb{R}^{C \times C \times L}$. $[\mathbf{C}_j]_{kl-}$ is an L-dimensional embedding representing the likelihood that predictor $\hat{f}_j$ confuses label k for l, when k is the true label.

Using $L > 1$ allows the instance difficulties and predictor competences to encode more information. An intuitive way to think about this is that we are embedding difficulties and

[2] We also perform a normalization step such that all elements of $\mathbf{Q}_{ij}$ are non-negative and such that each row sums to 1 (thus making each row a valid probability distribution).

competencies in a common latent space, which can be thought of as jointly clustering them. This is in fact very similar to how matrix factorization methods are used for collaborative filtering in recommender systems.

Our goal is to learn functions h_θ, d_ϕ, and c_ψ, given observations $\mathcal{D}$. To do that, we propose a generative process for our observations. For $i = 1, \ldots, N$, we first sample the true label for $\boldsymbol{x}_i$, $y_i \sim \text{Categorical}(h_\theta(\boldsymbol{x}_i))$. Then, for $j \in \mathcal{M}_i$, we sample the predictor output $\hat{y}_{ij} \sim \text{Categorical}([\mathbf{Q}_{ij}]_{y_i-})$, where $[\mathbf{Q}_{ij}]_{y_i-}$ represents the $y_i{}^{\text{th}}$ row of $\mathbf{Q}_{ij}$. We derive an EM algorithm for performing inference that is presented in (Platanios et al., 2019). The resulting approach can be thought of as introducing a new loss function for training the model h_θ using multiple noisy labels per training instance, each coming from a distinct noisy predictor. This new loss function introduces latent variables representing the ground truth labels, as well as a couple of auxiliary models that are learned, and which represent the instance difficulties and predictor competences. Perhaps most interestingly, a key difference between this approach and previous work is that we are able to explicitly learn functions that output the likelihood that a predictor will label a specific instance correctly. This enables using this method to perform crowdsourcing actively by assigning annotators to instances they are likely to label correctly, thus reducing redundancy and driving costs down.

3.2 Contextual Parameter Generation

In order to present the second major component of the proposed work we need to first provide some background. We refer to parameterized functions as *networks*. We denote a network by a lowercase English letter with a lowercase Greek letter subscript (e.g., f_θ), where the Greek letter refers to the network parameters. Therefore, given some input, x, the output of the network is simply defined as:

$$y = f_\theta(x). \tag{2}$$

Most deep learning models can be seen as networks. For example, we can have a convolutional neural network (CNN) that takes images as input, transforms them using convolutional filters (i.e., parameters), and produces distributions over labels (e.g., `cat` or `dog`). Research in deep learning has resulted in multiple network architectures that can successfully learn to solve various problems, and that each makes different assumptions about its input space. For example, CNNs assume that there is some periodical structure in the input space, whereas recurrent neural networks (RNNs) assume that each part of an input sequence can be processed using the same network parameters.

Never-ending learning requires a system to be able to perform multiple tasks; perhaps even previously unseen tasks that can be formulated in terms of other previously learned tasks. This means that traditional multi-task neural network architectures that use a different output layer for each task (e.g., Caruana, 1997) cannot be used in this context. That is because the set of tasks the system is learning to perform is not known a priori, when the neural network architecture is chosen. This motivates us to treat tasks as separate inputs.

We argue that, for most existing neural network architectures, it is hard or even impossible to encode assumptions about the contexts (e.g., tasks) in which they are used, to share information across these contexts, and to "personalize" them for each context. As we discuss in the end of this section, this limitation could be attributed to the fact that most existing architectures are only able to represent additive interactions between their inputs. Previously, there has been some success in encoding this kind of assumptions using probabilistic graphical models (PGMs). When working with PGMs, researchers typically first define a prior probabilistic model over how the data observations are generated and then perform inference to obtain a posterior distribution over the model parameters and possibly also latent variables. These generative models are often hierarchical, meaning that the parameters of the distribution from which the observations are sampled, are also often sampled themselves from a higher-level distribution. This results in an interesting type of information sharing across all the different distributions, and has been behind many successful models, such as latent Dirichlet allocation (Blei et al., 2003) and hierarchical Dirichlet processes (Teh et al., 2005). There have been efforts to combine such approaches with neural networks (e.g., Tran et al., 2018), but they are often expensive and impractical for large scale problems. Furthermore, in order to make probabilistic inference tractable they often limit model expressivity.

This motivated us to develop a method called *contextual parameter generation* (CPG) (Platanios et al., 2018). The core idea behind this method is that, given a network, f_θ, instead of learning θ directly while training, we define it as:

$$\theta = g_\phi(c), \tag{3}$$

where c is a description of the context in which we are applying the model (for example, if we are encoding text written in English as part of a multilingual machine translation model, the context could simply be a one-hot encoding of the English language). The parameters we learn during training are just those of g_ϕ, which we refer to as the parameter generation network. This allows us to share information across instances of f_θ used in different contexts. While we previously had to learn and use different parameters for each context in which f_θ is used, they are now all generated as a function of the context. For example, instead of using different encoders for text written in English and text written in German, we can now use one encoder and simply generate its parameters as a function of a language representation. Note that we can simply define g_ϕ as a lookup table over different contexts, and this would reduce to the previous setting in which there is no information sharing. However, the CPG formulation allows us to impose arbitrary information sharing structures by manipulating the functional form of the parameter generation network, g_ϕ. For example, we could learn embeddings for all language families and have all Romance language embeddings be defined as linear transforms of the corresponding Romance family embedding. When performing multi-task learning, we can think of each task as a context in which a network processes its inputs. Given a representation of this

context, we can generate the parameters of a single universal network that is used for all tasks. The way contexts are defined and processed to generate parameters can thus allow for controlled information sharing across multiple tasks. We refer to networks that employ CPG as *contextualized networks*, and we let them optionally have some of their parameters be generated by a CPG component, and some be directly learned (e.g., we may not want to generate the parameters of a batch normalization layer using CPG). Note that contextualized networks also have better generalization properties than plain networks because they can be used with previously unseen contexts, as long as the new contexts can be composed out of previously seen contexts (refer to Section 3 for more details).

Previous Uses of CPG. We first introduced CPG in a multi-task setting, as a means to tackle the multilingual machine translation (MT) problem (Platanios et al., 2018). Multilingual MT is challenging due to the low-resource nature of many languages (i.e., it is hard and often impossible to collect the massive amounts of parallel sentences required for training a neural MT system). It is therefore crucial to share information across languages, rather than train multiple pairwise translation models in isolation. In neural MT, when translating a sentence from `English` to `German`, we first encode the source sentence to some intermediate representation, and then we decode it into some target language. Applying CPG to this problem, we used the source language (i.e., English) as the context of the encoder and the target language (i.e., German) as the context of the decoder. We learned language embeddings and used linear transforms to obtain the encoder/decoder parameters from these embeddings. We were able to show significant performance gains over the same networks without CPG; especially so for the low-resource setting. Furthermore, we showed that CPG allows us to perform zero-shot translation (meaning to translate between pairs of languages that were not observed in the training data), thus indicating that it can be used to generate network parameters for new, previously unseen tasks. We also applied CPG in the problem of question answering over graphs (Platanios* et al., 2019), also known as link prediction. In this case, we are given a source entity (e.g., `Pittsburgh`) and a relation (e.g., `CityInCountry`), and are asked to predict the target entity (e.g., `USA`). Here, different questions correspond to different problems and can be used as the context in which to generate the parameters of a universal question-answering model. We were able to show that CPG outperforms all existing methods and thus establishes a new state-of-the-art for this problem. More recently, we have also started applying CPG to develop better image and video compression methods, and also as a means to handle task composition.

Feeding Context as an Additional Input. Why not just feed the context as another network input?

1. Structured Information Sharing: Similar to the motivation for PGMs described earlier in this section, CPG provides a structured way to share information across contexts.
2. Additive Interactions: Without loss of generality, let us split the input x to a neural network in two parts, x_0 and x_1. For example, assuming x is a vector, then x_0 and x_1 are vectors such that when concatenated, they form x. Most neural network architectures currently in use only allow for interactions of the following form:
$$y = f_\theta(h^0_{\phi_0}(x_0) + h^1_{\phi_1}(x_1)), \tag{4}$$
where f, h^0, and h^1 are arbitrary functions, and y is the output of the neural network. This form is very restrictive. For example, it cannot be used to represent simple `if-then-else` rules such as "`if` $x_0 = 2$`, then` $2x$ `else` $5x_1$". This is especially important for multi-task learning because we can think of x_0 as the description of some task and we might want to condition on that task while processing the rest of the model inputs. This would be the case for a mixture of experts model, for example. In order to represent this kind of interactions we have to explicitly encode them in the neural network architecture. Ideally, we want the model to be able to learn these interactions on its own, if they are necessary, instead of having them be hardcoded as part of the model architecture. CPG in fact allows for multiplicative or even polynomial interactions between $h^0_{\phi_0}(x_0)$ and $h^1_{\phi_1}(x_1)$, which would allows us to represent `if-then-else` rules.
3. Deployment: CPG allows us to generate a context-specific model and later use it without involving the parameter generation network. This can be beneficial for deployment. For example, if we only care about English-to-German translation due to an upcoming vacation trip, we could have Google Translate generate a translation model for this language pair and then store that model on a mobile device for offline use.
4. Modeling Assumptions: Different neural network architectures make different assumptions. For example, CNNs assume spatial invariance for the inputs (meaning that they contain repeating patterns in different locations). This assumption is unlikely to hold for an arbitrary context vector and so it is unreasonable to just concatenate an arbitrary context vector with an image and feed the result to a CNN. CPG avoids this problem because the architecture modification it entails does not affect the assumptions made by a network about the input data.

3.2.1 Related Work

Ha et al. (2018) are probably the first to introduce a similar idea to that of having one network (called a *hypernetwork*) generate the parameters of another. However, in that work, the input to the hypernetwork are structural features of the original network (e.g., layer size and index). Al-Shedivat et al. (2017) also propose a related method where a neural network generates the parameters of a linear model. Their focus is mostly on interpretability (i.e., knowing which features the network considers important). Dumoulin et al. (2018) provide a comprehensive review of some more related work from other fields, such as computer vision, that was published concurrently to our machine translation work. Furthermore, CPG is generic enough so that many existing methods can

be formulated as CPG variants. One such example is model-agnostic meta-learning by Finn et al. (2017), where a model is pre-trained over a large number of tasks and then fine-tuned on new tasks drawn from the same task distribution. In this case, the parameter generation network consists of taking a gradient descent step, using only the new task's data.

3.2.2 Proposed Work

We propose to divide our proposed work on CPG in two parts: (i) understand why CPG works and what the fundamental limitation of neural networks is that CPG tackles, and (ii) develop novel methodology that will allow models to learn what to condition on, rather than learn to condition on a specific pre-specified context.

