Neural Networks 139 (2021) 294–304

Contents lists available at ScienceDirect

Neural Networks

journal homepage: www.elsevier.com/locate/neunet

Design and independent training of composable and reusable neural modules

David Castillo-Bolado *, Cayetano Guerra-Artal, Mario Hernández-Tejera

SIANI Institute and Department of Computer Science, University of Las Palmas de Gran Canaria, Campus Universitario de Tafira, 35017 Las Palmas de Gran Canaria, Spain

ARTICLE INFO

Article history:
Received 14 June 2020
Received in revised form 17 March 2021
Accepted 22 March 2021
Available online 1 April 2021

Keywords:
Modular training
Modularity
Compositionality
Methodology
Learning by role
Visual Question Answering

ABSTRACT

Monolithic neural networks and end-to-end training have become the dominating trend in the field of deep learning, but the steady increase in complexity and training costs has raised concerns about the effectiveness and efficiency of this approach. We propose modular training as an alternative strategy for building modular neural networks by composing neural modules that can be trained independently and then kept for future use. We analyse the requirements and challenges regarding modularity and compositionality and, with that information in hand, we provide a detailed design and implementation guideline. We show experimental results of applying this modular approach to a Visual Question Answering (VQA) task parting from a previously published modular network and we evaluate its impact on the final performance, with respect to a baseline trained end-to-end. We also perform compositionality tests on CLEVR.

1. Introduction

Deep learning has been demonstrated to be a powerful part of machine learning, enabling the automatic discovery of complex patterns in data and as a result finding solution to problems that had previously been considered very difficult to solve or computationally unfeasible. As the research community teases the limits of this approach, the predominant trend is to design and train new monolithic neural networks for each new task, conducting the training in an end-to-end fashion.

The steady increase in complexity of these tasks has made the amount of resources invested in training neural networks a growing concern. A recent work of Strubell et al. (2019) analyses the training cost of state-of-the-art NLP models and reports CO_2 emissions equivalent to those of a trans-American flight. It also points out that the training cost represents only a little fraction of the total development cost, as the greater part falls into hyper-parameter optimization. This not only damages the environment, but also imposes access and creativity barriers to underfunded researchers. Therefore, prioritizing the computational efficiency over brute performance has often been recommended.

Guidelines of software engineering encourage the design and development of modular systems, which favour understanding, maintainability and reusability: this seems to be the key towards generalization in neural networks as well (see Bahdanau et al., 2019; Cai et al., 2017). In this paper, we elaborate on the meaning of modularity and compositionality within neural networks and how they can be leveraged to build neural modules that can be trained independently and assembled, while showing a highly compositional behaviour. Our main contributions are as follows:

- Taxonomy of compositional dependencies in the context of independent modular training and their corresponding solutions.
- Methodology for the implementation of modular interfaces with neural data types.
- Definition of *Learning by Role* as an essential mechanism for injecting behaviour into modules.
- Introduction of surrogate gradient modules for training neural modules under indirect supervision.
- Methodological recipes for making a modular neural network meet compositionality requirements.

We exemplify and validate all points mentioned above on a case study focused in the Visual Question Answering task (VQA, Antol et al., 2015), based on Neural Module Networks (NMN, Andreas et al., 2016). We also perform further compositionality tests on the synthetic CLEVR data set (Johnson et al., 2017).

The paper is organized as follows. In Section 2 we introduce modularity and compositionality for neural networks. Section 3 describes the modular training approach. In Section 4 we comment on previous work related to our own. Experimental results are presented in Section 5. In Section 6 we show our conclusions and we discuss implications and future work.

* Corresponding author.
E-mail addresses: david.castillo103@alu.ulpgc.es (D. Castillo-Bolado), cayetano.guerra@ulpgc.es (C. Guerra-Artal), mario.hernandez@ulpgc.es (M. Hernández-Tejera).

https://doi.org/10.1016/j.neunet.2021.03.034
0893-6080/

2. Modularity and compositionality

Modularity and compositionality are two different concepts, closely related, but whose implications depend heavily on the context in which they appear. As a result, in the field of deep learning, there has been certain amount of confusion in this regard and there has also been an important debate about the actual meaning of compositionality within neural networks (Hupkes et al., 2020). We provide a definition of *modularity* and *compositionality* as they should be understood within this article.

Modularity is a structural property that states that a system is composed of several parts, each one with its own functionality, with the purpose of easing implementation and maintenance. On the other hand, compositionality is a functional property that describes how well these parts can be rearranged while maintaining their original purpose to produce regular and predictable outcomes.

Artificial neural networks are modular in their structure and modern implementation of common neural layers and architectures benefit from that aspect in order to provide frameworks that are both easy to use and maintain (Abadi et al., 2015; Jia et al., 2014; Paszke et al., 2019). However, this kind of modularity does not transfer by any means into compositional behaviour. Montague (1970) defines compositionality as the algebraic capacity to understand and produce novel combinations from known components.

Good software design rules of high cohesion and low coupling can therefore be extended to differentiable programming, being the design and optimization the main sources of cohesion and coupling, respectively. Modules trained jointly might overfit to all kinds of interrelations and give raise to undesired emergent behaviours (Jacobs, 1990), preventing the compositional behaviour when assembled in layouts not seen during training. In this sense, there exists a family of neural architectures that, being built with functional modularity in mind, seek compositionality by design (see Socher et al., 2013): NMN define a set of neural modules, with possibly different module architectures and capacities, which are assembled in different layouts depending on the instance. Modules are then trained using backpropagation through structure (Goller & Kuchler, 1996). It is within this context and for this family of neural architectures that we propose this methodology.

In addition, modularity paves the way for a systematic exploitation of the divide and conquer strategy, solving complex tasks gradually and focusing efforts and resources in a more efficient way. The practice of reusing components, very common in software engineering, but still poorly exploited within neural networks, would extend further than the mere application of transfer learning to embedding layers. Modular and composable neural networks enable a new fully modular paradigm, where modules can be reused in all sorts of tasks, stimulating the creation of neural module libraries and even allowing to build new neural networks without the need for conducting end-to-end training.

How to train such modules is still an important challenge, however. At first glance, it is not easy task to find a general training recipe, therefore it is highly relevant to establish a methodology that facilitates the spread and adoption of the practice.

3. Modular training

In contrast to the well-known joint training process, modular training aims to independently train neural modules 1. The goal is to be able to train any module in the network, regardless of its role or interactions with other modules, in a way that all the advantages of the compositional modularity can be exploited. Ideally, there would be a data set for each module, where inputs are paired with their corresponding expected outputs, but this is rarely the case and keeping to the fully supervised training involves the end-to-end training of multi-module assemblies. Although this latter option is a valid strategy for avoiding monolithic end-to-end training, there are further factors to take into account.

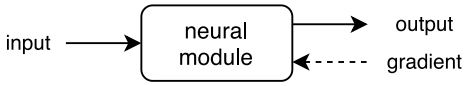

Fig. 1. Generic depiction of a neural module, with an input, output and a learning signal in the form of gradient.

It is well known that neural networks can overfit to spurious patterns in data and even interactions between different parts of the network (Novak et al., 2018). We aim to prevent such undesired emergent behaviour – or coupling – by identifying input dependencies and availability of supervision signal.

3.1. Compositional dependencies

We have identified five main types of cases regarding dependencies that may appear during modular training. Two of them are related to the availability of input data, two others have to do with the quality of the gradient reaching the module and the fifth case is a dependency that arises from the interaction between different outputs.

