Neural Function Modules with Sparse Arguments: A Dynamic Approach to Integrating Information across Layers

Alex Lamb
Universite de Montreal

Anirudh Goyal
Universite de Montreal

Agnieszka Słowik
University of Cambridge

Michael Mozer
Google Research / University of Colorado

Philippe Beaudoin
Element AI

Yoshua Bengio
Mila

Abstract

Feed-forward neural networks consist of a sequence of layers, in which each layer performs some processing on the information from the previous layer. A downside to this approach is that each layer (or module, as multiple modules can operate in parallel) is tasked with processing the entire hidden state, rather than a particular part of the state which is most relevant for that module. Methods which only operate on a small number of input variables are an essential part of most programming languages, and they allow for improved modularity and code re-usability. Our proposed method, Neural Function Modules (NFM), aims to introduce the same structural capability into deep learning. Most of the work in the context of feed-forward networks combining top-down and bottom-up feedback is limited to classification problems. The key contribution of our work is to combine attention, sparsity, top-down and bottom-up feedback, in a flexible algorithm which, as we show, improves the results in standard classification, out-of-domain generalization, generative modeling, and learning representations in the context of reinforcement learning.

Proceedings of the 24[th] International Conference on Artificial Intelligence and Statistics (AISTATS) 2021, San Diego, California, USA. PMLR: Volume 130.

1 Introduction

Much of the progress in deep learning architectures has come from improving the ability of layers to have a specific focus and specialization. One of the central drivers of practical progress in deep learning has been finding ways to make networks deeper and to create inductive bias towards specialization. Indeed, the general trend in state-of-the-art techniques has been from networks with just a few layers to networks with hundreds or thousands of layers, where each layer has a much more specific role. Iterative inference and generation [Marino et al., 2018, Jastrzębski et al., 2017, Greff et al., 2019] approaches also share this motivation: since they only require each pass to make a small change to an underlying representation. This has been applied in iterative inference for generative models [Wu et al., 2019].

Perhaps the best example of this is residual networks (ResNets) [He et al., 2016a], in which an additive skip-connection is placed between layers. This allows a layer to use any other layer via the linear additive skip connections, yet this is a rigid and inflexible communication between layers. Despite that limitation, residual networks have now become ubiquitous, and are now used in almost all networks where computational resources allow for a large number of layers.

Other ideas in feed-forward networks have explored how to make individual layers more narrowly focused. In the DenseNet [Huang et al., 2017], each layer takes as input all previous layers within the same block, concatenated together. This reduces the need to store redundant information in multiple layers, as a layer can directly use information from any prior layer given as input, even if it does not directly precede that layer. Neural ODEs [Chen et al., 2018a] explored a continuous variant of ResNets, in which it can be directly

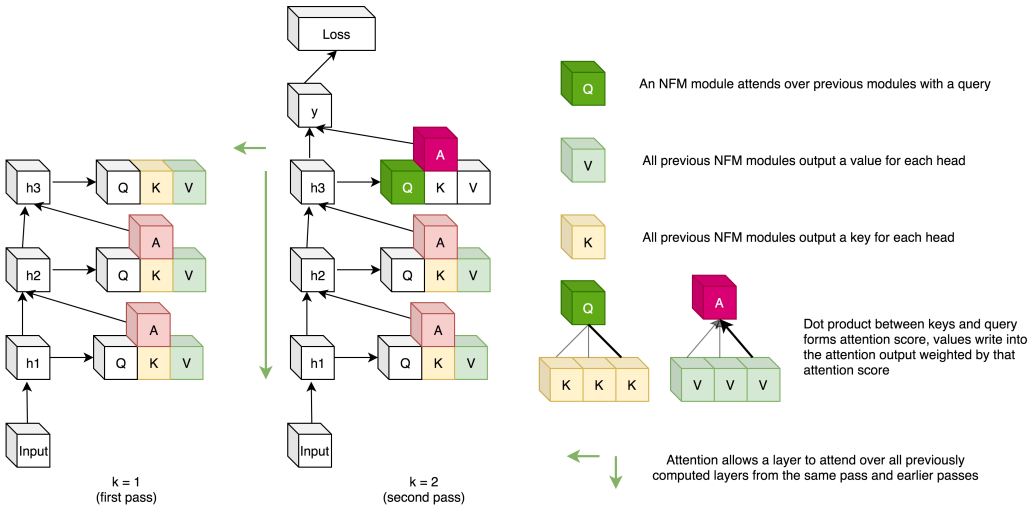


Figure 1: An illustration of NFM where a network is ran over the input twice ($\mathcal{K} = 2$). A layer where NFM is applied sparsely attends over the set of previously computed layers, allowing better specialization as well as top-down feedback.

shown that making each layer perform an even more narrowly focused computation (in this sense, meaning a smaller update) can improve accuracy.

While these techniques have greatly improved the generalization performance of deep networks, in large part by making individual layers more specialized, we argue that they are still limited in that they take the entire current program state as input, rather than dynamically selecting input arguments. To address this limitation, we take inspiration from computer programs, which are much richer and more flexible than today's deep networks. Computer programs are typically organized into methods, which perform a specific task, and only operate over a specific set of inputs which are relevant for that task. This has several advantages over writing code without these well-contained methods. One reason is that it leads to better separation of concerns and modularity. A method which is written once can be reused in different programs, even if the overall purpose of the program is very different. Another benefit to using methods is that it allows the programmer to avoid accidentally using or changing variables unrelated to the function. In this way a program could use a larger number of variables and have a greater amount of complexity, with a contained risk of introducing bugs. Another advantage is that mistakes can be isolated down to a specific method within the program, which can lead to a more concentrated and efficient credit assignment.

Additionally, methods in computer programs have the property that they are able to take arguments from the entire program state which is in scope. Thus they may use either recently computed variables, or variables computed much earlier in the program. This has some connection to bottom-up and top-down feedback signals from cognitive psychology, with recently computed variables or input streams being analogous to bottom-up signals and higher-level goals or descriptive variables being analogous to top-down signals. This idea has seen some work in the deep learning community [Zamir et al., 2017], although many deep architectures (such as feed-forward networks) rely exclusively on bottom-up processing.