3.3 Self-Reflection

In order to achieve never-ending learning, a system needs to be able to learn in a largely unsupervised fashion. This requires self-reflective behavior. We propose to introduce a novel mechanism that allows a system to *self-evaluate* when there is no external supervision and to *self-improve* by maximizing its own self-evaluation metric. Some of our prior work discussed in Section 3.1 can be used to self-evaluate, but it is not directly clear how to add a self-improvement mechanism. To this end, we plan to introduce a differentiable *intrinsic reward function* that can be used for both self-evaluation and self-improvement. This is a parametric function that is updated whenever a supervision signal is provided and that is otherwise used directly to perform model updates whenever there is no supervision signal. This is mostly relevant to reward shaping in reinforcement learning and represents a more long-term goal for this thesis. In Section 3.4.7 we propose how to integrate such a mechanism in a unified neural cognitive architecture.

3.4 Unified Architecture

The final part of our work consists of putting all the previously presented pieces together in a single neural cognitive architecture. In order to present that, we first provide some background on cognitive architectures.

3.4.1 Cognitive Architectures

Cognitive architectures can be broadly divided in *symbolic*, *subsymbolic*, and *hybrid* architectures. Symbolic systems rely on sets of rules and reason over discrete spaces (e.g., using first-order logic). Subsymbolic systems specify no such rules a priori and rely instead on emergent properties of several distinct processing units (e.g., neural networks). Hybrid approaches are a combination of the symbolic and subsymbolic approaches. Most past work on UTCs has focused on symbolic systems. In the following paragraphs, we describe two such successful systems.

Soar. Laird (2012) designed Soar, a general cognitive architecture for developing systems that exhibit intelligent behavior, that has been in use since 1983. The design of Soar can be seen as an investigation of an approximation to *complete rationality*, which would imply the ability to use all available knowledge for every task that the system encounters. The primary principle at the base of Soar's design is that "all decisions are made through the combination of relevant knowledge at runtime. In Soar, every decision is based on the current interpretation of sensory data, the contents of working memory created by prior problem solving, and any relevant knowledge retrieved from long-term memory." Soar relies on multiple learning mechanisms (chunking, and reinforcement, episodic, and semantic learning), and on many representations of long-term knowledge (procedural knowledge productions, semantic memory, and episodic memory).

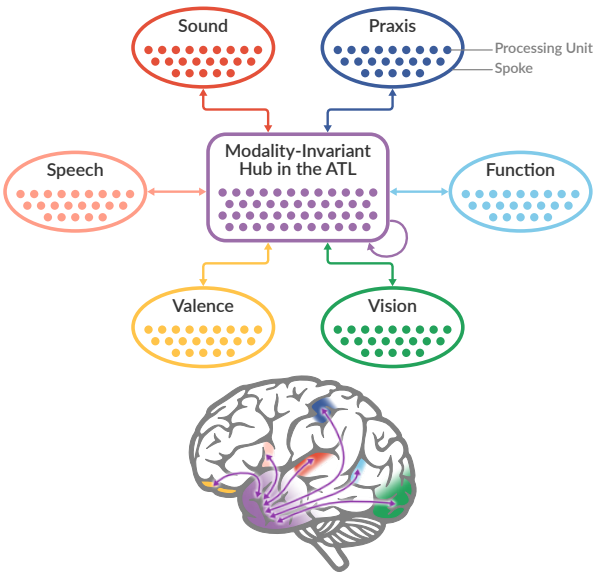

Figure 1: The original Hub-and-Spoke model (Rogers et al., 2004).

ACT-R. Anderson et al. (2004) propose an alternative cognitive architecture, ACT-R, aimed at simulating and understanding human cognition. ACT-R consists of constructs that reflect assumptions about human cognition and that are based on facts derived from psychology experiments. An important feature of ACT-R that distinguishes it from other UTCs is that it directly allows researchers to compare the system's performance to that of human participants.

In this thesis, we propose a novel cognitive architecture that also reflects assumptions about human cognition—inspired from the high-level design of the aforementioned systems—but that is subsymbolic and enables the use of neural networks and end-to-end training. It contains components that correspond to perception, action, reasoning, memory, world simulation, and learning.

3.4.2 The Hub-and-Spoke Theory

Rogers et al. (2004) proposed the *Hub-and-Spoke* theory of human cognition, which assimilates two important ideas: (i) multi-modal experiences provide the main "ingredients" for constructing concepts and they are encoded in modality-specific cortices, or *spokes*, that are distributed across the

brain, and (ii) cross-modal interactions between the modality specific spokes are mediated by a single trans-modal *hub* that is located bilaterally in the anterior temporal lobes (ATLs) of the human brain. A visualization is shown in Figure 1. This model of the human brain serves as one of the main inspirations for the high-level design of the proposed architecture.

3.4.3 Proposed Architecture

We propose a novel *neural cognitive architecture* (NCA) for general learning and intelligence. The proposed architecture is inspired from the *Hub-and-Spoke* model for human cognition (Rogers et al., 2004; Ralph et al., 2017), as well as human *goal priming* (Custers and Aarts, 2005; Aarts et al., 2008; Papies, 2016; Takarada and Nozaki, 2018). It consists of the following parts (an overview is shown in Figure 2):

- Perception and Action Spokes: Sensing input data consists of converting them to a common reasoning space, that is independent of the data modality. Much of the complexity of models like BERT[3] (Devlin et al., 2018), lies in perception, rather than reasoning. In fact, for BERT, reasoning often consists of a single linear layer, while perception consists of a Transformer (Vaswani et al., 2017). Similarly, taking an action consists of converting a common reasoning representation to some output data. This can include taking actions in some environment, or generating data of some structure (e.g., probabilistic distribution over labels).
- Reasoning Hub: Reasoning is performed in a latent space that is independent of the data modalities and the problem being solved. We argue that this is necessary for general learning and intelligence, as it allows for flexible sharing of information across different modalities and problems. Moreover, memory and simulations of the external world are all defined over the same latent space, abstracting away details about the perceived data that are not relevant to reasoning. Reasoning is described in detail in Section 3.4.5.
- Goal Contextualization: The problems that the system is learning to solve are processed such that they can contextualize any part of the neural cognitive architecture. This allows for the behavior of the system to vary across different problems, while still sharing information between them, similar to how it was done for machine translation, as described in Section 3.2. It further allows the system to generate its own target problems that it learns to solve. This is perhaps the most novel aspect of the proposed architecture and, as shown in the following paragraphs, derives its inspiration from human *goal priming* in psychology, and is described in more detail in Section 3.4.6.

This is inspired from work in multiple areas:

Deep Learning. Deep neural networks are very effective at learning abstract representations for arbitrary data modalities, that can then be used to perform multiple diverse tasks (e.g., Simonyan and Zisserman, 2015; He et al., 2016; Peters et al., 2018; Devlin et al., 2018). The typical deep learning workflow is that for each problem researchers build large deep neural models that pool together information from different sources and that are trained independently of each other. An alternative approach is to pre-train large models in a problem-independent manner and then fine-tune them for each problem (e.g., Peters et al., 2018; Devlin et al., 2018; Finn et al., 2017). However, most of these approaches do not allow for information learned from solving one problem to directly help solve another—something at which human intelligence excels. For example, BERT is pre-trained as a language model and is then fine-tuned separately for problems such as question answering and textual entailment. Therefore, learning to answer questions well does not affect how well BERT reasons about textual entailment. This motivates us to find ways to couple the learning of multiple problems in a way that results in *constructive interference* between the different problems, meaning that learning to solve one well, helps the system learn to solve others faster. It further motivates us to treat *perception* (i.e., learning informative representations of the input data) and *reasoning* (i.e., learning to solve each task in the latent space of learned representations) separately, as most deep neural networks that are trained end-to-end to solve multiple tasks effectively do that, and most of their complexity is often related to perception (e.g., in BERT problem-specific reasoning is performed by a linear layer).

Kernel Methods. Before deep learning was popular, some of the most successful machine learning methods were making use of *kernels* (Hofmann et al., 2008), by formulating learning problems in a reproducing kernel Hilbert space (RKHS) of functions defined on the data domain, expanded in terms of a kernel. These kernels effectively project the data to a space where reasoning is modeled as a linear problem. Such a projection can be thought of as a perception module, in terms of our formulation. Given the success of kernel methods, this further motivates separating the treatment of perception and reasoning.

Neuroscience. Neuroscientists have also observed that information processing in the human brain goes from low-level (i.e., sensory input processing) to high-level (i.e., reasoning). There is ample evidence to support this both for both auditory (Kaas and Hackett, 1998; Rauschecker, 1998; Romanski et al., 1999; Wessinger et al., 2001; Warren and Griffiths, 2003; Zatorre and Belin, 2001; Zatorre et al., 2004) and visual information (Mishkin et al., 1983; Felleman and Van, 1991). Furthermore, there has been evidence that the development of primary visual cortical networks is more rapid than the development of primary motor networks in humans (Gervan et al., 2011). This motivates the idea that perception is a low-level functionality that is not necessarily problem-specific and that can be learned before learning to reason and take actions. In addition to this, there is evidence that the brain relies on a set of canonical neural computations that are reused for different problems (Carandini and Heeger, 2012). For example, normalization of neural responses is one such operation that is thought to underlie multiple other operations such as the representation of odours, the modulatory effects of visual attention, the encoding of value, and the integration

[3] BERT is the current state-of-the-art model for a multitude of natural language processing tasks.