- **Decoupled input**. The module's input does not depend on other modules, nor can it be obtained independently. This is often the case of input modules, which receive raw data directly from the data set and are the first operation in the pipeline.
- **Dependent input**. The module's input comes from other modules and cannot be obtained by other means. This creates a sequential dependency, in a way that the corresponding input modules must be trained beforehand.
- **Direct supervision**. There is a loss function that can be computed directly over the module's output in order to guide the training, either because the module gives the final output, there are traces available or there is an ad-hoc or heuristic training loss.
- **Indirect supervision**. There is insufficient learning signal or no direct connection between the module's output and a loss function. In a joint training scenario, this would typically be the case of non-final modules, which commonly receive gradient through the output modules.
- **Codependent gradient**. The module's output interacts with other output values in a way that cannot be characterized as ensemble averaging. This means that a non-trivial interaction arises, which affects the values returned by the loss function and therefore, the function learned by each module taking part in the interaction.

Fig. 2 shows an example of a modular neural network – formed by an assembly of four distinct neural modules – in which all five mentioned scenarios occur. In this example, we suppose having a data set for the task that is carried out by the entire assembly, mapping inputs to expected outputs or labels. That being the case, modules with decoupled input (A and C) do not require any special treatment in this regard and can use the input *as is* from the dataset. Modules that provide the final output or contribute to it in an ensemble-like fashion (D) are said to have direct supervision and can receive a good learning signal in the form of gradient, which may just be computed as any kind of loss function over the output and its corresponding label.

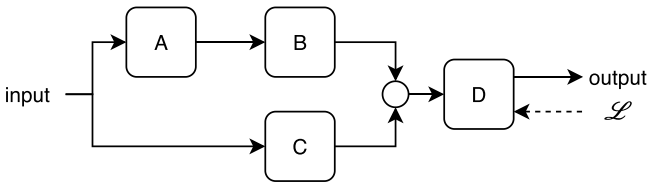

Fig. 2. Example arrangement of four distinct neural modules, where A and C receive decoupled input and only D has direct supervision. Module B has neither one nor the other. Outputs of B and C may interact before going into D.

However, cases of dependent input and indirect supervision do require special consideration. Modules that take other modules' output as input introduce sequential dependencies during training and cannot be learned until those dependencies are solved (B depends on A, and D on B and C). For such cases, we propose the creation of intermediate data sets that represent the relation between those intermediate outputs and the labels, thus avoiding the need for executing input modules after they have been trained.

In contrast, modules A, B and C may present cases of indirect supervision, and these are not straightforward to solve. In the absence of any kind of loss function for the module being trained, an alternative loss or task must be provided in order to generate some kind of gradient that is capable of guiding the module's parameters to a valid configuration (weak or self supervision is an option). In Section 3.3 we propose the leverage of an auxiliary module to enable modular training in an indirect supervision scenario whilst keeping the compositional properties of the module being trained.

Cases of output interdependence may arise anywhere where two or more outputs are put together (before input of module D). In a best case scenario, outputs of B and C are concatenated or expected to be parts of a redundant ensemble, thus enabling independent training. In a worst case scenario, outputs would be so entangled that both modules should be trained jointly. Between these two extremes, sequential training can be one of many tools to be leveraged (see case study in Section 5.5).

3.2. Modular interfaces and neural data types

Well-defined interfaces ease scaling and addition of functionality without interfering with existing functions or forcing a redesign of the system. In this section we will comment on the desired qualities of a neural data type in order to avoid inter-modular dependence and favour compositionality and generalization.

Although modular training prevents inter-modular coupling, which is expected to appear in an end-to-end training, the choice of data representation can still give raise to input dependencies if modules are trained sequentially. Modules may overfit to input centring, scale or other manifold-related aspects (Novak et al., 2018).

Differently from the ordinary practice, we aim to achieve compositionality when transferring the modules to new architectures too. In order to provide guarantees of compositionality, module inputs and outputs cannot depend on the module implementation, but on a shared specification. Unlike previously published neural modules, which produce non-typed representation vectors, a compositional neural module must output meaningful data that is consistent with the module's task and the use that will be made of it. In this regard, the neural data type must specify – to the fullest extent possible – the output domain and its expected manifold or distribution.

Finally, the universal approximation theorem (Hanin & Sellke, 2017; Lu et al., 2017) states that neural networks work arbitrarily well at interpolating, but provide no guarantees in the extrapolation regime. Bounded input values help keeping the network in the interpolation regime and avoid off-distribution issues (Gleave et al., 2020). One of the best examples of this quality is presented by Cai et al. (2017), where they give generalization guarantees for the first time in the field of neural networks. This motivates us to propose additional constraints related to boundedness.

Approximations to neural data types have been leveraged in the past without explicit consideration; for example in the gating units of Long Short-Term Memory networks (LSTM, Hochreiter & Schmidhuber, 1997) or flags in general (Reed & De Freitas, 2015), softmax outputs interpreted as probability mass functions or attention mechanisms and latent variables in generative adversarial networks (Brock et al., 2019, Appendix E). Among these cases we find output values confined to a bounded space, with a well-defined meaning (gate or probability mass distribution) and sometimes further manifold related restrictions (values adding up to 1 or coming from a Bernoulli distribution).

Implementation of neural data types therefore is to be done at every module's output and it can be achieved via two different types of constraints:

- Domain related or hard constraints: they set the range of the output values via clipping or activation functions. All output values will therefore remain in the domain by definition.
- Semantic or soft constraints: they determine the expected manifold for the output tensor. Some semantic constraints may be implemented as activation layers like softmax or normalization layers, but usually no guarantee can be given that output data will lie in the expected manifold. In such cases, the module is encouraged to comply with the semantic constraints through a training loss, either from labelled data, a heuristic loss, a discriminator or any other kind of auxiliary training module (see Section 3.3).

Differentiability is an additional requirement, given that we aim to train the module with gradient descent. However, this aspect of the data type is temporal and mainly useful during training, as there is a priori no need to propagate gradients through modules afterwards. For later modular use, generalization might be favoured by exploiting discrete or quantized output values.

3.3. The surrogate gradient module

In a scenario of indirect supervision, where insufficient or no direct supervision is available, we hypothesize that an effective training gradient can be obtained by letting an auxiliary module bridge the gap between the module's output and the nearest loss function (see Fig. 3). We call it the *surrogate gradient module*, as its goal is to help train the module independently by approximating the gradient needed for that matter, using the least amount of computational resources possible.

Surrogate models are commonly used in science and engineering for approximating the input–output behaviour of an expensive simulation or a system that cannot be easily measured (Jin, 2011; Queipo et al., 2005). For our part, we think of the surrogate gradient module as a way of approximating the gradient that a module would receive during an end-to-end training. From this premise, in our case, the incentive is not to model the actual input–output behaviour of the neural network between the module and the supervised point, but rather to model its input-gradient behaviour.

This goal is supported by the following desiderata: (i) availability of gradient at the module's output; (ii) compliance of the

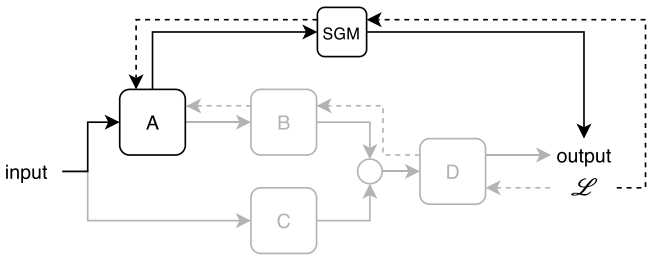

Fig. 3. Example application of the surrogate gradient module, placed to bridge the gap left by modules B and D. The output given by the surrogate gradient module matches the original output data type.

trained module's output with the module's target data type, lying in the desired domain and manifold; (iii) final behaviour is as expected, as if the module had learned in an end-to-end fashion. Note that if we were only interested in independent training, the point (ii) would be useless, but it is of high relevance for obtaining a generalizing compositional behaviour (Cai et al., 2017; Lu et al., 2017).