Our proposal is to allow modules (corresponding to computational blocks which can be executed in parallel or in sequence) in a deep network to attend over the previously computed modules from the network to construct their input. This proposal, which we call *Neural Function Modules (NFM)*, is motivated by analogy with functions in programming languages - which typically only take a few variables as arguments, rather than acting upon the entire global state of the program. Likewise an NFM module selects the hidden states from the previously computed modules as its inputs, and may use its entire hidden state to focus on these few inputs. The modules which come later in the network may attend to either that module or the modules preceding it.

Additionally, to allow the NFM to consider both top-down and bottom-up signal specialization, we use a multi-pass setup, in which we run the networks forward pass multiple times, and allow the NFM to attend over all of the previously seen passes. In a classifier, the later parts of the network from earlier passes correspond to "top-down" feedback, since they come from a network which has seen and has a compressed representation

of the entire example. Whereas attending to parts of the network closer to the input correspond to "bottom-up" feedback. In general we consider a two-pass setup, but also perform analysis on single-pass and three-pass setups.

We believe that the largest impact from adding this structure to deep learning will be in tasks where the question of when a layer is used is dynamic, which can occur as a result of the data distribution shifting during training, or the data distribution systematically differing between training and evaluation. For example, the former occurs organically while training a GAN generator, since the objective from the discriminator is constantly evolving as the adversarial game progresses.

Our newly proposed NFM module captures concepts of iterative inference, sparse dynamic arguments, and flexible computation, yet is straightforward to integrate into a variety of existing architectures. We demonstrate this by integrating NFM into convolutional classifiers, GAN generators, and VAE encoders. In each case, the structure of the NFM module is kept the same, but the details of the integration differ slightly.

Our method offers the following contributions:

- Allowing layers to focus their entire output space size without limiting the information accessible to later modules.
- Making modules more specialized, by allowing them to focus on particular relevant inputs, rather than the entire hidden state of the network.
- Providing a mechanism to allow for dynamic combination of top-down and bottom-up feedback signals.

The key contribution of our work is to combine attention, sparsity, top-down and bottom-up feedback, in a flexible algorithm which can improve the results in the standard classification, out-of-domain generalization, generative modeling and reinforcement learning (as demonstrated in Section 4). Most of the work in the context of feed-forward networks combining top-down and bottom-up feedback is limited to classification problems. To the best of our knowledge, no work has combined these ingredients together in a unified architecture, that can be used to show improvements in various different problems, i.e classification, generative modelling, out of distribution generalization and learning representations in the context of RL.

Our experiments show that these NFM modules can dynamically select relevant modules as inputs, leading to improved specialization of modules. As a result, we show that this leads to improved generalization to changing task distributions. We also demonstrate its flexibility, by showing that it improves performance in classification, relational reasoning, and GANs.

2 Related Work

DenseNet: These use concatenation to combine all of the layers within a block as inputs. The largest difference between NFM and DenseNet is that NFM uses sparse attention, whereas DenseNet uses concatenation. Another key difference is that NFM uses attention over many previously seen modules, even if they are of different sizes, whereas DenseNet only uses the handful of previously computed layers of the same spatial dimension. Most of the papers that have used DenseNets, have tested it on mostly computer vision problems (like classification), whereas NFM is a generic module that can be used for improving systematic generalization in relational reasoning, for classification as well as for generative modelling. An attentive variant of DenseNets was also proposed [Kim et al., 2018].

Conditional Computation and Modularity: Most of the current deep learning systerms are built in the form of one big network, consisting of a layered but otherwise monolithic structure, which can lead to poor adaptation and generalization [Andreas et al., 2016, Santoro et al., 2017a, Bahdanau et al., 2018, Bengio et al., 2019, Goyal et al., 2019]. In order to address this problem, there have been attempts in modularizing the network such that a neural network is composed dynamically from several neural modules, where each module is meant to perform a distinct function [Andreas et al., 2016, Shazeer et al., 2017, Rosenbaum et al., 2017, Goyal et al., 2019]. The motivation behind methods that use conditional computation is to dynamically activate only a portion of the entire network for each example [Bengio, 2013, Wu et al., 2018, Fernando et al., 2017, McGill and Perona, 2017]. Recently [Hu et al., 2018, Woo et al., 2018, Wang et al., 2018, Chen et al., 2018b, Veit and Belongie, 2018] have proposed to use a learned gating mechanism (either using reinforcement learning or evolutionary methods) to dynamically determine when to skip in ResNet in a context dependent manner to reduce computation cost. In this line of work, no multiple modules are explicitly defined. Instead, the whole network is dynamically configured by selectively activating model components such as hidden units and different layers for each input example. Most of these works have been applied to computer vision problems such as image classification or segmentation or object detection. Our work is more related to such methods that dynamically decide where to route information but instead of skipping a particular layer or a unit, NFM

Table 1: Desiderata and Related Work: showing how our motivations relate to prior work.

Desiderata	Related Methods
Sparse Arguments	SAB [Ke et al., 2018] Sparse Transformer [Child et al., 2019]
Iterative Inference	LOGAN [Wu et al., 2019],GibbsNet [Lamb et al., 2017]
Dynamically Skipping Layers	SkipNet [Wang et al., 2018]
Combining Top-Down and Bottom-Up Feedback	Deep Boltzmann Machines [Salakhutdinov and Hinton, 2009]

dynamically decide where to *query* information from (i.e., which layers to attend to), using sparse attention. Another key difference in the proposed method is that layers can attend to both the *bottom-up*, as well as *top-down* information which has been shown to improve systematic generalization as also evident in our experiments. Also, NFM is a generic module that can be used for other problems whereas most of these methods have been studied exclusively for image classification.

Transformers: The Transformer architecture uses attention over positions, but only on the previous layer. NFM attends over layers, making it complementary with Transformers, yet the methods share a related motivation of allowing parts of the network to dynamically select their inputs using attention. Future work can also investigate on integrating NFM module with transformers.

Sparse Attention in RNNs: Sparse Attentive Backtracking (SAB) [Ke et al., 2018] used sparse attention for assigning credit to a sparse subset of time steps in the context of RNNs leading to efficient credit assignment as well as efficient transfer. More recently, Recurrent Independent Mechanisms (RIMs) [Goyal et al., 2019] also use sparse attention for dynamically selecting a sparse subset of modules in an input dependent manner. Both SAB and RIMs uses sparse-attention in the context of RNNs, where the proposed method uses attention to dynamically integrate bottom-up as well as top-down information.