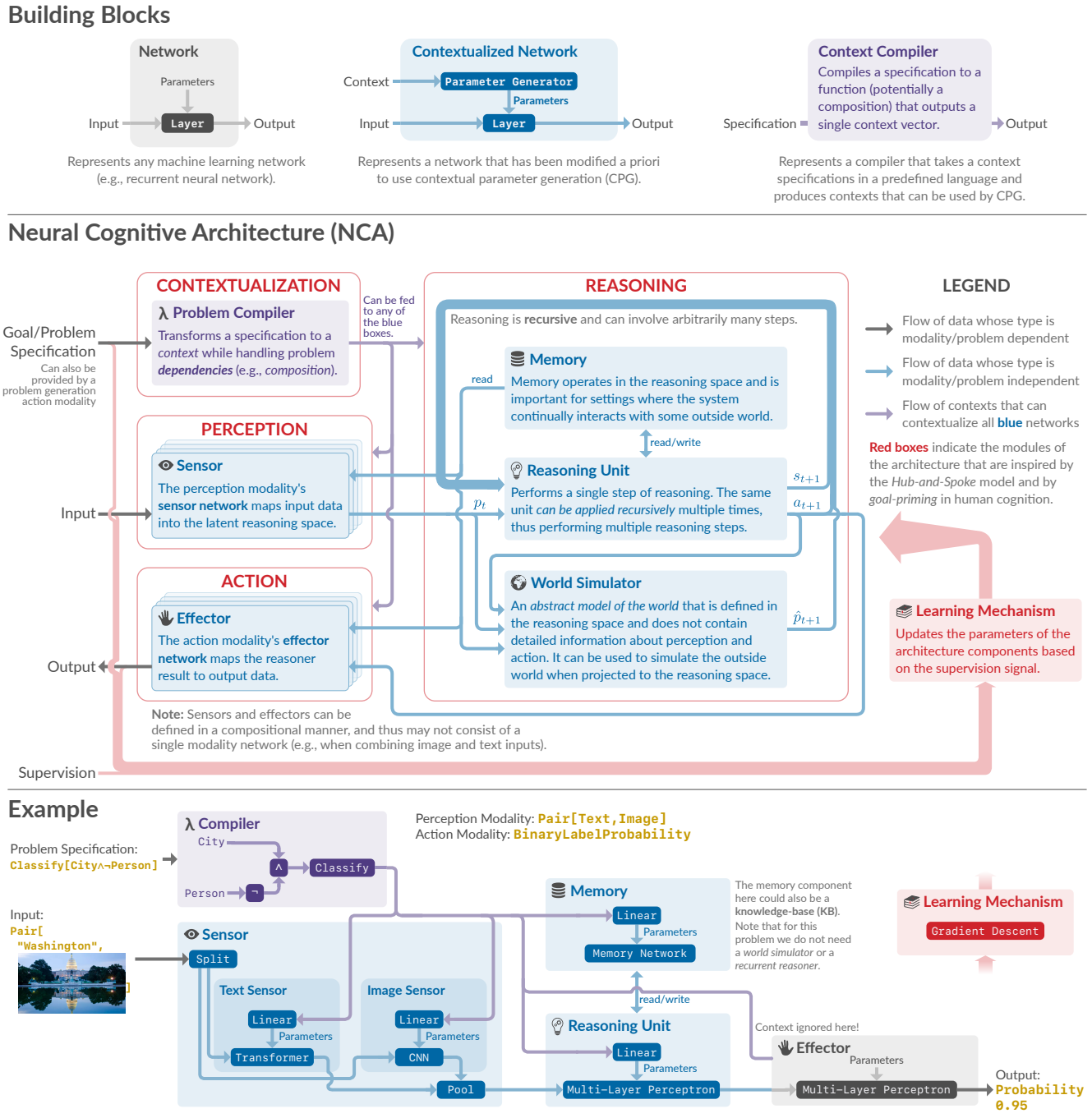

Figure 2: Overview of the proposed Neural Cognitive Architecture (NCA), its main building blocks, and a simple example showing an instance of this architecture for a classification problem. In the example, the noun "Washington" alone without the provided image, is ambiguous and would probably not result to a high probability of referring to a city and not a person, at the same time.

of multi-sensory information. This also supports the idea of abstracting over reasoning, by making the operations used to perform various tasks common across all tasks and finding other ways to specialize them.

Psychology. There has been significant evidence that *priming* is characteristic of human behavior (Tulving et al., 1982; Bargh and Chartrand, 2014; Weingarten et al., 2016). Priming is a technique where exposure to one stimulus influences the brain's response to a subsequent stimulus. For example, the word "dog" is recognized more quickly after having seen the word "animal". Priming can be perceptual, semantic, and conceptual. Perhaps most importantly for this thesis is *goal priming* (Custers and Aarts, 2005; Aarts et al., 2008; Papies, 2016; Takarada and Nozaki, 2018). Goal primes are cues that trigger goal-directed cognition and behaviour.

Here, a goal refers to a state or behaviour that has reward value and therefore motivates a person to pursue it. For example, priming the concept of drinking can increase soda consumption(Veltkamp et al., 2008), or priming the goal of impression formation leads to better memory organization and recall compared to a mere memorization goal (Chartrand and Bargh, 1996). Goal contextualization in our architecture is the computational equivalent of goal priming, in that having specific goals changes the way in which the different architecture parts function.

Benefits of Modularity. An important outcome of the Hub-and-Spoke architecture design is reducing the per-problem sample complexity. This means reducing the amount training data required to learn to solve each problem. This is because, for multiple existing machine learning models, most of the model complexity lies in perception (e.g., BERT). This becomes more prevalent in reinforcement learning systems playing video games where they receive as input the raw pixel values of video frames as they are being rendered while playing, and they are tasked with learning to extract information from these raw values (e.g., Bellemare et al., 2013; Bhonker et al., 2016; Vinyals et al., 2019). Such systems require massive amounts of training data to learn, and we argue that this is mostly due to their perception components. If these components were shared across multiple problems then their effective per-task sample complexity would be reduced significantly. In fact, Parisotto et al. (2015) show that pre-training agents on some arcade games, oftentimes helps them learn faster when deployed to play other, new, arcade games. Thus, assuming we can share the perception component across different problems, we only need problem-specific training data for the reasoning component. Moreover, due to the shared reasoning hub, the per-problem sample complexity can be further reduced, because the same reasoning component is used for solving all problems. An interesting setting is one where the perception component can be trained using supervised tasks with differentiable loss functions, and, at the same time, be shared with reinforcement learning (RL) tasks where the reward function is unknown and certainly not differentiable. We believe that this would significantly reduce the sample complexity of the RL tasks. In Section 4, we propose a case study for testing this hypothesis.

The proposed architecture components reflect assumptions about human cognition that are based on facts derived from psychology experiments, thus rendering the proposed architecture, a cognitive architecture. In the following sections we describe the different architecture components in more detail. Finally, in Sections 3.4.7 and 3.4.8, we describe how learning is performed. Note that, not all architectural components that we describe in the following sections are necessary for all problems. Therefore, for some problems, some of the components may be ignored (e.g., a world simulator may not be relevant for a text classification task).

3.4.4 Perception and Action Spokes

We define perception and action spokes using two kinds of data modalities: (i) *perception modalities* that represent data types that a model can receive as input, and (ii) *action modalities* that represent data types that a model can produce as output. Each kind of modality has a different specification:

- Perception Modalities: Input space modalities are defined as tuples $(\texttt{DataType}, \texttt{SensorNetwork})$, where `DataType` is the type of data supported by this modality (e.g., `String`), `SensorNetwork` is a contextualized network that takes inputs of type `DataType` and produces vectors of size L_s, and L_s is the reasoning input representation size. Given some data of type `DataType` (e.g., a string of characters with type `String`), and optionally, a context (described in the next section), the `SensorNetwork` produces a vector of size L_s, that the reasoning module can understand.
- Action Modalities: Output space modalities are defined as tuples $(\texttt{DataType}, \texttt{EffectorNetwork})$, where `DataType` is the type of data supported by this modality (e.g., scalar number in the interval $[0, 1]$), `EffectorNetwork` is a contextualized network that takes as input vectors of size L_e and produces outputs of type `DataType` (e.g., a linear transformation followed by a sigmoid activation function), and L_e is the reasoning output representation size.

Note that a modality can act as both a perception and an action modality, as long as both a *sensor* and an *effector* network are provided. In this case, we also allow the sensor and the effector networks to optionally share some or all of their parameters. Examples of various modalities are shown in Table 1. Modalities are defined such that, for any given input (or output) data type, there is a single matching perception (or action) modality that will be used.

Due to their generic definition, modalities can be *composed*. For example, given perception modalities $\mathcal{P}_1$ and $\mathcal{P}_2$, we can construct a pair modality $\texttt{Pair}[\mathcal{P}_1, \mathcal{P}_2]$, whose data type is a pair of $\mathcal{P}_1\texttt{.DataType}$ and $\mathcal{P}_2\texttt{.DataType}$, and whose sensor network is a function of the two modalities' sensor networks. For example:

$$x \mapsto \text{Pool}(\mathcal{P}_1\texttt{.SensorNetwork}(x\texttt{[0]}), \mathcal{P}_2\texttt{.SensorNetwork}(x\texttt{[1]})).$$

Compositionality gives the proposed NCA high expressive power with respect to the kinds of data it can handle. Compositionality, more generally (e.g., also at the problem space), is a core aspect of the proposed architecture and it is discussed in more detail in Section 3.4.6.

Communication and Language. An interesting direction that we wish to explore in the long term is to add support for a modality that corresponds to communication with other agents (i.e., an artificial learned language). This modality would act as both a perception and an action modality and we could define its data type as a fixed-size vector containing numeric values, for example. We can test for the ability of

Modality Examples			
Data Type	**Sensor Network**	**Effector Network**	**Description**
`String`	BERT Encoder	RNN Decoder	Text
`Image`	CNN	Deep Convolutional GAN	Image
`Scalar[0,1]`	–	MLP→Sigmoid	Binary Distribution
`Vector[0,1]`	–	MLP→Softmax	Categorical Distribution

Table 1: Example modalities. RNN stands for Recurrent Neural Network, CNN for Convolutional Neural Network, GAN for Generative Adversarial Network, MLP for Multi-Layer Perceptron, `Scalar[0,1]` for a single number in the interval $[0, 1]$, and `Vector[0,1]` for a vector containing numbers in the interval $[0, 1]$.

agents to learn a language and communicate effectively by conducting experiments in a multi-agent setting where solving certain problems requires coordination and collaboration. This is related to the work of Sukhbaatar et al. (2016) and Andreas et al. (2017).

3.4.5 Reasoning Hub

The reasoning component of the proposed architecture consists of a few parts. At the core lies the *reasoning unit*. This unit transforms the perception component output to an input for the action component, and is represented as a contextualized network. It is generally accepted that not all problems require the same amount of reasoning (Kahneman and Egan, 2011). For example, solving an algebra problem requires more *thinking* than recalling your own name. Therefore, we argue that the ability to reason for arbitrary amounts of time, depending on the problem being solved, is an important aspect of general learning and intelligence. Most existing machine learning approaches do not allow for a variable amount of reasoning, as the amount of computation is predefined and fixed, as part of the network architecture. The few attempts that do allow for this have been limited to very specific problems and have only shown small gains over preexisting fixed computation time approaches (Graves, 2016; Dehghani et al., 2019). In order to enable this capability in the proposed neural cognitive architecture, we decided to make the reasoning unit *recursive*, meaning that its output can optionally be fed back as input again, to recurse over the reasoning transformation. Each application of the reasoning transformation can be thought of as a reasoning step. The reasoning unit also outputs a decision on whether or not to stop, so that it can stop reasoning and produce an output at some point. The recursive nature of this unit introduces several challenges with respect to how it should be trained. Our initial plan is to incur a *pondering cost*, which is proportional to the number of reasoning steps used, and add that cost to the loss function used to train the reasoning unit.