We envision the surrogate gradient module as a minimalist surrogate architecture that processes the trained module's output and connects it with the next supervised point available in a meaningful way. Its purpose is not to achieve the best performance in the final end-to-end task, but rather to be the least complex pipeline representing the output data type's restrictions and relation to the supervised value. From this premise, we expect that such an auxiliary module may make the trained module converge to a point close to its optimal configuration. We base this vision on the ability of learning by role.

3.3.1. Learning by role

Learning by role is an emergent learning phenomenon that has frequently, yet quietly been showcased in the literature. Andreas et al. (2016) described it as when a part of the neural network *acquires its behaviour as a byproduct of the end-to-end training procedure*. We therefore have named it after *Role Play*, the equivalent technique from the field of language learning (Ladousse & Maley, 1987), where students are given a simulated scenario to develop a set of skills within a safe environment.

According to learning by role, each part of the network learns to do its best within the realm of scenarios it is allowed to express itself in. Early examples could be seen in convolutional neural networks, where the detection tasks that each unit can learn is determined by the features available at each layer (Yosinski et al., 2015). This purely functional role contrasts with other early cases, like forget gates in LSTM, where the interaction between these units and the memory is carefully designed and acts as a hard constraint, limiting the function domain to forgetting-related tasks (Hochreiter & Schmidhuber, 1997). Later on, generative adversarial networks showed that learning by role could go further by combining both domain and semantic role constraints, this time in the form of the generator's architecture and a discriminator, respectively (Goodfellow et al., 2014). However, one of the latest examples of learning by role in modular networks is presented in NMN, in which modules are not only designed to perform well in certain tasks, but also converge to the target behaviour as a consequence of the place they take in the network layout and the samples they are used for (Andreas et al., 2016; Hu et al., 2017).

3.3.2. Design considerations

Although the surrogate gradient module is not conceived to achieve any useful degree of task performance, we hypothesize that we may interpret its loss or accuracy as a relative improvement indicator. We provide an empirical proof of this hypothesis in Section 5.3. Nonetheless, it is also important to determine if the value given by the surrogate gradient module can be trusted. This is mainly an issue in the early stages of training or in the case of poorly configured hyperparameters, when the surrogate gradient module might be brittle, or not trained well enough. That is why we recommend computing its predictive uncertainty during validation, as done by Gal and Ghahramani (2016).

The design of a surrogate gradient module is an engineering task that has to be done ad-hoc for each case. Luckily, domain constraints and maybe some semantic constraints are already implemented as part of the output data type, but even so, the designer must consider the targeted role of the neural module, as well as its output specification and how the neural module's output relates to the supervised point. However, due to the uniqueness of this task, we can only provide general indications for its design:

- The architecture has to transform the neural module's output into the supervised data.
- It must not do it in a random way, as doing so would miss the point and would probably drive it to overfitting the surrogate task.
- The transformation has to make sense and be consistent with both output and supervised data types. Supervised data type must be built following the rules in Section 3.2.
- All qualities of the module's output data type should take part in the transformation in order to make all of them participate into the learning by role and contribute to shaping the data type.
- The surrogate gradient module will be discarded after training, therefore it is recommended to make it lightweight in order to spare resources.

4. Related work

Traditional understanding of modularity in neural networks has been centred on ensembling methods, considering several ways to partition the input space for the sake of redundancy, or delegating partitions among modules (Chen, 2015). Moreover, this notion of modularity does not consider functional compositionality.

Modern approaches extend the application of modularity to the functional level, building in this way a modular architecture or pipeline where compositional behaviour naturally emerges. A common scheme is to divide the architecture into several functional modules and a controller. Most memory networks rely on this scheme, having a controller making use of handcrafted memory modules like a stack (Grefenstette et al., 2015; Joulin & Mikolov, 2015), a tape (Kurach et al., 2015) or an associative memory (Danihelka et al., 2016; Graves et al., 2014, 2016). Aside from this, Véniat et al. (2019) uses a controller for dynamically adjusting the network complexity and Kirsch et al. (2018) leverages it for reusing modules across layers, thus working like a mixture of experts (Jacobs et al., 1991) in which input and output spaces are equally sized. However, all these networks conceive modules with a very low cohesion and high coupling. More complex and heterogeneous layouts can be found in Andreas et al. (2016). There are also domains in which the layout is basically given by the problem instance, often in tree shape and with a single module instance (Silver et al., 2016; Socher et al., 2013). These examples set a strong foundation and serve as inspiration to our research, although they all train the modules jointly in an end-to-end fashion.

Modular training has been previously performed by Cai et al. (2017), providing guarantees of perfect generalization for a Neural Programmer Interpreter (NPI, Reed & De Freitas, 2015) by means of a recursive approach and a fully supervised modular training. A later study by Castillo-Bolado et al. (2019) analyses training statistics and generalization capabilities of a modularly trained neural network, it also gives hints for designing modular neural networks. Both rely on the availability of traces for conducting a fully supervised training and have been tested on synthetic tasks. Synthetic gradients (Jaderberg et al., 2017) enable asynchronous training of blocks in a neural network, thus achieving a certain degree of decoupling, but they are not intended to work as a completely decoupled training method. Gupta et al. (2020) highlight the difficulties of jointly training a highly nonlinear modular network, they leverage auxiliary losses and heuristics and suggest the pretraining and reusing of modules.

Models intended to solve VQA tasks usually rely on architectures with a high degree of modularity, in which composition is favoured and often pre-trained modules come into play (see Hu et al., 2017; Yu et al., 2019). In fact, Yu et al. (2019) won the first place in the VQA Challenge 2019 with a modular architecture and both two first models use a pre-trained BERT model (Devlin et al., 2018) as a text feature extractor. Nevertheless, the majority of the modular architecture is still trained end-to-end and the role of the pretrained modules is merely to generate a rich representation of the data. Modules are also designed with very low cohesion and high coupling, as so is more suitable to train end-to-end. Singh et al. (2018) provide a modular framework for vision and language multimodal research, but it is intended for bootstrapping design and it relies on joint end-to-end training and posterior ensembling.

5. Experiments

Our experiments are focused in testing the feasibility of the modular training approach on an existing modular neural network and measuring the impact it has on several training and performance statistics. We base our case study in the first implementation of NMNs, which was designed for solving the VQA v1 task.

We compare the network trained end-to-end to two different approaches to modular training: with and without adjusting the original design. We run a random hyperparameter search for the end-to-end network, as well as for every neural module, consisting of 50 evaluations each. Moreover, we analyse these results to assess the performance of distinct modules of the network and discuss its relevance towards the final accuracy.

Finally, we adapt the VQA v1 implementation to the CLEVR data set in order to conduct compositionality tests. The code for replicating the experiments can be found at https://github.com/dcasbol/dnmn.

5.1. Case study: Neural module networks and VQA

Neural Module Networks (NMN) is a class of neural architectures introduced by Andreas et al. (2016) in which the neural network is formed by a set of composable neural modules that are assembled specifically for each input sample. These modules are designed with different architectures depending on their expected functionality, but they also learn different functions by-role as a consequence of the end-to-end training (example layout in Fig. 4).

For this paper, we wrote a PyTorch (Paszke et al., 2019) implementation based on that original version and we took it as a starting point for implementing and testing our modular approach.