3 Neural Function Modules

Our goal is to introduce the idea of Neural Function Modules (NFM) as a new way of composing layers in deep learning. To that end, we start with desiderata motivating our design without going into architectural details. Next, we give a detailed algorithmic description using a specific set of tools (such as top-k softmax [Ke et al., 2018] as a way of implementing sparse attention), while we note that our concept is more general than this specific architecture. We then describe how NFM can be integrated into various architectures.

3.1 Desiderata

We lay out desiderata motivating our design, in the hope of defining a class of architectures that share these goals:

- Creating deep networks in which the communication between layers is dynamic and state-dependent, allowing it to be invariant to prior layers which are not relevant for it;
- To allow a deep neural network to selectively route information flow around some layers, to break the bottleneck of sequential processing;
- To allow a model to dynamically combine bottom-up and top-down information processing using attention; and
- To introduce more flexibility into deep architectures, by breaking the constraint that each layer needs to be a suitable input for the following layer. For example, a layer which destroys fine-grained spatial information may be difficult to use earlier in the network, but could be valuable for later processing in deeper layers.

3.2 Proposed Implementation

We describe our particular implementation of the Neural Functional Module concept which uses sparse attention, although we view the basic idea as being more general, and potentially implementable with different tools. In particular, we introduce a new NFM module which is encapsulated and has its own parameters. It stores every previously seen layer (from the forward pass) in its internal state, and uses the current layer's state as a query for attending over those previously seen layers. This is described in detail at in Algorithm 1.

3.2.1 Mutli-Head Attention Mechanism

We aimed to use the multi-head attention mechanisms from Transformers with as few modifications as possible. We use linear operations (separate per module) to compute the key, value, and query for each module. We then use softmax top-k attention [Ke et al., 2018,

Algorithm 1 Neural Functional Module (NFM)

Input: An input x. A number of passes $\mathcal{K}$. A neural network with N modules for all $k \in \{1...\mathcal{K}\}$: $f^{(1)}_{\theta_k}$, $f^{(2)}_{\theta_k}$, $f^{(3)}_{\theta_k}$, ..., $f^{(N)}_{\theta_k}$
$\mathcal{M} := EmptyList$
for $k = 1$ to $\mathcal{K}$ **do**
  $h^{(0)} := x$
  for $i = 1$ to N **do**
    $\tilde{\mathcal{M}} := EmptyList$
    for $j = 1$ to i **do**
      $\tilde{\mathcal{M}}.append(rescale(\mathcal{M}_j), scale(h^{(i-1)}))$
    end for
    $A = Attention^{(i)}_{\tilde{\theta}_k}(K = \tilde{\mathcal{M}}, V = \tilde{\mathcal{M}}, Q = h^{(i-1)})$
    $h^{(i)} := f^{(i)}_{\theta_k}(residual = h^{(i-1)}, input = A)$
    $\mathcal{M}$.append$(h^{(i)})$
  end for
end for

Goyal et al., 2019], in which the attention only attends over the elements with the top-k highest attention scores. Then we use one linear layer, followed by an activation, followed by a linear layer to project back to the original size. Depending on the problem setting, we may or may not use batch normalization.

We now lay out the structure of the multi-headed attention mechanism which is used in Algorithm 1. The $Attention^{(i)}_{\tilde{\theta}_K}$ function takes a set of keys and values to be attended-over: K, V, a query q, and a residual connection R. The keys and queries are given dimension d_k and the values are given dimension d_v. It is specific to the layer index i and the parameters for attention for the layer i: $(W_q, W_k, W_v, W_{o_1}, W_{o_2}, \gamma) = \tilde{\theta}^{(A)}_i$. These W refer to weight matrices and γ is a learned scalar which is initialized to zero at the start of training. Additionally a k to specify the k for top-k attention (typically k is set to a value less than ten). An activation function σ must also be selected (in our case, we used ReLu). We place a batch normalization module directly before applying the activation σ.

We set $\hat{Q} = qW_q$, $\hat{K} = KW_k$, $\hat{V} = VW_v$. Then we compute $A = Softmax(\frac{\hat{Q}\hat{K}^T}{d_k})V$. Afterwards the output hidden state is computed as $h_1 = \sigma(AW_{o_1})W_{o_2}$. Then the final output from the attention blocked is multiplied by a γ scalar and added to the residual skip-connection: $h_2 = R + \gamma h_1$. The weighting with a γ scalar was used in self-attention GANs [Zhang et al., 2018].

Default Computation Additionally, we append a zero-vector to the beginning of the list of keys and values for the layers to be attended over. Thus, a querying position on some (or all) of its heads may elect to look at these zeros rather than reading from an input element. This is analogous to performing a function with a number of arguments which is less than the number of heads.

Table 2: Classification Results (% Test Accuracy) with NFM show consistent improvements across tasks and architectures. All results use Input Mixup with $\alpha = 1.0$ and the experimental procedure in [Verma et al., 2018]

Methods	Base Arch.	Baseline	NFM
CIFAR-10	PreResNet18	96.27 ± 0.2	96.56 ± 0.09
CIFAR-10	PreResNet34	96.79 ± 0.1	97.05 ± 0.1
CIFAR-10	PreResNet50	97.31 ± 0.1	97.58 ± 0.2
CIFAR-100	PreResNet18	78.15 ± 0.1	78.66 ± 0.3
CIFAR-100	PreResNet34	80.13 ± 0.3	80.77 ± 0.1
Tiny-Imagenet	PreResNet18	57.12 ± 0.3	58.32 ± 0.2
Imagenet	PreResNet18	76.72	77.1

3.2.2 Rescaling Layers (Automatic Type Conversion)

When we query from a module of spatial dimension $len(h_i)$, we consider attending over modules which may have either the same or different spatial dimensions $len(h_j)$. For simplicity, we discuss 1D layers, but the rescaling approach naturally generalizes to states with 2D or 3D structure.