Recursion. More formally, at each time step t, the reasoning unit performs the following transformation[4]:

$$[a_{t+1}, s_{t+1}, \texttt{STOP}_{t+1}] = R(p_t, a_t, s_t), \tag{5}$$

where R represents the reasoning unit transformation, a_t represents the reasoning unit output at time t, s_t represents the internal state of the unit at time t, p_t represents the reasoning unit input at time t which comes from the perception component (note that if the system operates in a real-time environment, this may be different across different reasoning steps), and $\texttt{STOP}_t$ is a boolean flag representing the decision of the reasoning unit about whether or not to stop reasoning at time t. Finally, a_T is fed to the action component, where T is such that $\texttt{STOP}_T = \texttt{True}$. Enhancing the reasoning unit with a state significantly increases its modeling capacity; it can now even perform a search with backtracking support (e.g., dynamic programming). This initial approach is inspired by the work of Graves (2016).

Memory. In designing general learning architectures, we need allow for an explicit way for learning systems to remember experiences. This can happen implicitly, through the learned model parameters (assuming high capacity networks), but it can also be modeled explicitly by equipping the agent with a memory component. Cognitive architectures often use some form of memory that is symbolic, such as a knowledge-base (KB) that contains learned facts. We propose to add a memory component to our architecture, where all memories are represented in the latent reasoning space, rather than being grounded in the perception or action modality data types. This allows the memory to abstract away details about the data that are not relevant to the reasoning process. The way memory is added to our architecture is through the reasoning unit, which is enhanced such that it can read and write to memory, while performing its transformation. More formally, we define the memory component as consisting of two functions, $M_{\texttt{READ}} : K \mapsto V$ and $M_{\texttt{WRITE}} : (K, V) \mapsto ()$, where K and V correspond to the memory key and value types, respectively, and the "$\mapsto$" notation is used to denote the function input and output types[5]. Possible design choices for the memory include memory networks (Sukhbaatar et al., 2015), or even KBs defined over the latent reasoning space.

We propose to start with a simple, yet novel[6], attention-based memory mechanism. In this case, the memory is defined as a pair of matrices, $M_k \in \mathbb{R}^{M \times D_k}$, and $M_v \in \mathbb{R}^{M \times D_v}$, where M is the memory size, D_k is the dimensionality of the keys, and D_v is the dimensionality of the values stored in the memory. M_k contains the memory keys and M_v contains the corresponding memory values. Let us refer to the K-valued input of $M_{\texttt{READ}}$ and $M_{\texttt{WRITE}}$ as the *query*. Queries are

[4] This is not equivalent to simply using a recurrent neural network (RNN), because the number of recursion steps is not predetermined.

[5] We use "()" to represent the "void" type, meaning that the function returns no values, and is only used for its side effects.

[6] Novel because we are not aware of prior work that learns a memory indexing mechanism.

defined as vectors of size D_k. When a component wants to access a value stored in memory, it needs to provide a query "describing" that value[7]. We also define an *indexing* function, $I : K \mapsto \Delta^M$, where Δ^M denotes the M-simplex, which contains all vectors of size M whose elements are in $[0, 1]$ and sum to 1. Intuitively, the indexing function maps from a query to a distribution over memory locations The indexing function that we plan to use initially is the scaled dot-product attention by Vaswani et al. (2017):

$$I(q) = \text{Softmax}\left(\frac{qM_k^T}{\sqrt{D_k}}\right), \tag{6}$$

which effectively measures the similarity between the query and all the memory keys. Then, the memory read function is defined as (in pseudocode):

$$M_{\texttt{READ}}(q) : \text{return } I(q)M_v, \tag{7}$$

which returns a convex combination of all stored values, based on the computed index. The memory write function is similarly defined as:

$$M_{\texttt{WRITE}}(q, v) : M_v := \lambda I(q)v + (1 - \lambda I(q))M_v, \tag{8}$$

where $:=$ is used to denote assignment, and λ is an M-sized vector with values in $[0, 1]$ that denotes the strength of the write operation. If λ is closer to 1, then old values are forgotten faster. λ can be set adaptively, based on how often each value is being read. For example, it can be set closer to 1 for values that are rarely read. The learnable parameters of this learning mechanism consist of the parameters of I, and the memory keys, M_k. We can initialize M_v with zeros.

Allowing the memory indexing mechanism to be learnable, by using separate keys and values[8], enables *associative learning and memories*, which have been shown to be important aspects of human cognition (Fanselow and Poulos, 2005; Ranganath and Ritchey, 2012). In psychology, associative memory is defined as the ability to learn and remember the relationship between unrelated items (e.g., remembering the name of someone or the aroma of a particular perfume). This is enabled by our indexing mechanism because it allows for two unrelated values to have similar keys. This is mainly because we learn keys separately from the values they correspond to. Note that our proposed memory mechanism also allows for a natural way of *forgetting*, where the keys of unused values change while learning to the point where they may be used for storing other unrelated values instead.

We also allow the sensor and effector networks to optionally read from this memory. This can be important in cases where perception depends on past experiences. Tulving et al. (1982) provides some evidence supporting that this has been observed to be true of human perception (this is known as *priming* in psychology literature).

[7]Note that, the querying mechanisms are also learned, similar to the indexing mechanism.

[8]As opposed to indexing by comparing queries to values as done in memory networks.

World Simulator. An important aspect of human reasoning is simulating the external world. Jay Wright Forrester, the father of system dynamics, described a mental model as: *"The image of the world around us, which we carry in our head, is just a model. Nobody in his head imagines all the world, government or country. He has only selected concepts, and relationships between them, and uses those to represent the real system."* (Forrester, 1971). There is significant evidence of the importance of simulation in neuroscience (Singer et al., 2018). For example, Nijhawan (1994) shows that to strike a cricket ball one must estimate its future location, rather than where it is now. Bialek et al. (2001) show that prediction has the fundamental theoretical advantage that a system which parsimoniously predicts future inputs from their past, and that generalizes well to new inputs, is likely to contain representations that reflect their underlying causes. Furthermore, they show that much of sensory processing involves discarding irrelevant information, such as that which is not predictive of the future, to arrive at a representation of what is important in the environment for guiding action. Another related line of work is in the importance of auditory feedback (i.e., when we hear ourselves speaking). The study of neural mechanisms underlying audio-vocal integration has shown that auditory feedback may be used for updating internal representations of mappings between voice feedback and speech motor control. One of the earliest demonstrations of the role of auditory feedback in voice control is the Lombard effect, where people raise their voice amplitude to overcome environmental noise (Lombard, 1911; Lane and Tranel, 1971). A related phenomenon is side-tone amplification, in which people increase their voice loudness when their self-perceived loudness is too quiet to achieve a communication goal, and vice versa (Lane and Tranel, 1971). Given this strong evidence from neuroscience, we argue that in an interactive setting, where the learning agent keeps interacting with an outside world—which may also include other agents—being able to simulate that world can be very important. For example, this ability could enable a search over the potential implications its decisions will have on that outside world.

We thus propose to add a *world simulator* component to our neural cognitive architecture. Formally, the simulator S performs the following prediction:

$$\hat{p}_{t+1} = S(p_t, a_{t+1}), \tag{9}$$

where $\hat{p}_{t+1}$ is a prediction estimate of p_{t+1}. Furthermore, we allow the world simulator to read from memory (as defined in the previous paragraph), but not write to it. Intuitively, the world simulator is trying to predict the next perception input, given the current perception input and action output, while operating only in the latent reasoning space. Similar to the memory component, this allows the simulator to abstract away information that is not relevant to the problems the system is learning to solve. This type of world simulation in a latent reasoning space is also supported by neuroscientific evidence (e.g., Keller et al., 2012).

Recently, Ha and Schmidhuber (2018) proposed using an RNN-based world simulator for playing games in an RL set-

ting. They use a variational auto-encoder (VAE) to compress the input images to a smaller vector representation and then learn a model that simulates the environment in this vector space. This differs from our proposal in that, we are simulating the world in the latent reasoning space that our system learns. This should help us obtain a representation, that has higher information content that is relevant for the reasoner.

3.4.6 Goal Contextualization

Even though deep learning methods are very effective at learning representations for arbitrary data modalities, they are often treated as black-box methods offering little control over how information is shared across different tasks, and over what exactly the networks are learning. For example, we can rarely guarantee that a network will generalize well to new tasks, and we often also have to keep training the network with new problem-specific data, in order for it to generalize better. Furthermore, deep learning approaches often render generalizing to new tasks, for which we might have no data at all, impossible. However, most real-world problems can be defined in terms of simpler problems (e.g., translating sentences relies on first being able to translate single words). Therefore, we argue that the ability to represent problems in a way such that they can be transformed and composed out of other problems, is an important aspect of general learning and intelligence. As discussed in Section 3.2, this motivated our recent work in contextual parameter generation (CPG) for machine translation (Platanios et al., 2018) and question answering (Platanios* et al., 2019), and forms the basis of *contextualization*. In the proposed neural cognitive architecture, contextualization plays the important role of emulating the *goal priming* mechanism that is inherent in human intelligence and learning. We now describe how this is achieved, in three parts: (i) we first describe how problems (or goals) are specified through some language, (ii) we then define an architectural component that *compiles* the problem specification to a representation that can be used to contextualize other parts of the NCA by using CPG, and (iii) we describe how this allows for the learning system to generate its own target problems (or goals) that it aims to learn.

As shown in Figure 2, we also allow the sensor and effector networks to be contextualized because perception and action are often not independent of the problem being solved. This is motivated by the fact that priming in humans can be perceptual, semantic, and conceptual (Bargh and Chartrand, 2014). From a machine learning perspective, we have also shown the usefulness of contextualizing equivalents of perception and action modules, when we proposed using CPG for universal neural machine translation (Platanios et al., 2018).

Problem Specification. We first need to define a representation for problems. We propose to use a fixed language for this representation, which could take multiple forms:

- Fixed-Size Vector: Problems could be represented as continuous-valued, fixed-size, vectors (e.g., Snell et al., 2017; Wang et al., 2017b; Grover et al., 2018). For example, given a fixed number of pre-specified problems the system may learn vector embeddings to represent them. The main disadvantage of this approach is that the vector representations of learning problems may not be interpretable.
- Natural Language: This could be a problem description that is provided as input to the system (e.g., "*Identify human faces in the input image.*"). This is the approach taken, for example, by McCann et al. (2018).
- Structured Language: This could be first-order logic (e.g., "`Collect[JellyBean]`$\wedge\neg$`Collect[Onion]`"), or more general (e.g., "`If[JellyBean]Then[Collect]Else[Avoid]`", or even a Python program).