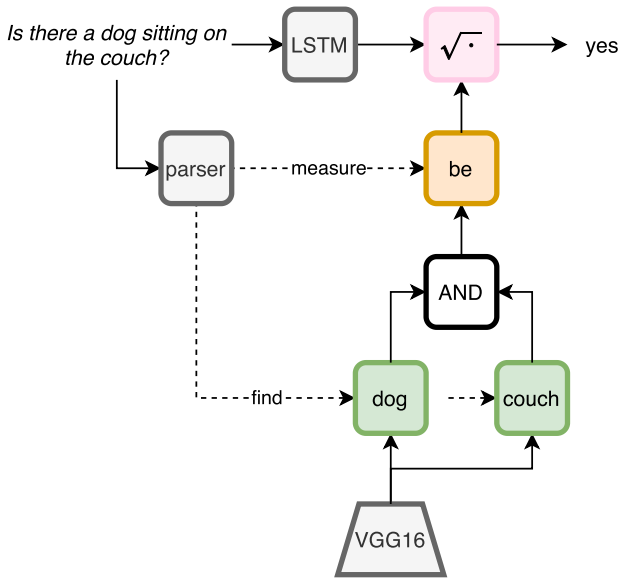

Fig. 4. Original NMN architecture. The sentence goes through a parser, which determines the module layout to use and their instances. An LSTM processes the sentence separately and both answers are combined via geometric averaging.

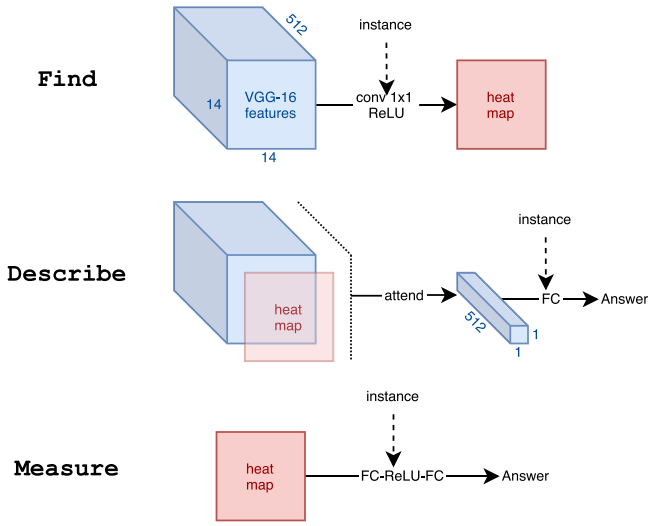

Fig. 5. Schematic depiction of NMN's composable modules. The module's weights are distinct for each instance and the attention mechanism is based on multiplication and weighted average.

The input question goes through the Stanford Parser (Klein & Manning, 2003) and the representation obtained is used to determine the network layout. Image features are extracted from a VGG16 (Simonyan & Zisserman, 2014) network pre-trained on ImageNet (Deng et al., 2009) and the `QuestionEncoder` is implemented by an LSTM (Hochreiter & Schmidhuber, 1997) that maps the question to a distribution over possible answers. A basic illustration of the other module's implementation can be seen in Fig. 5. In order to gain flexibility we do all tests locally, taking part of the official validation set as test set. We also provide official test results for the final models.

We take the NMN trained in an end-to-end fashion as our baseline, registering an accuracy of 51.49% on the official test server.

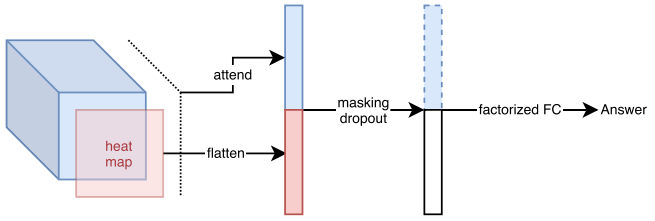

Fig. 6. Surrogate gradient module used to train the `Find` module. Input is processed in an equivalent way as in the full NMN, although with a single factorized linear matrix. Dropout is leveraged for computing predictive uncertainty.

5.2. Implementing a surrogate gradient module

The case study presents a typical case of indirect supervision when training the `Find` module. Its input comes directly from the pre-trained VGG16 network, but there is no loss function that we can directly apply over the module's output.

Instead of generating training labels or leveraging other training techniques with low supervision requirements, we can implement here a surrogate gradient module. We also impose a low-complexity constraint for proving that no high end-to-end performance is required to approximate the gradient (see Fig. 6). We accomplished this by factorizing the weight matrix of the fully connected layer into two smaller matrices. Being $W \in \mathbb{R}^{N\times M}$, we let it be built as a multiplication of two matrices $W_A \in \mathbb{R}^{N\times L}$ and $W_B \in \mathbb{R}^{L\times M}$, where $L \ll \min(N, M)$.

The surrogate gradient module helps training the neural module by imposing restrictions over the neural module's output that are equivalent to those encountered during the end-to-end training. In other words, the surrogate gradient module must consider data types to make a meaningful use of the module's output.

In the particular case of the module `Find`, the original output data type is a *heatmap of unnormalized attention*. That is, a 2D map of positive unnormalized values that are intended to function as an attention map. In the original implementation, this output may be used directly to answer yes/no questions or as an input for an attention mechanism over the VGG16 features. The surrogate gradient module has to mimic this behaviour if we expect the neural module to work well when fitted into the full architecture.

5.3. Validation of the surrogate gradient module

In Section 3.3 we propose the use of a surrogate gradient module to train a neural module in a scenario of indirect supervision and we hypothesize that the loss value obtained from the surrogate gradient module is an indicator of the actual module's performance. In order to validate this hypothesis, we have measured the correlation between the mean validation losses of the surrogate gradient module and the complete network.

We designed this experiment in three parts. First, we used the surrogate gradient method to train $N = 100$ `Find` modules with randomly chosen hyperparameters and for a randomly set number of epochs. We thus generated N neural modules with different degrees of performance and recorded their surrogate loss and the predictive variance during validation.

Secondly, from this set of trained modules, we discarded all modules that resulted in a predictive variance over 0.0025 on the output softmax domain ($2\sigma = 0.1$), which denotes that their behaviour is not very trustworthy, and from the remaining subset, we selected the modules exhibiting the lowest variance, trying to uniformly cover the range of validation losses (blue dots in Fig. 7).

Finally, and for each `Find` module in the selected subset, we transferred the weights to the end-to-end model and we trained it jointly while keeping the pre-trained weights frozen. This allowed forming a plot where surrogate loss and final loss are put together (Fig. 8). Apart from some degree of noise, naturally expected from random batching and gradient descent, the plot shows a correlation between both measures, which supports our hypothesis and opens the door to using the surrogate gradient module not only for conducting module training, but also for module selection. The correlation found has a Pearson correlation coefficient of 0.9 with a p-value of 10^{-14}.