If $len(h_i) = len(h_j)$, then the two modules are of the same scale, and no rescaling is performed. If $len(h_i) > len(h_j)$, then h_j needs to be upsampled to the size of h_i, which we do by using nearest-neighbor upsampling. The interesting case is when $len(h_i) < len(h_j)$, which is when downsampling is required. The simplest solution would be to use a nearest-neighbor downsampling, but this would involve arbitrarily picking points from h_j in the downsampling, which could discard interesting information. Instead, we found it performed slightly better to use a SpaceToDepth operation to treat all of the points in the local window of size $\frac{len(h_j)}{len(h_i)}$ as separate positions for the attention. Thus when attending over a higher resolution module, NFM can pick specific positions to attend over.

3.3 Integrating NFM

We have presented an encapsulated NFM module which is flexible and could potentially be integrated into many different types of networks. Notably, in all of these integrations, the internal structure of the NFM module itself remains the same, demonstrating the simplicity and flexibility of the approach. Additionally, using NFM introduces no additional loss terms.

4 Experiments

Our experiments have the following goals:

Table 3: Results on Atari games with NFM show improved scores with NFM used in the game-image encoder.

Game	Architecture	Baseline	NFM
Ms. Pacman	ResNet18	6432	6300 ± 3384
Alien	ResNet18	12500	14382 ± 239
Amidar	ResNet18	3923	3948 ± 23
Q-bert	ResNet18	27433	33253 ± 2302
Crazy Climber	ResNet18	333242	334323 ± 2389

- Demonstrate that Neural Function Modules can improve results on a wide array of challenging benchmark tasks, with the goal of demonstrating the practical utility and breadth of the technique.
- To show that NFM addresses the bottleneck problem in the size of the hidden layers, by achieving drastically improved performance when the model is made narrow, with very few hidden units per module.
- To show that NFM improves generalization when the train and test set differ systematically, as a result of improved specialization of the modules over sparse functional sub-tasks.

4.1 GANs

Intuitively, generating images with structured objects involves various sparse functional operations - for example creating a part such that it is consistent with another part, or generating a part with particular properties. Based on this intuition we integrated NFM into InfoMax GAN [Kwot Sin Lee and Cheung, 2019], and we modify only the generator of the GAN by integrating NFM (the discriminator and all of the losses are kept the same). In our integration, we use two passes $\mathcal{K} = 2$ and we place an NFM module before each residual block. Thus a total of 8 NFM modules are integrated. We used $d_k = 32$, $d_v = 32$, and 4 heads for the attention. For both CIFAR-10 and Imagenet our base generator architecture is a ResNet18.

We integrated NFM into an Infomax-GAN for both CIFAR-10 and Tiny-Imagenet. We elected to integrate NFM into the generator only, since it is computationally much cheaper than using it in both the generator and the discriminator, as multiple discriminator updates are done for each generator update. The original Infomax-GAN 32x32 generator consists of a linear layer from the original latents to a 4x4 spatial layer with 256 channels. This is then followed by three residual blocks and then a final convolutional layer. Our integration applies NFM after this first spatial layer and after the output of each residual block. Since we use two passes $\mathcal{K} = 2$, the NFM module is applied a total of 8 times (4 in each pass). The only change we introduced was the integration of NFM and the two-pass generator. Aside from that, the hyperparameters for training the GAN are unchanged from [Lee and Town, 2020].

We used the [Lee and Town, 2020] GAN code base with default hyperparameters for all of our GAN experiments. We only changed the architecture of generator to incorporate NFM. Thus it might be the case that we could have achieved even better performance if we re-tuned the hyperparameters specifically for our the NFM architecture, yet in practice we found solid improvements even without doing this. A significant improvement on GANs using NFM are shown in Table 4 as measured by Frechet Inception Distance (FID) [Heusel et al., 2017] and Inception Score (IS) [Salimans et al.,]. We integrate NFM directly into the Pytorch Mimicry codebase which contains a variety of techniques: SNGAN, SSGAN, InfoMax-GAN, and WGAN-GP. The NFM model outperforms all of these strong baselines (Table 4).

4.2 Stacked MNIST Generalization

We consider a simple multi-object classification task in which we change the number of object between training and evaluation, and verify how well the model is able to generalize. Our reasoning is that if the model learns sparse functional modules for recognizing the objects, then it should be able to handle novel numbers of objects. To keep this task as simple as possible, we construct synthetic 64x64 images containing multiple MNIST digits, and we train a convnet with the output as a multi-label binary classifier for each digit.

For the integration with NFM, we used two passes ($\mathcal{K} = 2$) and use a top-k sparsity of $k = 5$. Thus the NFM module is used 8 times with $d_k = 16$, $d_v = 16$, and 4 heads. For more details and experimental setup, see Appendix B).

4.3 Relational Reasoning

In relational reasoning a model is tasked with recognizing properties of objects and answering questions about their relations: for example, "is the red ball in front of the blue ball". This problem has a clear sparse functional structure which requires first recognizing the properties of objects from images and then analyzing specific relations, and thus we sought to investigate if NFM could improve results in this domain. We used the Sort-of-CLEVR [Santoro et al., 2017b] task to evaluate NFM in the context of visual reasoning and compositional generalization. We analyzed the effect of extending a generic convolutional baseline (CNN_MLP) with NFM (specifically, NFM-ConvNet; Table 6). The baseline implementations and dataset

Table 4: Improved generation with GANs (no use of class labels) on CIFAR-10 and Tiny-Imagenet, outperforming many strong baselines on Inception Score (IS) and Frechet Inception Distance (FID). We compare our NFM(InfoMax-GAN) against three external baselines: SNGAN [Miyato et al., 2018], SSGAN [Chen et al., 2019], and InfoMax-GAN [Kwot Sin Lee and Cheung, 2019].

Methods	CIFAR-10 FID	CIFAR IS	Tiny-Imagenet FID	Tiny-Imagenet IS
SNGAN	16.77 ± 0.04	7.97 ± 0.06	23.04 ± 0.06	8.97 ± 0.12
SSGAN	14.65 ± 0.04	8.17 ± 0.06	21.79 ± 0.09	9.11 ± 0.12
InfoMax-GAN	15.12 ± 0.10	8.08 ± 0.08	20.68 ± 0.02	9.04 ± 0.10
NFM(InfoMax-GAN)	13.15 ± 0.06	8.34 ± 0.02	18.23 ± 0.08	9.12 ± 0.09

Table 5: Recognizing images with multiple mnist digits: training on one or three digits (top), one or five digits (bottom), provides evidence of improved specialization over the digit recognition sub-task (test accuracy %).