Problem Compilation. Given a problem specification, we need to define a *compiler* that takes it as input and produces a composition of learnable functions that, when evaluated, results in a single structured representation for the problem (e.g., a set of vectors). This representation can then be used to contextualize different parts of the proposed architecture (e.g., sensor or effector networks, or parts of the reasoner that are discussed in the next section). Given that the representation can potentially be a set of vectors, we could use different parts of that structure to contextualize different parts of the architecture. For example, text sensor networks could be contextualized using an embedding of the language in which the text is written. Note that contextualizing networks is optional, as it is sometimes not necessary (e.g., the effector network used in the bottom of Figure 2 is not contextualized).

The choice of the problem compiler is important. For fixed-size vectors and natural language specifications the compiler could be as simple as just a neural network (e.g., a multi-layer perceptron, a recurrent neural network, or a Transformer network). However, for other structured languages the compiler would be something more similar to programming languages compilers. Some examples of representations and their corresponding compiled forms are shown in Table 2. Following from the previous section examples, given a problem specification that is written as a Python program, we could also compile it into a composition of learnable functions.

This definition of problem specifications and problem compilers allows us to make the contextualization mechanism very flexible and extensible by introducing operators that compose compiled forms in arbitrary ways. For example, we could have two problem specifications, each with their own compiler, and a separate operator that allows us to merge the two compiled forms, resulting in a single final context vector.

Problem Generation. An important aspect of human learning is that, even though nature provides us some reward signals for our actions (e.g., eating resolves hunger), we often "invent" new problems that we learn to solve. We could argue that this is a way of structuring much larger overarching problems into multiple subproblems. This human behavior aspect is very interesting and, at the same time, not really tackled at all by current machine learning systems. Therefore, we propose to let our learning system "invent" problems on its own. For this section, we will use a reinforcement learning setting where a learning agent can perceive certain things about the

Problem Compiler Examples		
Specification	**Compiled Form**	**Explanation**
`Classify[City]`	$g_{\texttt{Classify}}(c_{\texttt{City}})$	Predict if the input (e.g., "Washington") is a city.
`Classify[City∧¬Person]`	$g_{\texttt{Classify}}(g_{\wedge}(c_{\texttt{City}}, g_{\neg}(c_{\texttt{Person}})))$	Predict if the input is a city and not a person.
`Caption[Image,English]`	$g_{\texttt{Caption}}(c_{\texttt{English}})$	Generate a short English sentence describing the input. E.g., generate captions for images.
`Translate[English,German]`	$g_{\texttt{Translate}}(c_{\texttt{English}}, c_{\texttt{German}})$	Translate the input from English to German. This is interesting because the modularity of our architecture means that this problem specification could even be used to translate images containing text, for example.

Table 2: Example uses of the problem compiler. We use c with different subscripts to denote context vectors representing primitives in the problem specification language, and g with different subscripts to denote transformation functions for context vectors (which could be defined as learnable neural networks, for example).

environment in which it "lives" and take actions. Oftentimes, the agent receives a reward, but it may not know why. Thus, in such a setting, it would make sense for the agent to try and "invent" problems to solve, that would result in higher collected rewards. We propose to introduce one additional action modality that allows the agent to generate problem specifications, that are directly fed in the problem compiler[9], and can contextualize multiple parts of the architecture.

For the fixed-size vector specification format, this could be implemented by having the effector network output a vector representing the problem. Perhaps more interestingly though, we could define a structured language that only depends on the agent's perception and action modalities. This would allow the agent to generate arbitrary problem specifications that only depend on what it is able to perceive and how it can act. For example, given a perception modality that identifies the types of items in the environment, and an action modality that can collect items, we could define the problem specification language to be:

`(¬)Collect[<Item>](∧(¬)Collect[<Item>])*`,

where $\neg$ denotes the logical NOT operation, $\wedge$ the logical AND operation, parenthesis denote optional parts, `<Item>` denotes any item type that can be sensed by the item identification perception modality, and $\star$ denotes that the term in parenthesis preceding it can be repeated zero or more times. Note that `Collect[·]` acts as a logic predicate that can be applied on any item type. An example specification in this language is `Collect[JellyBean]∧¬Collect[Onion]`.

We propose to formalize this problem generation mechanism and allow learning systems to decide on the problems they are learning to solve.

3.4.7 Learning Mechanisms

The architecture components presented so far depend on parameters that need to be learned (e.g., the weights of neural network layers used). Learning consists of setting the values of these parameters so that the system as a whole can solve the target problems. We assume that all components are formulated as functions that are differentiable with respect to their parameters[10]. Under this assumption, we define our learning mechanism as follows:

1. Each action modality can optionally provide a *feedback mechanism*. Let us denote the output of the modality's effector network as a function, $f_\theta(x)$, where x represents all inputs that it depends on. In this case, f represents the composition of all architecture modules that participated in producing this output (i.e., this includes the reasoning module, the goal contextualization module, and the relevant perception modalities). Then, we define the feedback mechanism as a function, h, of $f_\theta(x)$ and the external environment. For example, if $f_\theta(x)$ is producing a distribution over classes (for a multi-class classification problem), h could be defined as:
$$h(f_\theta(x), y) = f_\theta(x) - y, \tag{10}$$
where y represents a one-hot representation of the true class assignment provided by the environment. The main constraint on h is that it should produce an output that can be multiplied with $\nabla_\theta f_\theta(x)$.
2. Whenever an action modality produces an output and a corresponding feedback signal is returned from the environment, a gradient-based parameter update is performed along the following direction:
$$\mathcal{D}_\theta \triangleq \underset{\text{External}}{h(f_\theta(x), \mathcal{E})} \underset{\text{Internal}}{\nabla_\theta f_\theta(x)}, \tag{11}$$
where $\mathcal{E}$ represents the external environment. Note that the first part, shown in blue, is provided from the external environment, whereas the second part, shown in red, can be computed internally from the learning system itself. This separation is interesting from a human cognition perspective because, intuitively: (i) a human would know how to tweak their brain to move their hand further forward (internal update), while (ii) the external environment could tell them that to achieve a particular goal they would need to move their hand forward (external update). The model update could be a stochastic gradient descent step:
$$\theta_{t+1} = \theta_t + \lambda_t \mathcal{D}_{\theta_t}, \tag{12}$$

[9] In this case, we assume that no problem specification is provided to the agent as input.

[10] Note that this is a very general assumption that holds for most deep learning models, and a lot of machine learning models, more generally.

where λ_t represents the learning rate, or it could be a more elaborate update such as when using Adam (Kingma and Ba, 2014) or AMSGrad (Reddi et al., 2018).

Equation 11 is interesting because it can be used to unified multiple different learning paradigms, such as supervised, semi-supervised, unsupervised, and reinforcement learning, under one formulation. For example:

- Supervised Learning: In this case, the gradient-based updates as computed by differentiating a loss function, $\mathcal{L}(f_\theta(x), \mathcal{E})$. This fits in our formulation by defining the feedback mechanism using the chain rule of differentiation:
$$h(f_\theta(x), \mathcal{E}) \triangleq \frac{\partial \mathcal{L}(f_\theta(x), \mathcal{E})}{\partial f_\theta(x)}. \quad (13)$$
For example, for L2 loss we have $h(f_\theta(x), y) \triangleq f_\theta(x) - y$, and for the cross-entropy classification loss we have $h(f_\theta(x), y) \triangleq y / f_\theta(x)$.
- Semi-Supervised Learning: Can often also be formulated in terms of minimizing a differentiable loss function and thus Equation 13 also applies here.
- Unsupervised Learning: In this case, $h(f_\theta(x), \mathcal{E})$ does not depend on $\mathcal{E}$ at all and could be defined internally as well. More specifically, h could be used to perform some sort of *self-reflection*. This is a direction we wish to explore more in the future, but may be outside the scope of this thesis and is described in a bit more detail in the last section.
- Reinforcement Learning: In the case of Q-learning (Watkins and Dayan, 1992), we can have an action modality that predicts the Q-function value (Mnih et al., 2013) and then the learning mechanism can use a supervised learning feedback function, h, to learn it using the rewards provided by the environment. In the case of policy gradient methods (Sutton et al., 2000), h can be defined as the advantage function being used, or even some function of the advantage for more complex methods (Mnih et al., 2016; Wang et al., 2017a; Schulman et al., 2017). More interestingly, if we want to use experience replay, as done by (Mnih et al., 2013), we could develop a variant where: (i) the perception and action modality parameters are fixed and we are training only the problem compiler and the reasoning modules, and (ii) the stored experiences that are replayed are not represented in the original data space, but rather in the more abstract and compact reasoning space. This has the significant advantage of being able to store a lot more experiences, as memory is typically the bottleneck when using experience replay. Furthermore, we would only be storing information that is relevant to reasoning.

Our learning mechanism manages all feedback mechanisms and determines how to apply the corresponding updates and what learning rate to use for each one. Initially, we plan to use the same learning rate for all parameters and feedback mechanisms with exponential decay over time. However, our definition allows us to use potentially different learning rates for each parameter and for each learning goal (defined by corresponding feedback mechanisms). Next, we plan to integrate the ideas presented in Sections 3.1 and 3.3, to this learning mechanism. In the long term, we would like to explore other interesting directions such as *staged learning*.

Staged Learning. The aforementioned reinforcement learning example on experience replay demonstrates the idea of staged learning. In staged learning, we freeze the learning of the perception and action modalities early on during training (e.g., by significantly lowering the corresponding learning rate), and then focus more on training the reasoning module. As discussed in the beginning of Section 3, this would be more similar to how human learning works. Assuming that the perception and action modality networks have already been trained using a diverse set of learning goals, freezing them should allow for the reasoning module to tackle new learning goals in a fixed latent space, determined by these pretrained networks. We believe that this will result in significantly faster training times.

Mixed-Paradigm Learning. As shown earlier, our learning mechanism is a generalization of multiple existing learning paradigms thus allowing us to mix them together by simply intertwining their gradient-based updates. For example, we can take a gradient descent step towards minimizing a supervised cross-entropy classification loss, and then take a gradient descent step that improves the current Q-function estimate, in a reinforcement learning setting. This introduces multiple challenges that we will have to overcome, including, but not limited to:How do we properly balance the gradient contribution from each learning problem? How do we set the per-learning-goal and per-parameter learning rates? How do we make the learning mechanism scale? How do we properly batch the training data? Other learning paradigms, such as active learning and curriculum learning, can also be supported by designing appropriate perception and action modalities.