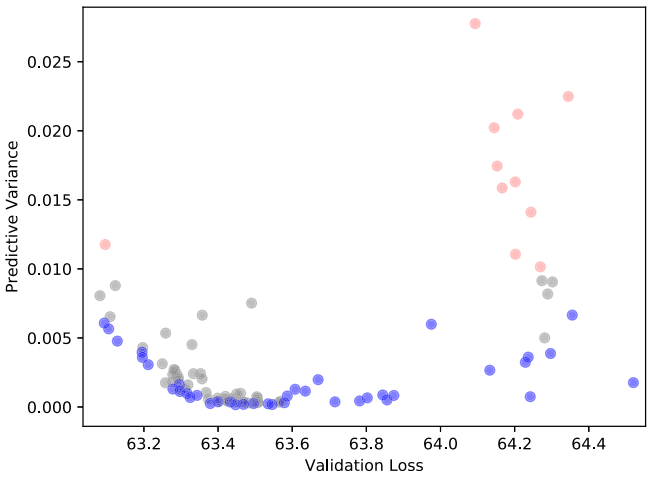

Fig. 7. Representation of all the randomly trained `Find` modules. For each one we show the mean loss of the surrogate gradient module over the validation set and its prediction variance. Modules in red were a priori discarded due to high uncertainty. Modules selected for the experiment are shown in blue. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

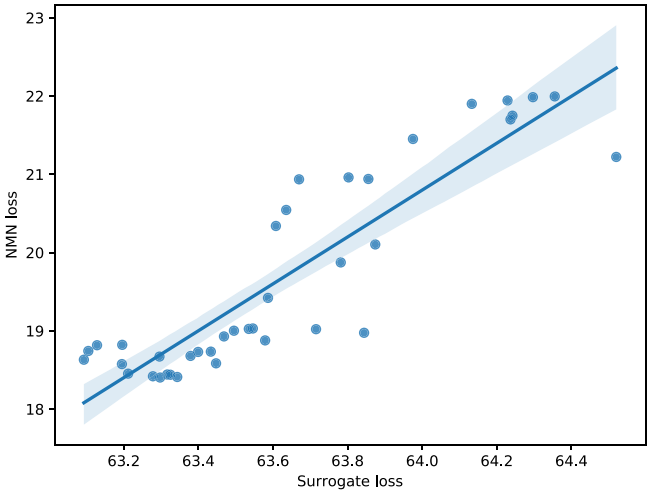

Fig. 8. Correlation plot between surrogate module and final assembly loss. The line shows the correlation found and shadowed area represents the 95% confidence interval for the regression line.

5.4. Direct modular training

A modular training of the NMN can be done just by paying attention to the compositional dependencies (Section 3.1) and solving them as indicated by the guidelines provided. In this particular case, we find a variety of input and supervision dependencies, the `QuestionEncoder` being the easiest module to train and needing a surrogate gradient module for training the `Find` module. All dependencies found are described in Table 1.

Table 1
Compositional dependencies found in each module of NMN.

	Input		Supervision	
Module	Decoupled	Dependent	Direct	Indirect
`Encoder`	✓		✓	
`Find`	✓			✓
`Describe`		✓	✓	
`Measure`		✓	✓	

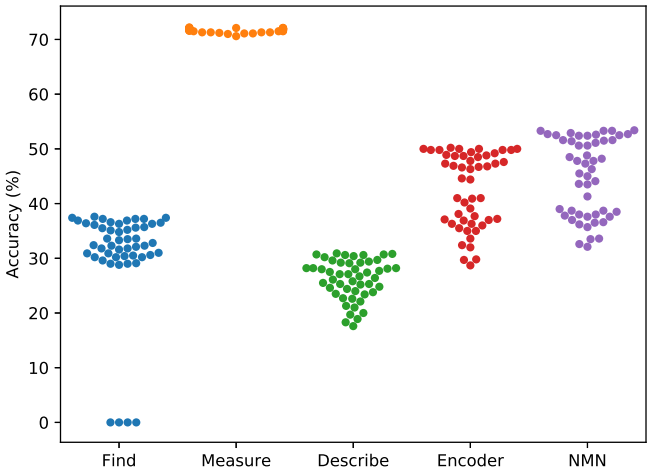

Fig. 9. Accuracy obtained for every modular and end-to-end evaluation during hyperparameter optimization.

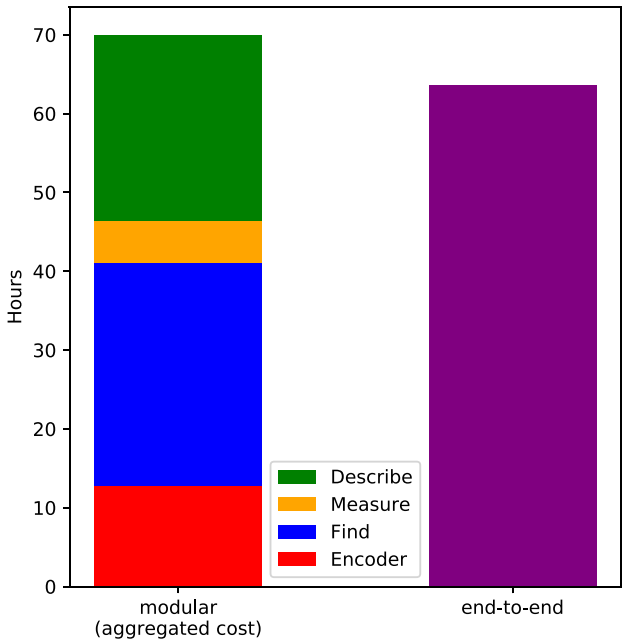

Fig. 10. Total training times during hyperparameter optimization. End-to-end training has a constant cost, while the cost for the modular approach is commonly a fraction of the aggregated costs, depending on the availability of pre-trained modules and what modules have been added or redesigned.

Moreover, as we will train the network from scratch, `Find` must be trained before `Describe` and `Measure`. Note that this only occurs because there is no pretrained `Find` module beforehand and root modules have an input dependency on `Find`. Having to train all modules in a modular network is actually an extreme case, as it only serves as a bootstrapping step if there is no module library or a previous version of the network.

This sequential dependency is mitigated by the creation of a cached virtual dataset, storing resulting heat maps and attended vectors for each sample in the original VQA dataset, thus avoiding the execution of the `Find` module during the training of the modules `Describe` and `Measure`.

We have optimized batch size, learning rate, weight decay and dropout rate independently for each module and taken the best performing configuration in each case. The final assembly achieves a test accuracy of 54.49%, getting very close to the baseline (see Table 2).

As an additional benefit of modularity, we can inspect this result in detail, obtaining insight about each module's performance and its susceptibility to hyperparameter configurations. In Fig. 9 we show the accuracy obtained at each evaluation for each module and also for the end-to-end version of the NMN. Note that the results for the `Find` module are those given by its surrogate gradient module and we can even see some of them at 0% accuracy, which correspond to configurations that did not surpass the minimum threshold we had set for predictive uncertainty.

Yes/no questions are assigned to `Measure`, which make a 35.81% of the data set, and the remaining 64.19% to `Describe`. This kind of information is very useful towards increasing the efficiency of the hyperparameter search – focusing the resources in more sensible modules – or for conducting an informative redesign of the network's architecture. In this case, results would point to considering changes to the `Describe` module.

In Fig. 10 we show the total time invested for both modular training and the end-to-end baseline. We exclude the time for generating the virtual dataset because it is negligible — less than 5 min, once after module `Find` is ready. The extra time required for the worst scenario in modular training represents roughly a 10% of the time needed for the end-to-end baseline. It is therefore safe to say that modular training will almost always require less time than training end-to-end.

In this very particular case, where the entire model fits in memory and there is no pre-trained module at hand, it is probably better to go for an end-to-end training. On the other hand, having to fit only one module in GPU means two things: (1) it is possible to train the module with less expensive equipment and (2) the remaining memory can be used to increase module capacity or batch size. This factor has been left out in this section in order to keep it simple, but it will be exploited in the next section.

5.5. Adjusted modular training

While conducting modular training without making any changes to the network is possible, in order to obtain higher degrees of compositionality and reusability, we recommend designing proper modular interfaces and implementing neural data types (see Section 3.2). This procedure includes redesigning some modules in order to make them better reflect their desired behaviour and increase their utility.