Methods	Baseline	NFM
Trained (1,3) digits		
One Digit	99.27 ± 0.05	99.23 ± 0.03
Three Digits	87.83 ± 0.02	87.78 ± 0.46
Two Digits	76.7 ± 1.34	88.36 ± 0.85
Four Digits	62.57 ± 0.04	66.12 ± 1.86
Five Digits	24.27 ± 1.02	29.48 ± 3.12
Trained (1,5) digits		
One Digit	99.22 ± 0.09	99.16 ± 0.04
Five Digits	68.87 ± 4.25	71.76 ± 2.79
Two Digits	74.69 ± 2.88	84.01 ± 6.06
Three Digits	57.11 ± 3.57	66.58 ± 5.24
Four Digits	75.90 ± 2.48	79.04 ± 2.37

Table 6: Test accuracy on Relational reasoning (Sort-of-CLEVR) from images.

Task	CNN	NFM(CNN)
Relational qst	68.26 ± 0.61	74.95 ± 1.58
Non-relational qst	71.78 ± 9.3	79.37 ± 2

generation follow description in the paper which introduced Sort-of-CLEVR [Santoro et al., 2017b]. Similarly as in Section 4.2, we use two passes ($\mathcal{K} = 2$) and a top-k sparsity of $k = 5$. We report mean test accuracy and standard deviation (over three trials) in Table 6 and Table 7.

Images in Sort-of-CLEVR consist of 6 randomly placed geometrical shapes of 6 possible colors and 2 possible shapes. There are 10 relational and 10 non-relational questions per image. Non-relational questions have two possible answers (random guess accuracy is 50%). This also applies to relational questions of type 1 and type 2 (reasoning over distances between the objects). Relational questions of type 3 (count) have 6 possible answers. Therefore on average a random baseline for relational questions has accuracy of $\approx 39\%$. While Sort-of-CLEVR is a simple dataset in its original form, it can be made substantially more difficult by introducing a distribution shift (Table 7). Table 7 shows the results of omitting N color-shape combinations from the training set and testing on the original $N = 12$ combinations. We include the results of a strong baseline, Relation Networks (RNs) [Santoro et al., 2017b]. Note that RNs contain a module specifically tailored for answering relational questions in this task.

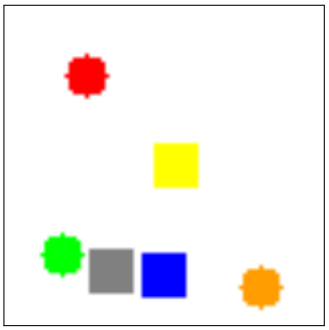
The accuracy of both baselines decreases towards random performance with a distribution shift between training and test data. Our model outperforms the simple baseline and RNs in the non-relational set of questions, suggesting that NFM improves the stage of recognizing object properties even in the presence of a distribution shift. While generalization to out-of-distribution samples remains challenging when combined with relational reasoning, NFM might alleviate the need for additional fully connected layers such as those used in RNs. For more details regarding the setup we ask the reader to refer to Appendix C.

4.4 Classification and Generalization to Occlusions

We evaluated NFM on the widely studied classification benchmarks CIFAR-10, CIFAR-100, Tiny-Imagenet, and Imagenet and found improvements for all of them, with the goal of demonstrating the utility and versatility of NFM (Table 2). We used a two-pass setup with $k = 5$, and we trained both the NFM and baseline models with Mixup [Zhang et al., 2017]. We integrated with base architectures of PreActResNet18, PreActResNet34, and PreActResNet50 [He et al., 2016b].

As an ablation study on the classifier, we tried training with NFM normally, but at test time changed the attention values to be random (drawn from a Gaussian). We found that this dramatically hurt results on CIFAR-10 classification (96.5% to 88.5% test accuracy). This is evidence that the performance of NFM is dependent on the attention selectively picking modules as inputs.

In addition to improving classification results, we found that classifiers trained with NFM had better robustness



Relational questions:

1. What is the shape of the object closest to the red object? ⇒ square
2. What is the shape of the object furthest to the orange object? ⇒ circle
3. How many objects have same shape with the blue object? ⇒ 3

Non-relational questions:

1. What is the shape of the red object? ⇒ Circle
2. Is green object placed on the left side of the image? ⇒ yes
3. Is orange object placed on the upside of the image? ⇒ no

Figure 2: A sample from the Sort-of-CLEVR dataset.

Table 7: Compositional generalization (Sort-of-CLEVR) to unseen variations, suggesting better specialization over uncovering attributes and understanding relations.

Number of hold-out combinations	CNN	Relation networks	NFM(CNN)
$N = 1$: Relational qst	57.67 ± 0.47	50.67 ± 0.94	54.67 ± 1.7
$N = 1$: Non-relational qst	57 ± 1.63	48 ± 0.82	59 ± 0.81
$N = 2$: Relational qst	47 ± 3.27	45 ± 3.56	46 ± 1.41
$N = 2$: Non-relational qst	54.33 ± 0.47	43.33 ± 7.31	57 ± 1.63
$N = 3$: Relational qst	20.66 ± 3	40.67 ± 1.7	38.33 ± 3.3
$N = 3$: Non-relational qst	42.33 ± 1.25	46 ± 3.56	49.67 ± 3.3
Average	46.99	45.61	50.78

to occlusions (not seen at all during training). Evidence from neuroscience shows that feedback from frontal (higher) brain areas to V4 (lower brain areas) is critical in interpreting occluded stimuli. [Fyall et al., 2017]. Additionally research has shown that neural nets with top-down feedback better handle occluded and cluttered displays. [Spoerer et al., 2017]. Using our normally trained NFM model on CIFAR-10 with PreActResNet18, we improve test accuracy with occlusion boxes of size 16x16, with a single occlusion box per image, from 82.46% (baseline) to 84.11% (NFM). Our occlusion used the same parameters as from the Cutout paper [DeVries and Taylor, 2017]. For more details refer to Appendix A.