3.4.8 Never-Ending Learning

A never-ending learning system must be highly modular and allow for the addition and removal of modules without requiring a complete retraining from scratch. For this reason, we plan to implement the proposed architecture in a highly modular manner, with each module being completely independent of the rest and having a fixed, well-defined, and generic interface. This will allow for adding and removing perception and action modalities and for extending the problem specification language, without requiring a complete retraining from scratch every time such a modification is made. Furthermore, each module will be solely responsible for persisting its state, so that we can keep extending the architecture and avoiding training restarts, as much as possible. Our goal by the end of this project is for the proposed architecture to have been training for the duration of this thesis, with some modules having been trained for a year and some newer ones only for a few days. This will allow us to provide convincing evidence for its never-ending learning capabilities. Moreover, unlike NELL (Mitchell et al., 2018), we aim for this system to fully avoid complete training restarts throughout its lifetime. Finally, we want to explore directions where the latent reasoning representation is also extensible without requiring complete training restarts. This is a long term goal that goes

beyond the scope of this thesis.

4 Evaluation

In order to test our hypothesis from Section 1, we propose to perform multiple case studies. Some of these case studies are performed over a simulated world that we have designed and built, called the *Jelly-Bean World (JBW)*[11]. We designed this world specifically for enabling us to test the properties of never-ending learning systems, that are otherwise hard and very expensive to evaluate using real-world datasets. In the following section we describe this simulated world, and then we provide a list of the proposed case studies.

4.1 Jelly Bean World (JBW)

The JBW offers a controlled environment where a learning agent "lives", and which defines the problems that the agent can solve and the reward it obtains for solving each problem. The JBW is a procedurally generated two-dimensional grid world, where items of various types can be placed on each grid cell. An example illustration is shown in Figure 3. In this world, time is discrete and measured in terms of simulation steps. Each item has a color and a scent, each represented by a fixed-size continuous-valued vector. The learning agents have a visual field range within which they can see the colors of the items in each cell. They can also smell the scent of their current cell. The scent of each cell is computed by simulating the diffusion of scent across all items in the world[12]. Items are also allowed to have other properties. For example, walls can block agent movement and onions may tend to cluster together. The JBW has the following desirable properties:

- Multiple Problems: We can define multiple learning problems. For example, we can have the agent learn to collect jelly beans and avoid onions. This results in a reinforcement learning setting where the agent receives sparse rewards. We can then have the same agent learn to predict the color and scent of each item, and also classify which item a specific color or scent corresponds to. These problems can be learned in a supervised fashion, using a differentiable loss function defined over the items the agents has "stepped over" so far. When combined with the original RL problem, they may help reduce its sample complexity.
- Continual Learning: In the JBW, the learning agent never "dies". It is instead learning continually, in a never-ending fashion. The world is generated in a procedural manner, and thus, no matter how far the agent decides to explore, the JBW imposes no limits. This allows us to observe how fast the agent learns to solve different problems, and to also observe how its learning rate changes as time progresses and the agent learns to solve other problems. In fact, these conditions are also closer to human learning, in that it is also never-ending and of a non-episodic nature.
- Multiple Modalities: There exist two perception modalities, scent and vision, which have very different characteristics. Vision has a limited range, but very high precision, meaning that the agent knows exactly what color each grid cell has, within its visual field. On the other hand, scent has infinite range (through diffusion it can propagate to very long distances), but very low precision (it can be very hard to decompose the scent at the current grid cell into all the items that contribute to it, and their distance from the agent). At the same time, knowing the scent of an item and computing the difference in the agent's perceived scent between one cell and another, can provide a lot of information about which direction an item's scent is coming from.
- Ever-Changing Learning Problems: The mechanics of the JBW allow us to construct conditions for ever-evolving learning problems. For example, let us assume that some of the items are notes that contain learning problem descriptions, such as `Collect[JellyBean]` $\rightarrow 10 \wedge$ `Collect[Onion]` $\rightarrow -10$. This describes an item collection and avoidance problem, along with associated rewards. Such notes can be spread around the world and, until the agent finds them, he cannot tackle the learning problems they describe. Some of these notes may contain recipes for building new items, that give the agent unique abilities (e.g., binoculars to extend its visual range). Furthermore, some items may be invisible to the agent (both in terms of color and scent) when it starts learning. For example, some of the color vector dimensions may only become unmasked if the agent manages to obtain a specific item, such as an X-ray machine. This creates conditions similar to human learning, where learning problems exist abundantly in the world (and may also be generated in a procedural manner), but are not available until certain other problems are solved.

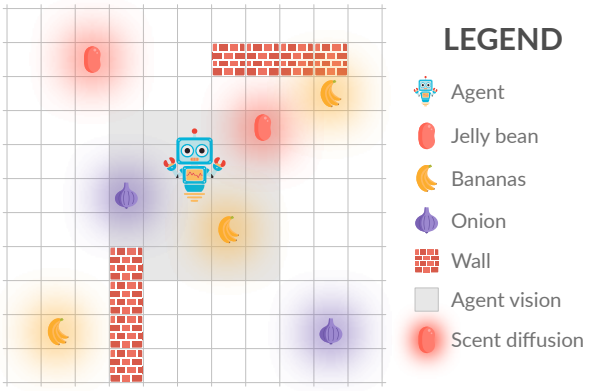

Figure 3: Jelly-Bean World example.

In JBW, we can also control for the relationship between learning problem difficulty versus reward. This can create interesting situations where solving more difficult problems does not necessarily imply collecting a higher reward. The JBW can thus get arbitrarily complex and difficult, while remaining controllable. It therefore allows us to test several aspects of our hypothesis, as well as test whether the various parts of the proposed neural cognitive architecture are beneficial to learning, or not (e.g., we can design problems where memory is necessary, such as ones that require counting items). We propose to use simple metrics to measure learning performance in the JBW, such as the performance for each

[11] https://github.com/eaplatanios/nel_framework

[12] Scent diffusion means that strength of an item's scent decays with distance from the item.

problem and across simulation steps. Performance could be measured in terms of a metric computed over a validation set, or simply in terms of cumulative reward collected, and its rate of change. Furthermore, our JBW simulator already supports multiple agents interacting with the same grid-world and with each other, and thus also allows us to conduct multi-agent experiments (e.g., test for agent communication).

4.2 Case Studies

We propose to perform the following case studies:

- **JBW #1:** The agent gets a positive reward for collecting some items and a negative rewards for collecting some other items. We will test performance when only being provided a single problem specification (e.g., `Collect[JellyBean]`$\wedge\neg$`Collect[Onion]`), and when also trying to classify or recognize items based on their color or scent. The latter case should help us test whether the mixed learning paradigm scenario results in better learning performance for our architecture.
- **JBW #2:** Same as JBW #1, except that we let the agent generate the problem specification, rather than having it be provided as input from the environment. This should help us test the problem generation and goal contextualization capabilities of the proposed architecture.
- **JBW #3:** Design some tasks in the JBW that require the use of memory (e.g., counting items) and world simulation, so that we can test the relevant parts of the reasoner.
- **Atari Games:** Learn to play multiple Atari games using a single learning system. This should helps us test whether modularizing and sharing perception and action modalities across games can help reduce the sample complexity of learning to play a new game, after having learned to play some others. In this case, the problem specification language will consist simply of an Atari game identifier.
- **NLP:** Tackle multiple natural language processing (NLP) problems using a single NCA learning system. An example would be to try and outperform BERT in the problems Devlin et al. (2018) tackle, or to compete in the decaNLP challenge (McCann et al., 2018). It will also be interesting to explore multi-modal NLP problems such as visual question answering and problems involving knowledge graphs. This will allows us to test for multi-modal learning aspects.

5 Proposed Timeline

We propose to structure the proposed thesis work in four main chapters, as discussed in Section 3:

1. **Learning from Multiple Noisy Labels** `[DONE]`**:** Published in (Platanios et al., 2014; 2016; 2017; 2019).
2. **Contextual Parameter Generation** `[01/18-09/19]`**:** We have already performed extensive empirical evaluations of the core idea behind contextual parameter generation (e.g., Platanios et al., 2018; Platanios* et al., 2019). In the next couple of months we aim to obtain a theoretical understanding of when and why contextual parameter generation works and of the limitations it addresses.
3. **Self-Reflection** `[07/19-12/19]`**:** We plan to develop and evaluate the differentiable intrinsic reward mechanism that was briefly discussed in Section 3.3.
4. **Unified Architecture** `[09/19-05/20]`**:** We plan to work towards developing a unified architecture as presented in Section 3.4, in the following order:
   i. Goal Contextualization: Can we use contextual parameter generation to achieve the equivalent of goal priming in a machine learning system? We have already shown that we can do that in a couple specific applications (Platanios et al., 2018; Platanios* et al., 2019). However, it will be challenging to extend that to a multi-problem setting with a problem specification language handling structured information sharing across the different problems.
   ii. Module Sharing: Can we effectively share the same perception, reasoning, and action modules across multiple problems? There has been some early work in this direction (e.g., Kaiser et al., 2017), but performance generally drops for problems for which we have a lot of data available. It will be challenging to overcome this issue, but succeeding would pave the way for more general learning systems.
   iii. Unified Learning Paradigm: Can we design a unified learning paradigm that encompasses supervised, semi-supervised, unsupervised, and reinforcement learning, and can be successfully used to learn in mixed-paradigm settings? We proposed a first step in this direction in Section 3.4.7, but it may be challenging to successfully deploy such a system in real-world applications. If successful, this could potentially lead to a merge between some seemingly distinct ideas about machine learning.
   iv. Goal Generation: Can we design and implement a system that can successfully generate its own learning goals and let them guide its learning process? How do we evaluate such a capability?
   v. Self-Reflection: Can the proposed neural cognitive architecture achieve *self-reflection* capabilities? Self-reflection capabilities could form a basis for unsupervised learning and could potentially be achieved by designing appropriate *self-sensors* and *self-effectors* (i.e., designing appropriate perception and action modalities). In fact, we hope that we may be able to model the learning mechanism itself as the result of the interaction between certain self-sensors and self-effectors. Given our framing of the learning mechanism using Equation 11, we believe we may be able to model self-reflection by using the internal component of the feedback direction, and having the feedback mechanism be provided by self-sensors, rather than by the external environment.

   Points (iii), (iv), and (v) above, are long-term goals that may not be finished during the indicated time frame. However, we hope to make some first steps before defending this thesis in May 2020.

References

Aarts, H., Custers, R., and Veltkamp, M. (2008). Goal Priming and the Affective-Motivational Route to Nonconscious Goal Pursuit. *Social Cognition*, 26(5):555–577.

Al-Shedivat, M., Dubey, A., and Xing, E. P. (2017). Contextual Explanation Networks. *CoRR*, abs/1705.10301.

Anderson, J. R., Bothell, D., Byrne, M. D., Douglass, S., Lebiere, C., and Qin, Y. (2004). An Integrated Theory of the Mind. *Psychological Review*, 111(4):1036.

Andreas, J., Dragan, A., and Klein, D. (2017). Translating Neuralese. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 232–242.