Regarding this latter aspect, Andreas et al. (2016) justify the use of the `QuestionEncoder` pointing to the aggressive simplifications made by the parser and the intention to model syntactic and semantic regularities in the data. This implies that the `QuestionEncoder` is intended to learn some sort of common sense. However, they implement it as a mask over answers, while it could be implemented as a prior that is overruled by new evidence provided by other modules. Merging answers through geometric average may actually hurt generalization as it will block out any evidence that contradicts training data and introduces a strong gradient codependency. We thus have substituted the geometric average by the most common implementation of

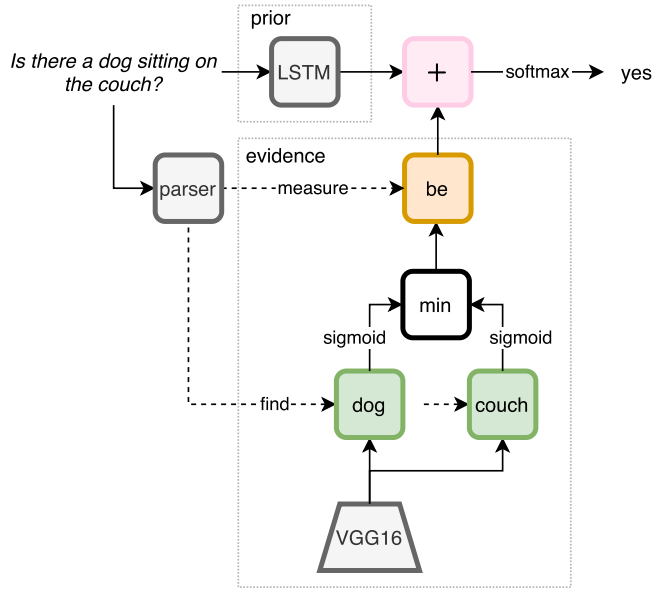

Fig. 11. Modified NMN architecture for improved compositionality. LSTM now provides a prior to the other branch and `Find` generates bounded soft masks, thus enabling the use of the minimum as the AND operator.

a prior in neural networks: an additive bias. `QuestionEncoder` provides this way an actual prior for the other branch of the network to work on, delegating to other modules the gathering of evidence from visual data (see Fig. 11). This new way of merging the outputs reduces the gradient codependency, but still the prior given by the `QuestionEncoder` alters the target function of the other branch, so it must be trained foremost. We have also created an additional virtual dataset for caching these prior logits and using them as a bias term for `Measure`, `Describe` and `Find`'s surrogate gradient module.

Moving on to data types, the original implementation of the `Find` module relies on *heatmaps or unnormalized attention*, which are good enough for an end-to-end training but enter in conflict with the boundedness criteria introduced in Section 3.2. We have therefore substituted the ReLU activation function for a sigmoid, bounding so the value per pixel to the range (0, 1) and giving soft masks as output instead of heat maps. This also enables the use of the minimum and maximum operations as alternative implementations of the AND and OR operators. These implementations preserve the data type of the input, thus avoiding extrapolation issues and favouring generalization.

Additionally, in Section 5.4 we comment that modular optimization enables making use of the extra GPU memory in more trainable parameters or greater batch sizes. We have taken this opportunity to benefit from the combinatorial advantages of modular hyperparameter exploration. We explored batch sizes up to 4 times greater than in the end-to-end approach and also some other hyperparameters specific to certain modules: whether to use a bias in `Find` or not, softmax or weighted average as attention mechanism, different embedding sizes, number of hidden units and dropout rates for `QuestionEncoder` and `Measure`.

In Fig. 12 we show the accuracies on the validation set for every modular evaluation done during the hyperparameter search. The effect of using `QuestionEncoder` as a prior is clear, setting a floor for the surrogate gradient module's accuracy, which adds 5.5% on top of it. Accuracies for `Measure` and `Describe` have also increased with respect to values in Fig. 9. This model achieves a 56.66% accuracy on the test set, which is 1.79 points higher than the end-to-end baseline (see Table 2).

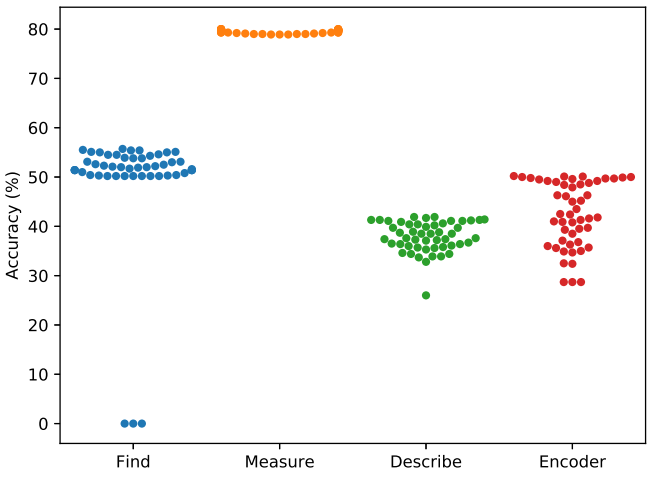

Fig. 12. Representation of the modular hyperparameter search after changes for improved compositionality.

Table 2
Comparison of accuracies achieved by different versions of the same NMN architecture.

Model version	Accuracy
end-to-end baseline	54.87%
modular	54.49%
modular + data types	**56.66**%

Table 3
Gradient-oriented implementation of several interface-related elements and their corresponding proposed neural data type for the experiment.

Element	Gradient-oriented implementation	Neural data type
Attention map	heat map (ReLU)	mask (sigmoid)
Answer	logits	softmax values
AND	multiplication	minimum
OR	addition	maximum

5.6. Compositionality

With the aim of evaluating how neural data types help neural modules generalize and exhibit compositional behaviour, we conducted a series of experiments on the CLEVR data set, in which we trained the neural modules on functional programs with a maximum of five consecutive operations, in an end-to-end fashion, and then tested them on deeper programs. Being CLEVR a synthetic data set, it provides functional programs up to 22 operations in depth.

For conducting these experiments, we translated CLEVR functional programs to network layouts. We then let one implementation have interfaces with neural data types, while we leave the other use interfaces optimized for gradient flow and end-to-end training. Thus, both implementations share architecture and differ solely on the interfaces, as described in Table 3.

The first experiment focuses on layouts from the training set with depths greater than those seen during training (Fig. 13 left). As these layouts are part of the same set, input images and features are from the same distribution, thus isolating the effect of the composition. We can see how the implementation with neural data types systematically beats the one without them. The error increases drastically from depth 6 to depth 7 and there is a tendency to increase the error for deeper layouts, which is almost negligible if neural data types are implemented. As a reference, the dotted lines show the model's error after having been trained on all program depths, which further evidences the benefit of neural data types.

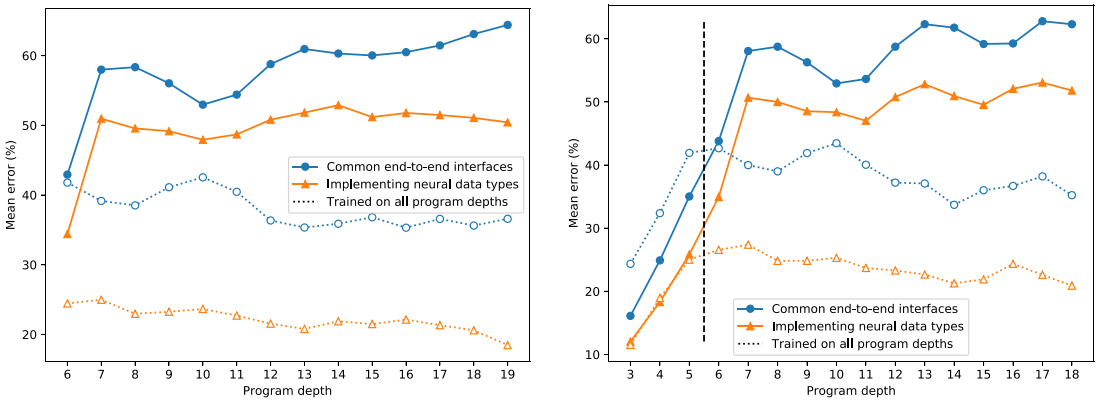

Fig. 13. Compositionality experiments on train (left) and validation data (right). We only show layout depths unseen during training, for which a significant number of samples was available. In validation, a dashed line separates depths seen and not seen during training. For reference, we also show the error after training on all program depths.