4.5 Atari

The Atari 2600 game environment involves learning to play simple 2D games which often contain small modules with clear and sparsely defined behavior patterns. For this reason we sought to investigate if using an encoder with NFM modules could lead to improved results. All compared methods used the same ResNet backbone, input preprocessing, and an action repeat of 4. Our NFM integration used two passes ($\mathcal{K} = 2$) and top-k sparsity of $k = 4$. We chose games that require some degree of planning and exploration as opposed to purely reactive ones: Ms. Pacman, Frostbite, Alien, Amidar, Hero, Q-bert, Crazy Climber. We choose this set of games, as it was previously used by [Vezhnevets et al., 2016]. We integrate NFM into the resnet encoder of a Rainbow IQN [Dabney et al., 2018, Hessel et al., 2018]. We use keysize d_k of 32, value size d_v of 32, 4 heads, two-passes $\mathcal{K} = 2$, and top-k sparsity of $k = 4$. Thus the NFM module is added at four places in the integration. After integrating NFM, we kept the same hyperparameters for training the IQN model. We use exactly the same setup as in [Lieber, 2019] for RL algorithm, as well as other hyper-parameters not specific to the propose architecture. We show substantially improved results on four of these five games (see Table 3).

4.6 Transformer-NFM for Language Modeling

4.7 Analysis of Hyperparameters

On CIFAR-100 classification (PreActResNet34) we jointly varied the keysize, valsize, and number of heads used for the NFM process (Table 9).

On the relational reasoning task, we tried using one-pass $\mathcal{K} = 1$, and we achieved test accuracy of 73.07 ± 1.17 on relational questions, which is better

Table 8: Results on Wikitext-2 with NFM integration in perplexity (lower is better) after 15 epochs of training. Note that in the transformer integration, NFM does not add any additional parameters and only a trivial amount of additional computation.

Model	Mechanisms	Perplexity
No-NFM	2	31.12
NFM	2	30.78
No-NFM	4	31.50
NFM	4	31.05

Table 9: Varying the key dimension, value dimension, top-k sparsity, and number of heads used for the attention process, when running PreActResNet34 on CIFAR-100 classification. Note that all results outperform the baseline accuracy of 80.13%.

Heads	Top-k	d_k	d_v	Test Accuracy (%)
2	3	8	16	80.37
2	3	8	32	80.47
2	3	8	64	80.41
2	4	8	16	80.26
2	4	8	32	80.77
2	4	8	64	80.52
4	3	16	16	80.31
4	3	16	32	80.26
4	3	16	64	80.27
4	3	32	32	80.40
4	4	16	32	80.55
4	4	32	64	80.33

than the baseline's 68.26 ± 0.61 accuracy, but worse than the NFM with two-passes $\mathcal{K} = 2$ (accuracy of 74.95 ± 1.58 on relational questions). We also saw an improvement thanks to introducing attention sparsity. In the reported results, both baseline CNN and NFM(CNN) use 24 initial channels. We also looked at the relation between the number of model parameters, test accuracy and adding/removing NFM in an image classification task (Appendix A).

We also experimented with higher capacity baselines for our GAN and relational reasoning experiments. Our CIFAR-10 baseline had an FID of 15.12. Doubling the number of channels at each layer improved the FID to 14.65 and doubling the number of layers improved the FID to 14.43. With NFM and the original number of channels/layers, we achieved a much greater improvement, reducing the FID to **13.15**. We also experimented with higher capacity baselines on the Sort-of-Clevr relational reasoning task. With a DenseNet3 (depth 16) we achieved an accuracy of 66.7 $\pm$ 3.5 whereas NFM achieves a much higher accuracy of 74.95 $\pm$ 1.58. When we doubled the number of channels and the number of layers the accuracy slightly improved to **76.5** $\pm$ **1.1** (with NFM) and 69.21 $\pm$ 1.36 (without NFM), suggesting that the improvement from simply adding more capacity is much smaller than the improvement from using NFM.

On the relational reasoning task, we tried using one-pass $\mathcal{K} = 1$, and we achieved test accuracy of 73.07 ± 1.17 on relational questions, which is better than the baseline's 68.26 ± 0.61 accuracy, but worse than the NFM with two-passes $\mathcal{K} = 2$ (accuracy of 74.95 ± 1.58 on relational questions). We also saw an improvement thanks to introducing attention sparsity. In the reported results, both baseline CNN and NFM(CNN) use 24 initial channels.

Additionally, we tried replacing the normal re-scaling process (Section 3.2.2) from NFM and replaced it with a simple nearest-neighbor re-scaling. On CIFAR-10 PreActResNet18 classification, the average test accuracy dropped from 96.56% to 96.47%

5 Conclusion

The central concept behind the success of deep learning is that models should consist of multiple components (layers), each performing specific functions. Many of the most successful ideas in deep learning have the purpose of allowing individual layers to serve a more specific and incremental role. However, we note that most neural networks still process all of the layers in a fixed sequence, giving each layer the previous layer as input. We have instead proposed an alternative setup, which we called Neural Function Modules (NFM), which is inspired by functions in programming languages, which operate over specific arguments. Our main contribution lies in a new algorithm design, which connects several ideas that are important in deep learning (attention, sparsity, specialized modules, top-down and bottom-up feedback, long-range dependencies). The proposed implementation of these ideas (Neural Function Modules) is a generic and highly flexible architecture. While it is improved by NFM, increased standard test accuracy on classification tasks such as ImageNet was not the main focus of our work. We have shown that the proposed method substantially improves the performance across many different tasks (including systematic generalization), and we have shown ways in which this opens up new opportunities for architecture design - by removing the constraint that each layer must serve as input for the successive layer.

References

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

[Antoniou et al., 2018] Antoniou, A., Słowik, A., Crowley, E. J., and Storkey, A. (2018). Dilated densenets for relational reasoning. *arXiv preprint arXiv:1811.00410*.

[Bahdanau et al., 2018] Bahdanau, D., Murty, S., Noukhovitch, M., Nguyen, T. H., de Vries, H., and Courville, A. (2018). Systematic generalization: what is required and can it be learned? *arXiv preprint arXiv:1811.12889*.

[Bengio, 2013] Bengio, Y. (2013). Deep learning of representations: Looking forward. In *International Conference on Statistical Language and Speech Processing*, pages 1–37. Springer.