Andreas, J., Rohrbach, M., Darrell, T., and Klein, D. (2016a). Learning to compose neural networks for question answering. *arXiv preprint arXiv:1601.01705*.

Andreas, J., Rohrbach, M., Darrell, T., and Klein, D. (2016b). Neural module networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 39–48.

Angluin, D. and Laird, P. (1988). Learning from Noisy Examples. *Machine Learning*, 2(4):343–370.

Balcan, M.-F., Blum, A., and Mansour, Y. (2013). Exploiting Ontology Structures and Unlabeled Data for Learning. *International Conference on Machine Learning*, pages 1112–1120.

Bargh, J. A. and Chartrand, T. L. (2014). The Mind in the Middle: A Practical Guide to Priming and Automaticity Research.

Bellemare, M. G., Naddaf, Y., Veness, J., and Bowling, M. (2013). The Arcade Learning Environment: An Evaluation Platform for General Agents. *Journal of Artificial Intelligence Research*, 47:253–279.

Bengio, Y. and Chapados, N. (2003). Extensions to Metric-Based Model Selection. *Journal of Machine Learning Research*, 3:1209–1227.

Bengio, Y., Courville, A., and Vincent, P. (2013). Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1798–1828.

Bhonker, N., Rozenberg, S., and Hubara, I. (2016). Playing SNES in the Retro Learning Environment. *arXiv preprint arXiv:1611.02205*.

Bialek, W., Nemenman, I., and Tishby, N. (2001). Predictability, Complexity, and Learning. *Neural Computation*, 13(11):2409–2463.

Blei, D. M., Ng, A. Y., and Jordan, M. I. (2003). Latent Dirichlet Allocation. *Journal of Machine Learning Research*, 3(Jan):993–1022.

Blum, A. and Mitchell, T. (1998). Combining Labeled and Unlabeled Data with Co-training. In *Conference on Computational Learning Theory (COLT)*, pages 92–100.

Cao, Y., Yu, W., Ren, W., and Chen, G. (2013). An overview of recent progress in the study of distributed multi-agent coordination. *IEEE Transactions on Industrial informatics*, 9(1):427–438.

Carandini, M. and Heeger, D. J. (2012). Normalization as a Canonical Neural Computation. *Nature Reviews Neuroscience*, 13(1):51.

Caruana, R. (1997). Multitask Learning. *Machine Learning*, 28(1):41–75.

Chartrand, T. L. and Bargh, J. A. (1996). Automatic Activation of Impression Formation and Memorization Goals: Nonconscious Goal Priming Reproduces Effects of Explicit Task Instructions. *Journal of Personality and Social Psychology*, 71(3):464.

Collins, J. and Huynh, M. (2014). Estimation of Diagnostic Test Accuracy Without Full Verification: A Review of Latent Class Methods. *Statistics in Medicine*, 33(24):4141–4169.

Collins, M. and Singer, Y. (1999). Unsupervised Models for Named Entity Classification. In *Joint Conference on Empirical Methods in Natural Language Processing and Very Large Corpora*.

Collobert, R. and Weston, J. (2008). A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th international conference on Machine learning*, pages 160–167. ACM.

Custers, R. and Aarts, H. (2005). Positive Affect as Implicit Motivator: On the Nonconscious Operation of Behavioral Goals. *Journal of Personality and Social Psychology*, 89(2):129.

Dasgupta, S., Littman, M. L., and McAllester, D. (2001). PAC Generalization Bounds for Co-training. In *Neural Information Processing Systems*, pages 375–382.

Dawid, A. P. and Skene, A. M. (1979). Maximum Likelihood Estimation of Observer Error-Rates Using the EM Algorithm. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 28(1):20–28.

Dehghani, M., Gouws, S., Vinyals, O., Uszkoreit, J., and Kaiser, Ł. (2019). Universal Transformers. In *International Conference on Learning Representations*.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Dumoulin, V., Perez, E., Schucher, N., Strub, F., Vries, H. d., Courville, A., and Bengio, Y. (2018). Feature-wise Transformations. *Distill*. https://distill.pub/2018/feature-wise-transformations.

Elmore, J. G., Longton, G. M., Carney, P. A., Geller, B. M., Onega, T., Tosteson, A. N., Nelson, H. D., Pepe, M. S., Allison, K. H., Schnitt, S. J., et al. (2015). Diagnostic Concordance among Pathologists Interpreting Breast Biopsy Specimens. *Journal of the American Medical Association*, 313(11):1122–1132.

Fanselow, M. S. and Poulos, A. M. (2005). The Neuroscience of Mammalian Associative Learning. *Annu. Rev. Psychol.*, 56:207–234.

Felleman, D. J. and Van, D. E. (1991). Distributed Hierarchical Processing in the Primate Cerebral Cortex. *Cerebral cortex (New York, NY: 1991)*, 1(1):1–47.

Finn, C., Abbeel, P., and Levine, S. (2017). Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks. *arXiv preprint arXiv:1703.03400*.

Forrester, J. W. (1971). Counterintuitive Behavior of Social Systems. *Technological Forecasting and Social Change*, 3:1–22.

Franceschi, L., Frasconi, P., Salzo, S., and Pontil, M. (2018). Bilevel programming for hyperparameter optimization and meta-learning. *arXiv preprint arXiv:1806.04910*.

Frénay, B. and Verleysen, M. (2014). Classification in the Presence of Label Noise: A Survey. *IEEE Transactions on Neural Networks and Learning Systems*, 25(5):845–869.

Gervan, P., Berencsi, A., and Kovacs, I. (2011). Vision First? The Development of Primary Visual Cortical Networks is More Rapid than the Development of Primary Motor Networks in Humans. *PloS one*, 6(9).

Graves, A. (2016). Adaptive Computation Time for Recurrent Neural Networks. *arXiv preprint arXiv:1603.08983*.

Grover, A., Al-Shedivat, M., Gupta, J. K., Burda, Y., and Edwards, H. (2018). Learning Policy Representations in Multiagent Systems. *arXiv preprint arXiv:1806.06464*.

Gulshan, V., Peng, L., Coram, M., Stumpe, M. C., Wu, D., Narayanaswamy, A., Venugopalan, S., Widner, K., Madams, T., Cuadros, J., et al. (2016). Development and Validation of a Deep Learning Algorithm for Detection of Diabetic Retinopathy in Retinal Fundus Photographs. *Journal of the American Medical Association*, 316(22):2402–2410.

Ha, D., Dai, A., and Le, Q. V. (2018). HyperNetworks. In *International Conference on Learning Representations*.

Ha, D. and Schmidhuber, J. (2018). World Models. *arXiv preprint arXiv:1803.10122*.

Haarnoja, T., Pong, V., Zhou, A., Dalal, M., Abbeel, P., and Levine, S. (2018). Composable deep reinforcement learning for robotic manipulation. *arXiv preprint arXiv:1803.06773*.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.

Hofmann, T., Schölkopf, B., and Smola, A. J. (2008). Kernel Methods in Machine Learning. *The Annals of Statistics*, pages 1171–1220.

Hu, R., Andreas, J., Rohrbach, M., Darrell, T., and Saenko, K. (2017). Learning to reason: End-to-end module networks for visual question answering. *CoRR, abs/1704.05526*, 3.

Kaas, J. H. and Hackett, T. A. (1998). Subdivisions of AuditoryCortex and Levels of Processing in Primates. *Audiology and Neurotology*, 3(2-3):73–85.

Kahneman, D. and Egan, P. (2011). *Thiking, Fast and Slow*, volume 1. Farrar, Straus and Giroux New York.

Kaiser, L., Gomez, A. N., Shazeer, N., Vaswani, A., Parmar, N., Jones, L., and Uszkoreit, J. (2017). One Model to Learn them All. *arXiv preprint arXiv:1706.05137*.

Kearns, M. (1998). Efficient Noise-tolerant Learning from Statistical Queries. *Journal of the ACM (JACM)*, 45(6):983–1006.

Keller, G. B., Bonhoeffer, T., and Hübener, M. (2012). Sensorimotor Mismatch Signals in Primary Visual Cortex of the Behaving Mouse. *Neuron*, 74(5):809–815.

Khetan, A., Lipton, Z. C., and Anandkumar, A. (2017). Learning from Noisy Singly-Labeled Data. *arXiv preprint arXiv:1712.04577*.

Kingma, D. P. and Ba, J. (2014). Adam: A Method for Stochastic Optimization. *arXiv preprint arXiv:1412.6980*.

Laird, J. E. (2012). *The Soar Cognitive Architecture*. MIT Press.

Lane, H. and Tranel, B. (1971). The Lombard Sign and the Role of Hearing in Speech. *Journal of Speech and Hearing Research*, 14(4):677–709.

Lombard, E. (1911). Le signe de l'elevation de la voix. *Ann. Mal. de L'Oreille et du Larynx*, pages 101–119.

Madani, O., Pennock, D., and Flake, G. (2004). Co-Validation: Using Model Disagreement on Unlabeled Data to Validate Classification Algorithms. In *Neural Information Processing Systems*.

McCann, B., Keskar, N. S., Xiong, C., and Socher, R. (2018). The Natural Language Decathlon: Multitask Learning as Question Answering. *arXiv preprint arXiv:1806.08730*.

Mishkin, M., Ungerleider, L. G., and Macko, K. A. (1983). Object Vision and Spatial Vision: Two Cortical Pathways. *Trends in Neurosciences*, 6:414–417.

Mitchell, T. M., Cohen, W. W., Hruschka Jr, E. R., Pratim Talukdar, P., Betteridge, J., Carlson, A., Dalvi, B., Gardner, M., Kisiel, B., Krishnamurthy, J., Lao, N., Mazaitis, K., Mohamed, T. P., Nakashole, N., Platanios, E. A., Ritter, A., Samadi, M., Settles, B., Wang, R. C., Wijaya, D., Gupta, A., Chen, X., Saparov, A., Greaves, M., and Welling, J. (2018). Never-Ending Learning. *Communications of the ACM*, 61(5):103–115.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., and Kavukcuoglu, K. (2016). Asynchronous Methods for Deep Reinforcement Learning. In *International Conference on Machine Learning*, pages 1928–1937.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (2013). Playing Atari with Deep Reinforcement Learning. *arXiv preprint arXiv:1312.5602*.

Moreno, P. G., Artés-Rodríguez, A., Teh, Y. W., and Perez-Cruz, F. (2015). Bayesian Nonparametric Crowdsourcing. *Journal of Machine Learning Research*, 16.

Natarajan, N., Dhillon, I. S., Ravikumar, P. K., and Tewari, A. (2013). Learning with Noisy Labels. In *Advances in Neural Information Processing Systems*, pages 1196–1204.