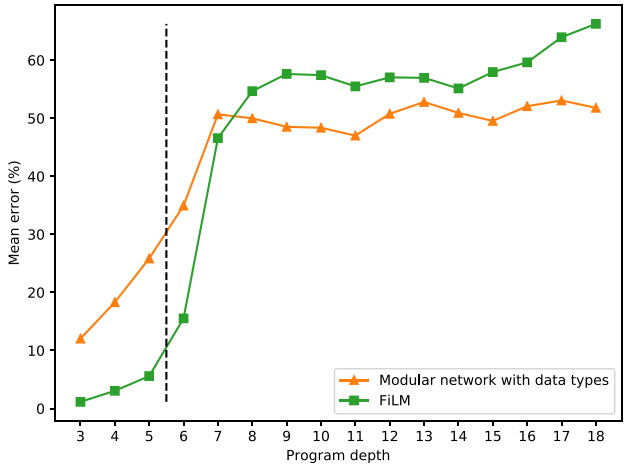

Fig. 14. Comparison of the generalization error at different program depths between a modular neural network with neural data types and a state-of-the-art monolithic neural network. Program depths seen during training are shown before the dashed line.

The second experiment is run on validation data (Fig. 13 right), including also depths seen during training. Here we also see the systemic prevalence of the implementation with neural data types, even on layout depths seen during training. However, the error raises quicker than on training data. This leaves image features as the main source of generalization error. We must point out that VGG-16 features do not comply with our definition of neural data types.

One final experiment focuses on giving context to these results, comparing the previously shown modular network against a monolithic architecture that has achieved state-of-the-art results in the CLEVR task (FiLM, Perez et al., 2018). We train the FiLM network on instances corresponding to program depths not greater than five, and test it afterwards on different program depths from the validation set (see Fig. 14). Despite exhibiting much better performance on trained depths than the otherwise low-capacity modular network, FiLM suffers a performance drop which is significantly worse.

6. Discussion and future work

In this work we have proposed a series of principles and guidelines for conducting modular training and improving compositional behaviour of modular neural networks. We have presented a formal framework for identifying and solving dependencies between modules in the network, enabling independent training and improving compositional behaviour and generalization. As far as we know, this is the first time that such an approach has been proposed.

We have tested our proposal on a case study based on the first implementation of NMN for the VQA v1 task, proving that independent modular training is possible and achieving accuracies that improve up to 3.28 points over the end-to-end baseline with greater robustness. We have also given insights into how modularity helps model analysis and redesign, improving robustness and resource utilization. One of our most relevant contributions is the surrogate gradient module, which makes it possible to train a module via indirect supervision (see Section 3.1) and model the output data type via a *de facto* specification.

We have concluded that neural data types help generalization and compositionality, pointing at non-typed interfaces as an important source of compositionality issues and identifying some of the existing neural network architectures that have leveraged them with broad success.

While working with surrogate gradient modules, we have detected some evidence that role implementations can present several degrees of fidelity, ranging from low-fidelity roles – which are a rough approximation of the target role and perform rather poorly at training the neural module – to high-fidelity roles — which perfectly reflect requirements and restrictions on the neural module's output and result in an optimal training. We have also speculated about the use of synthetic gradients for this task. Christiano et al. (2017) and Jaderberg et al. (2017) provide evidence indicating that relatively simple neural networks are able to learn a loss function or its gradient. It would be of great interest to see if this kind of techniques could be leveraged for having high quality modular losses or gradients and thus making it possible to retrain modules without having to specify a surrogate gradient module.

We lay these foundations with the intention of bootstrapping modularity by design in the field of neural networks. There remains the application of the modular approach to new instances of modular networks, reusing components and building neural libraries on the way. We find specially interesting the building of modular intelligent systems for heterogeneous tasks and data, where modular and dynamic neural networks may be an excellent option. Future work should also focus on mixing neural modules with other modular software pieces and the training of neural policies to optimally combine them.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

This research did not receive any specific grant from funding agencies in the public, commercial, or not-for-profit sectors.

The authors deeply thank the reviewers for their valuable comments and suggestions.

References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., ... Zheng, X. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org. https://www.tensorflow.org/.

Andreas, J., Rohrbach, M., Darrell, T., & Klein, D. (2016). Neural module networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 39–48).

Antol, S., Agrawal, A., Lu, J., Mitchell, M., Batra, D., & Zitnick, C. L. (2015). VQA: Visual question answering. In *International conference on computer vision*.

Bahdanau, D., Murty, S., Noukhovitch, M., Nguyen, T. H., de Vries, H., & Courville, A. (2019). Systematic generalization: What is required and can it be learned? In *International conference on learning representations*. URL https://openreview.net/forum?id=HkezXnA9YX.

Brock, A., Donahue, J., & Simonyan, K. (2019). Large scale GAN training for high fidelity natural image synthesis. In *International conference on learning representations*. URL https://openreview.net/forum?id=B1xsqj09Fm.

Cai, J., Shin, R., & Song, D. (2017). Making neural programming architectures generalize via recursion. In *International conference on learning representations*. URL https://openreview.net/forum?id=BkbY4psgg.

Castillo-Bolado, D., Guerra-Artal, C., & Hernández-Tejera, M. (2019). Modularity as a means for complexity management in neural networks learning. In *Proceedings of the AAAI 2019 spring symposium on combining machine learning with knowledge engineering*. URL http://ceur-ws.org/Vol-2350/paper8.pdf.

Chen, K. (2015). Deep and modular neural networks. In *Springer handbook of computational intelligence*. Springer, Ch. 28.

Christiano, P. F., Leike, J., Brown, T., Martic, M., Legg, S., & Amodei, D. (2017). Deep reinforcement learning from human preferences. In *Advances in neural information processing systems* (pp. 4299–4307).

Danihelka, I., Wayne, G., Uria, B., Kalchbrenner, N., & Graves, A. (2016). Associative long short-term memory. In M. F. Balcan, & K. Q. Weinberger (Eds.), *Proceedings of machine learning research: Vol. 48, Proceedings of the 33rd international conference on machine learning* (pp. 1986–1994). New York, New York, USA: PMLR, URL http://proceedings.mlr.press/v48/danihelka16.html.

Deng, J., Dong, W., Socher, R., Li, L. -J., Li, K., & Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition* (pp. 248–255). IEEE.

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

Gal, Y., & Ghahramani, Z. (2016). Dropout as a Bayesian approximation: Representing model uncertainty in deep learning. In *Proceedings of the 33rd international conference on international conference on machine learning - Volume 48* ICML'16, (pp. 1050–1059). JMLR.org.

Gleave, A., Dennis, M., Wild, C., Kant, N., Levine, S., & Russell, S. (2020). Adversarial policies: Attacking deep reinforcement learning. In *International conference on learning representations*. URL https://openreview.net/forum?id=HJgEMpVFwB.