[Bengio et al., 2019] Bengio, Y., Deleu, T., Rahaman, N., Ke, R., Lachapelle, S., Bilaniuk, O., Goyal, A., and Pal, C. (2019). A meta-transfer objective for learning to disentangle causal mechanisms. *arXiv preprint arXiv:1901.10912*.

[Chen et al., 2018a] Chen, R. T. Q., Rubanova, Y., Bettencourt, J., and Duvenaud, D. K. (2018a). Neural ordinary differential equations. In Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R., editors, *Advances in Neural Information Processing Systems 31*, pages 6571–6583. Curran Associates, Inc.

[Chen et al., 2019] Chen, T., Zhai, X., Ritter, M., Lucic, M., and Houlsby, N. (2019). Self-supervised gans via auxiliary rotation loss. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 12154–12163.

[Chen et al., 2018b] Chen, Z., Li, Y., Bengio, S., and Si, S. (2018b). Gaternet: Dynamic filter selection in convolutional neural network via a dedicated global gating network. *arXiv preprint arXiv:1811.11205*.

[Child et al., 2019] Child, R., Gray, S., Radford, A., and Sutskever, I. (2019). Generating long sequences with sparse transformers.

[Dabney et al., 2018] Dabney, W., Ostrovski, G., Silver, D., and Munos, R. (2018). Implicit quantile networks for distributional reinforcement learning. *arXiv preprint arXiv:1806.06923*.

[DeVries and Taylor, 2017] DeVries, T. and Taylor, G. W. (2017). Improved regularization of convolutional neural networks with cutout. *arXiv preprint arXiv:1708.04552*.

[Fernando et al., 2017] Fernando, C., Banarse, D., Blundell, C., Zwols, Y., Ha, D., Rusu, A. A., Pritzel, A., and Wierstra, D. (2017). Pathnet: Evolution channels gradient descent in super neural networks. *arXiv preprint arXiv:1701.08734*.

[Fyall et al., 2017] Fyall, A. M., El-Shamayleh, Y., Choi, H., Shea-Brown, E., and Pasupathy, A. (2017). Dynamic representation of partially occluded objects in primate prefrontal and visual cortex. *Elife*, 6:e25784.

[Goyal et al., 2019] Goyal, A., Lamb, A., Hoffmann, J., Sodhani, S., Levine, S., Bengio, Y., and Schölkopf, B. (2019). Recurrent independent mechanisms. *arXiv preprint arXiv:1909.10893*.

[Greff et al., 2019] Greff, K., Kaufmann, R. L., Kabra, R., Watters, N., Burgess, C., Zoran, D., Matthey, L., Botvinick, M., and Lerchner, A. (2019). Multi-object representation learning with iterative variational inference. *arXiv preprint arXiv:1903.00450*.

[He et al., 2016a] He, K., Zhang, X., Ren, S., and Sun, J. (2016a). Deep residual learning for image recognition. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[He et al., 2016b] He, K., Zhang, X., Ren, S., and Sun, J. (2016b). Identity mappings in deep residual networks. In *European conference on computer vision*, pages 630–645. Springer.

[Hessel et al., 2018] Hessel, M., Modayil, J., Van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., Horgan, D., Piot, B., Azar, M., and Silver, D. (2018). Rainbow: Combining improvements in deep reinforcement learning. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

[Heusel et al., 2017] Heusel, M., Ramsauer, H., Unterthiner, T., Nessler, B., and Hochreiter, S. (2017). Gans trained by a two time-scale update rule converge to a local nash equilibrium. In *Advances in neural information processing systems*, pages 6626–6637.

[Hu et al., 2018] Hu, J., Shen, L., Albanie, S., Sun, G., and Vedaldi, A. (2018). Gather-excite: Exploiting feature context in convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 9401–9411.

[Huang et al., 2017] Huang, G., Liu, Z., Van Der Maaten, L., and Weinberger, K. Q. (2017). Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708.

[Jastrzębski et al., 2017] Jastrzębski, S., Arpit, D., Ballas, N., Verma, V., Che, T., and Bengio, Y. (2017). Residual connections encourage iterative inference. *arXiv preprint arXiv:1710.04773*.

[Ke et al., 2018] Ke, N. R., GOYAL, A. G. A. P., Bilaniuk, O., Binas, J., Mozer, M. C., Pal, C., and Bengio, Y. (2018). Sparse attentive backtracking: Temporal credit assignment through reminding. In *Advances in neural information processing systems*, pages 7640–7651.

[Kim et al., 2018] Kim, S., Hong, J., Kang, I., and Kwak, N. (2018). Semantic sentence matching with densely-connected recurrent and co-attentive information. *CoRR*, abs/1805.11360.

[Kwot Sin Lee and Cheung, 2019] Kwot Sin Lee, N.-T. T. and Cheung, N.-M. (2019). Infomax-gan: Mutual information maximization for improved adversarial image generation.

[Lamb et al., 2017] Lamb, A. M., Hjelm, D., Ganin, Y., Cohen, J. P., Courville, A. C., and Bengio, Y. (2017). Gibbsnet: Iterative adversarial inference for deep graphical models. In *Advances in Neural Information Processing Systems*, pages 5089–5098.

[Lee and Town, 2020] Lee, K. S. and Town, C. (2020). Mimicry: Towards the reproducibility of gan research.

[Lieber, 2019] Lieber, O. (2019). Rltime: A reinforcement learning library for state-of-the-art q-learning. `https://github.com/opherlieber/rltime`.

[Marino et al., 2018] Marino, J., Yue, Y., and Mandt, S. (2018). Iterative amortized inference. *arXiv preprint arXiv:1807.09356*.

[McGill and Perona, 2017] McGill, M. and Perona, P. (2017). Deciding how to decide: Dynamic routing in artificial neural networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 2363–2372. JMLR. org.

[Miyato et al., 2018] Miyato, T., Kataoka, T., Koyama, M., and Yoshida, Y. (2018). Spectral normalization for generative adversarial networks. *arXiv preprint arXiv:1802.05957*.

[Rosenbaum et al., 2017] Rosenbaum, C., Klinger, T., and Riemer, M. (2017). Routing networks: Adaptive selection of non-linear functions for multi-task learning. *arXiv preprint arXiv:1711.01239*.

[Salakhutdinov and Hinton, 2009] Salakhutdinov, R. and Hinton, G. (2009). Deep boltzmann machines. In *Artificial intelligence and statistics*, pages 448–455.