Nettleton, D. F., Orriols-Puig, A., and Fornells, A. (2010). A Study of the Effect of Different Types of Noise on the Precision of Supervised Learning Techniques. *Artificial Intelligence Review*, 33(4):275–306.

Newell, A. (1990). *Unified Theories of Cognition*. Harvard University Press, Cambridge, MA, USA.

Nijhawan, R. (1994). Motion Extrapolation in Catching. *Nature*.

OpenAI et al. (2018). Learning dexterous in-hand manipulation. *arXiv preprint arXiv:1808.00177*.

Papies, E. K. (2016). Health Goal Priming as a Situated Intervention Tool: How to Benefit from Nonconscious Motivational Routes to Health Behaviour. *Health Psychology Review*, 10(4):408–424.

Parisi, F., Strino, F., Nadler, B., and Kluger, Y. (2014). Ranking and combining multiple predictors without labeled data. *Proceedings of the National Academy of Sciences*.

Parisotto, E., Ba, J. L., and Salakhutdinov, R. (2015). Actor-Mimic: Deep Multitask and Transfer Reinforcement Learning. *arXiv preprint arXiv:1511.06342*.

Peters, M. E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., and Zettlemoyer, L. (2018). Deep contextualized word representations. *arXiv preprint arXiv:1802.05365*.

Platanios, E. A., Al-Shedivat, M., Xing, E., and Mitchell, T. M. (2019). Learning from Multiple Noisy Labels. In *Review for Advances in Neural Information Processing Systems*.

Platanios, E. A., Blum, A., and Mitchell, T. M. (2014). Estimating Accuracy from Unlabeled Data. In *Conference on Uncertainty in Artificial Intelligence*, pages 1–10.

Platanios, E. A., Dubey, A., and Mitchell, T. M. (2016). Estimating Accuracy from Unlabeled Data: A Bayesian Approach. In *International Conference in Machine Learning*, pages 1416–1425.

Platanios, E. A., Poon, H., Horvitz, E., and Mitchell, T. M. (2017). Estimating Accuracy from Unlabeled Data: A Probabilistic Logic Approach. In *Advances in Neural Information Processing Systems*.

Platanios, E. A., Sachan, M., Neubig, G., and Mitchell, T. (2018). Contextual Parameter Generation for Universal Neural Machine Translation. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Brussels, Belgium.

Platanios*, E. A., Stretcu*, O., Stoica*, G., Poczos, B., and Mitchell, T. (2019). Contextual Parameter Generation for Question Answering. In *Annual Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*.

Rajpurkar, P., Jia, R., and Liang, P. (2018). Know What You Don't Know: Unanswerable Questions for SQuAD. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 784–789. Association for Computational Linguistics.

Ralph, M. A. L., Jefferies, E., Patterson, K., and Rogers, T. T. (2017). The Neural and Computational Bases of Semantic Cognition. *Nature Reviews Neuroscience*, 18(1):42.

Ranganath, C. and Ritchey, M. (2012). Two Cortical Systems for Memory-Guided Behaviour. *Nature Reviews Neuroscience*, 13(10):713.

Ratner, A., Bach, S. H., Ehrenberg, H., Fries, J., Wu, S., and Ré, C. (2017). Snorkel: Rapid Training Data Creation with Weak Supervision. *Proceedings of the VLDB Endowment*, 11(3):269–282.

Rauschecker, J. P. (1998). Cortical Processing of Complex Sounds. *Current opinion in neurobiology*, 8(4):516–521.

Reddi, S. J., Kale, S., and Kumar, S. (2018). On the Convergence of Adam and Beyond. In *International Conference on Learning Representations*.

Rogers, T. T., Ralph, L., Matthew, A., Garrard, P., Bozeat, S., McClelland, J. L., Hodges, J. R., and Patterson, K. (2004). Structure and Deterioration of Semantic Memory: A Neuropsychological and Computational Investigation. *Psychological Review*, 111(1):205.

Romanski, L. M., Bates, J. F., and Goldman-Rakic, P. S. (1999). Auditory Belt and Parabelt Projections to the Prefrontal Cortex in the Rhesus Monkey. *Journal of Comparative Neurology*, 403(2):141–157.

Samarakoon, S., Bennis, M., Saady, W., and Debbah, M. (2018). Distributed federated learning for ultra-reliable low-latency vehicular communications. *arXiv preprint arXiv:1807.08127*.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal Policy Optimization Algorithms. *arXiv preprint arXiv:1707.06347*.

Schuurmans, D., Southey, F., Wilkinson, D., and Guo, Y. (2006). Metric-Based Approaches for Semi-Supervised Regression and Classification. In *Semi-Supervised Learning*.

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., et al. (2017). Mastering the Game of Go without Human Knowledge. *Nature*, 550(7676):354.

Simonyan, K. and Zisserman, A. (2015). Very Deep Convolutional Networks for Large-Scale Image Recognition. In *International Conference on Learning Representations*.

Singer, Y., Teramoto, Y., Willmore, B. D., Schnupp, J. W., King, A. J., and Harper, N. S. (2018). Sensory Cortex is Optimized for Prediction of Future Input. *eLife*, 7:e31557.

Smith, V., Chiang, C.-K., Sanjabi, M., and Talwalkar, A. S. (2017). Federated multi-task learning. In *Advances in Neural Information Processing Systems*, pages 4424–4434.

Snell, J., Swersky, K., and Zemel, R. (2017). Prototypical Networks for Few-Shot Learning. In *Advances in Neural Information Processing Systems*, pages 4077–4087.

Sukhbaatar, S., Fergus, R., et al. (2016). Learning Multiagent Communication with Backpropagation. In *Advances in Neural Information Processing Systems*, pages 2244–2252.

Sukhbaatar, S., Weston, J., Fergus, R., et al. (2015). End-to-End Memory Networks. In *Advances in Neural Information Processing Systems*, pages 2440–2448.

Sutton, R. S., McAllester, D. A., Singh, S. P., and Mansour, Y. (2000). Policy Gradient Methods for Reinforcement Learning with Function Approximation. In *Advances in Neural Information Processing Systems*, pages 1057–1063.

Takarada, Y. and Nozaki, D. (2018). Motivational Goal-Priming With or Without Awareness Produces Faster and Stronger Force Exertion. *Scientific Reports*, 8(1):10135.

Teh, Y. W., Jordan, M. I., Beal, M. J., and Blei, D. M. (2005). Sharing Clusters Among Related Groups: Hierarchical Dirichlet Processes. In *Advances in Neural Information Processing Systems*, pages 1385–1392.

Thrun, S. and Pratt, L. (1998). *Learning to learn*. Springer.

Tian, T. and Zhu, J. (2015). Max-Margin Majority Voting for Learning from Crowds. In *Neural Information Processing Systems*.

Tran, D., Mike, D., van der Wilk, M., and Hafner, D. (2018). Bayesian Layers: A Module for Neural Network Uncertainty. *arXiv preprint arXiv:1812.03973*.

Tulving, E., Schacter, D. L., and Stark, H. A. (1982). Priming Effects in Word-Fragment Completion are Independent of Recognition Memory. *Journal of experimental psychology: learning, memory, and cognition*, 8(4):336.

Van Hasselt, H., Guez, A., and Silver, D. (2016). Deep Reinforcement Learning with Double Q-Learning. In *Thirtieth AAAI Conference on Artificial Intelligence*.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is All You Need. In *Advances in Neural Information Processing Systems*, pages 5998–6008.

Veltkamp, M., Aarts, H., and Custers, R. (2008). On the Emergence of Deprivation-Reducing Behaviors: Subliminal Priming of Behavior Representations turns Deprivation into Motivation. *Journal of Experimental Social Psychology*, 44(3):866–873.

Vinyals, O., Babuschkin, I., Chung, J., Mathieu, M., Jaderberg, M., Czarnecki, W. M., Dudzik, A., Huang, A., Georgiev, P., Powell, R., Ewalds, T., Horgan, D., Kroiss, M., Danihelka, I., Agapiou, J., Oh, J., Dalibard, V., Choi, D., Sifre, L., Sulsky, Y., Vezhnevets, S., Molloy, J., Cai, T., Budden, D., Paine, T., Gulcehre, C., Wang, Z., Pfaff, T., Pohlen, T., Wu, Y., Yogatama, D., Cohen, J., McKinney, K., Smith, O., Schaul, T., Lillicrap, T., Apps, C., Kavukcuoglu, K., Hassabis, D., and Silver, D. (2019). AlphaStar: Mastering the Real-Time Strategy Game StarCraft II. https://deepmind.com/blog/alphastar-mastering-real-time-strategy-game-starcraft-ii/.

Wang, Z., Bapst, V., Heess, N., Mnih, V., Munos, R., Kavukcuoglu, K., and de Freitas, N. (2017a). Sample Efficient Actor-Critic with Experience Replay. In *International Conference on Learning Representations*.

Wang, Z., Merel, J. S., Reed, S. E., de Freitas, N., Wayne, G., and Heess, N. (2017b). Robust Imitation of Diverse Behaviors. In *Advances in Neural Information Processing Systems*, pages 5320–5329.

Warren, J. D. and Griffiths, T. D. (2003). Distinct Mechanisms for Processing Spatial Sequences and Pitch Sequences in the Human Auditory Brain. *Journal of Neuroscience*, 23(13):5799–5804.

Watkins, C. J. and Dayan, P. (1992). Q-learning. *Machine learning*, 8(3-4):279–292.

Weingarten, E., Chen, Q., McAdams, M., Yi, J., Hepler, J., and Albarracín, D. (2016). From Primed Concepts to Action: A Meta-Analysis of the Behavioral Effects of Incidentally Presented Words. *Psychological Bulletin*, 142(5):472.

Wessinger, C., VanMeter, J., Tian, B., Van Lare, J., Pekar, J., and Rauschecker, J. P. (2001). Hierarchical Organization of the Human Auditory Cortex Revealed by Functional Magnetic Resonance Imaging. *Journal of cognitive neuroscience*, 13(1):1–7.

Yosinski, J., Clune, J., Bengio, Y., and Lipson, H. (2014). How transferable are features in deep neural networks? In *Advances in neural information processing systems*, pages 3320–3328.

Zatorre, R. J. and Belin, P. (2001). Spectral and Temporal Processing in Human Auditory Cortex. *Cerebral Cortex*, 11(10):946–953.

Zatorre, R. J., Bouffard, M., and Belin, P. (2004). Sensitivity to Auditory Object Features in Human Temporal Neocortex. *Journal of Neuroscience*, 24(14):3637–3642.

Zhou, D., Liu, Q., Platt, J. C., Meek, C., and Shah, N. B. (2015). Regularized Minimax Conditional Entropy for Crowdsourcing. *CoRR*, abs/1503.07240.