Goller, C., & Kuchler, A. (1996). Learning task-dependent distributed representations by backpropagation through structure. In *Neural Networks, 1996., IEEE international conference on. Vol.1* (pp. 347–352). IEEE, http://dx.doi.org/10.1109/icnn.1996.548916.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., & Ozair, S. (2014). Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, & K. Q. Weinberger (Eds.), *Advances in neural information processing systems 27* (pp. 2672–2680). Curran Associates, Inc., URL http://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf.

Graves, A., Wayne, G., & Danihelka, I. (2014). Neural turing machines. arXiv preprint arXiv:1410.5401.

Graves, A., Wayne, G., Reynolds, M., Harley, T., Danihelka, I., & Grabska-Barwińska, A. (2016). Hybrid computing using a neural network with dynamic external memory. *Nature*, *538*(7626), 471–476. http://dx.doi.org/10.1038/nature20101.

Grefenstette, E., Hermann, K. M., Suleyman, M., & Blunsom, P. (2015). Learning to transduce with unbounded memory. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, & R. Garnett (Eds.), *Advances in neural information processing systems 28* (pp. 1828–1836). Curran Associates, Inc., URL http://papers.nips.cc/paper/5648-learning-to-transduce-with-unbounded-memory.pdf.

Gupta, N., Lin, K., Roth, D., Singh, S., & Gardner, M. (2020). Neural module networks for reasoning over text. In *International conference on learning representations*. URL https://openreview.net/forum?id=SygWvAVFPr.

Hanin, B., & Sellke, M. (2017). Approximating continuous functions by relu nets of minimal width. arXiv preprint arXiv:1710.11278.

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, *9*(8), 1735–1780.

Hu, R., Andreas, J., Rohrbach, M., Darrell, T., & Saenko, K. (2017). Learning to reason: End-To-End module networks for visual question answering. In *the IEEE international conference on computer vision*.

Hupkes, D., Dankers, V., Mul, M., & Bruni, E. (2020). Compositionality decomposed: How do neural networks generalise? *Journal of Artificial Intelligence Research*, http://dx.doi.org/10.1613/jair.1.11674.

Jacobs, R. (1990). *Task decomposition through competition in a modular connectionist architecture* (Ph.D. thesis), Amherst, MA, USA: University of Massachusets.

Jacobs, R. A., Jordan, M. I., Nowlan, S. J., & Hinton, G. E. (1991). Adaptive mixtures of local-experts. *Neural Computation*, *3*(1), 79–87.

Jaderberg, M., Czarnecki, W. M., Osindero, S., Vinyals, O., Graves, A., & Silver, D. (2017). Decoupled neural interfaces using synthetic gradients. In *Proceedings of the 34th international conference on machine learning-Volume 70* (pp. 1627–1635). JMLR.org.

Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., & Girshick, R. (2014). Caffe: Convolutional architecture for fast feature embedding. arXiv preprint arXiv:1408.5093.

Jin, Y. (2011). Surrogate-assisted evolutionary computation: Recent advances and future challenges. *Swarm and Evolutionary Computation*, *1*(2), 61–70. http://dx.doi.org/10.1016/j.swevo.2011.05.001.

Johnson, J., Hariharan, B., van der Maaten, L., Fei-Fei, L., Zitnick, C. L., & Girshick, R. (2017). CLEVR: A Diagnostic dataset for compositional language and elementary visual reasoning. In *CVPR*.

Joulin, A., & Mikolov, T. (2015). Inferring algorithmic patterns with stack-augmented recurrent nets. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, & R. Garnett (Eds.), *Advances in neural information processing systems 28* (pp. 190–198). Curran Associates, Inc., URL http://papers.nips.cc/paper/5857-inferring-algorithmic-patterns-with-stack-augmented-recurrent-nets.pdf.

Kirsch, L., Kunze, J., & Barber, D. (2018). Modular networks: Learning to decompose neural computation. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, & R. Garnett (Eds.), *Advances in neural information processing systems 31* (pp. 2408–2418). Curran Associates, Inc., URL http://papers.nips.cc/paper/7508-modular-networks-learning-to-decompose-neural-computation.pdf.

Klein, D., & Manning, C. D. (2003). Accurate unlexicalized parsing. In *Proceedings of the 41st annual meeting on association for computational linguistics - Volume 1* ACL '03, (pp. 423–430). USA: Association for Computational Linguistics, http://dx.doi.org/10.3115/1075096.1075150.

Kurach, K., Andrychowicz, M., & Sutskever, I. (2015). Neural random-access machines. arXiv preprint arXiv:1511.06392.

Ladousse, G., & Maley, A. (1987). *Oxford English, Role play*. OUP Oxford.

Lu, Z., Pu, H., Wang, F., Hu, Z., & Wang, L. (2017). The expressive power of neural networks: A view from the width. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, & R. Garnett (Eds.), *Advances in neural information processing systems 30* (pp. 6231–6239). Curran Associates, Inc., URL http://papers.nips.cc/paper/7203-the-expressive-power-of-neural-networks-a-view-from-the-width.pdf.

Montague, R. (1970). Universal grammar. *Theoria*, *36*(3), 373–398.

Novak, R., Bahri, Y., Abolafia, D. A., Pennington, J., & Sohl-Dickstein, J. (2018). Sensitivity and generalization in neural networks: An empirical study. In *International conference on learning representations*. URL https://openreview.net/forum?id=HJC2SzZCW.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E. and DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., ... Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché Buc, F., Fox, E., & Garnett, R. (Eds.), *Advances in neural information processing systems 32* (pp. 8024–8035). Curran Associates, Inc., URL http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

Perez, E., Strub, F., de Vries, H., Dumoulin, V., & Courville, A. C. (2018). FiLM: Visual reasoning with a general conditioning layer. In *AAAI*.

Queipo, N. V., Haftka, R. T., Shyy, W., Goel, T., Vaidyanathan, R., & Tucker], P. K. (2005). Surrogate-based analysis and optimization. *Progress in Aerospace Sciences*, *41*(1), 1–28. http://dx.doi.org/10.1016/j.paerosci.2005.02.001.

Reed, S., & De Freitas, N. (2015). Neural programmer-interpreters. arXiv preprint arXiv:1511.06279.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., & van den Driessche, G. (2016). Mastering the game of go with deep neural networks and tree search. *Nature*, *529*, 484–489.

Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. arXiv e-prints, arXiv:1409.1556.

Singh, A., Goswami, V., Natarajan, V., Jiang, Y., Chen, X., & Shah, M. (2018). Pythia-a platform for vision & language research. In *Sysml workshop, NeurIPS: Vol. 2018*.

Socher, R., Perelygin, A., Wu, J., Chuang, J., Manning, C. D., & Ng, A. Y. (2013). Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing* (pp. 1631–1642).

Strubell, E., Ganesh, A., & McCallum, A. (2019). Energy and policy considerations for deep learning in NLP. In *Proceedings of the 57th annual meeting of the association for computational linguistics* (pp. 3645–3650). Florence, Italy: Association for Computational Linguistics, http://dx.doi.org/10.18653/v1/P19-1355.

Véniat, T., Schwander, O., & Denoyer, L. (2019). Stochastic adaptive neural architecture search for keyword spotting. In *ICASSP 2019-2019 IEEE international conference on acoustics, speech and signal processing* (pp. 2842–2846). IEEE.

Yosinski, J., Clune, J., Nguyen, A., Fuchs, T., & Lipson, H. (2015). Understanding neural networks through deep visualization. arXiv preprint arXiv:1506.06579.

Yu, Z., Yu, J., Cui, Y., Tao, D., & Tian, Q. (2019). Deep modular co-attention networks for visual question answering. In *the IEEE conference on computer vision and pattern recognition*.