[Salimans et al.,] Salimans, T., Goodfellow, I., Zaremba, W., Cheung, V., Radford, A., and Chen, X. Improved techniques for training gans. arxiv 2016. *arXiv preprint arXiv:1606.03498*.

[Santoro et al., 2017a] Santoro, A., Raposo, D., Barrett, D. G., Malinowski, M., Pascanu, R., Battaglia, P., and Lillicrap, T. (2017a). A simple neural network module for relational reasoning. In *Advances in neural information processing systems*, pages 4967–4976.

[Santoro et al., 2017b] Santoro, A., Raposo, D., Barrett, D. G., Malinowski, M., Pascanu, R., Battaglia, P., and Lillicrap, T. (2017b). A simple neural network module for relational reasoning. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Advances in Neural Information Processing Systems 30*, pages 4967–4976. Curran Associates, Inc.

[Shazeer et al., 2017] Shazeer, N., Mirhoseini, A., Maziarz, K., Davis, A., Le, Q., Hinton, G., and Dean, J. (2017). Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538*.

[Spoerer et al., 2017] Spoerer, C. J., McClure, P., and Kriegeskorte, N. (2017). Recurrent convolutional neural networks: a better model of biological object recognition. *Frontiers in psychology*, 8:1551.

[Veit and Belongie, 2018] Veit, A. and Belongie, S. (2018). Convolutional networks with adaptive inference graphs. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 3–18.

[Verma et al., 2018] Verma, V., Lamb, A., Beckham, C., Najafi, A., Mitliagkas, I., Courville, A., Lopez-Paz, D., and Bengio, Y. (2018). Manifold mixup: Better representations by interpolating hidden states. *arXiv preprint arXiv:1806.05236*.

[Vezhnevets et al., 2016] Vezhnevets, A., Mnih, V., Osindero, S., Graves, A., Vinyals, O., Agapiou, J., et al. (2016). Strategic attentive writer for learning macro-actions. In *Advances in neural information processing systems*, pages 3486–3494.

[Wang et al., 2018] Wang, X., Yu, F., Dou, Z.-Y., Darrell, T., and Gonzalez, J. E. (2018). Skipnet: Learning dynamic routing in convolutional networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 409–424.

[Woo et al., 2018] Woo, S., Park, J., Lee, J.-Y., and So Kweon, I. (2018). Cbam: Convolutional block attention module. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 3–19.

[Wu et al., 2019] Wu, Y., Donahue, J., Balduzzi, D., Simonyan, K., and Lillicrap, T. (2019). Logan: Latent optimisation for generative adversarial networks.

[Wu et al., 2018] Wu, Z., Nagarajan, T., Kumar, A., Rennie, S., Davis, L. S., Grauman, K., and Feris, R. (2018). Blockdrop: Dynamic inference paths in residual networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8817–8826.

[Zamir et al., 2017] Zamir, A. R., Wu, T.-L., Sun, L., Shen, W. B., Shi, B. E., Malik, J., and Savarese, S. (2017). Feedback networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1308–1317.

[Zhang et al., 2017] Zhang, H., Cisse, M., Dauphin, Y., and Lopez-Paz, D. (2017). mixup: Beyond empirical risk minimization. iclr 2018. *arXiv preprint arXiv:1710.09412*.

[Zhang et al., 2018] Zhang, H., Goodfellow, I., Metaxas, D., and Odena, A. (2018). Self-attention generative adversarial networks. *arXiv preprint arXiv:1805.08318*.

A Classification Task

We considered classification on CIFAR-10, CIFAR-100, Tiny-Imagenet, and Imagenet. We consider variants on the PreActResNet architecture [He et al., 2016b]. For CIFAR-10, CIFAR-100, Tiny-Imagenet, and Imagenet and followed the same hyperparameters and configuration as the input mixup baseline in [Verma et al., 2018]. On all datasets except for Imagenet, we trained for 600 epochs, with a starting learning rate of 0.1, and dropped the learning rate by 10x at 200 epochs, 400 epochs, and 500 epochs. We averaged our obtained test accuracy over the last 10 epochs. We used input mixup with a rate of $\alpha = 1.0$ [Zhang et al., 2017], except on Imagenet, where we used $alpha = 0.5$. For these experiments, we used a keysize of 32 and valsize of 32, with 4 heads. We integrated NFM modules after each residual block. Thus we used 8 NFM modules (4 in the first pass and 4 in the second pass). We used a top-k sparsity for the attention of $k = 5$.

We also investigated if a single pass of NFM improves the accuracy regardless of the number of parameters in the model. We used the recent image classification task from `https://github.com/ElementAI/synbols`. Preliminary results are in Table 10.

Table 10: Default Synbols dataset (100k examples).

Num parameters	Num inital planes	NFM?	Test Accuracy (%)
206k	12	Yes	83.14
177k	12	No	81.89
757k	24	Yes	84.03
698k	24	No	82.61
5M	64	Yes	87.63
4M	64	No	85.2

While more investigation is needed, in image classification adding NFM seems to scale better (in terms of test accuracy) than increasing the model size by adding more layers. Without NFM, the accuracy increases by 0.72% at the cost of $698k - 177k = 521k$ additional parameters, whereas by adding NFM we get an increase of 1.25% at the cost of adding $206k - 177k = 29k$ parameters. The effect persists for $12, 24, 64$ initial planes (Table 10).

B Stacked MNIST Generalization

For stacked mnist, we consider a base CNN with nine convolutional layers, with every other layer having a stride of two. Each convolutional layer had a kernel size of 3. The first convolutional layer had 32 channels, which we doubled every time the resolution is reduced, leading the final number of channels to be 512. Each batch contains a certain number of digits per image, either (1 or 3) digits or (1 or 5) digits. As a result of the batches having variable characteristics, we elected to remove the batch normalization layer when training on this task.

We trained all models with Adam with a learning rate of 0.001 for 300 epochs, and report the test accuracy from the epoch with the highest validation accuracy.

The images in our stacked MNIST task are 64x64, and some examples with 5 mnist digits are shown in Figure 3. Each digit is reduced to a size of 16x16 by nearest-neighbor downsampling and then pasted into a random position within the frame.

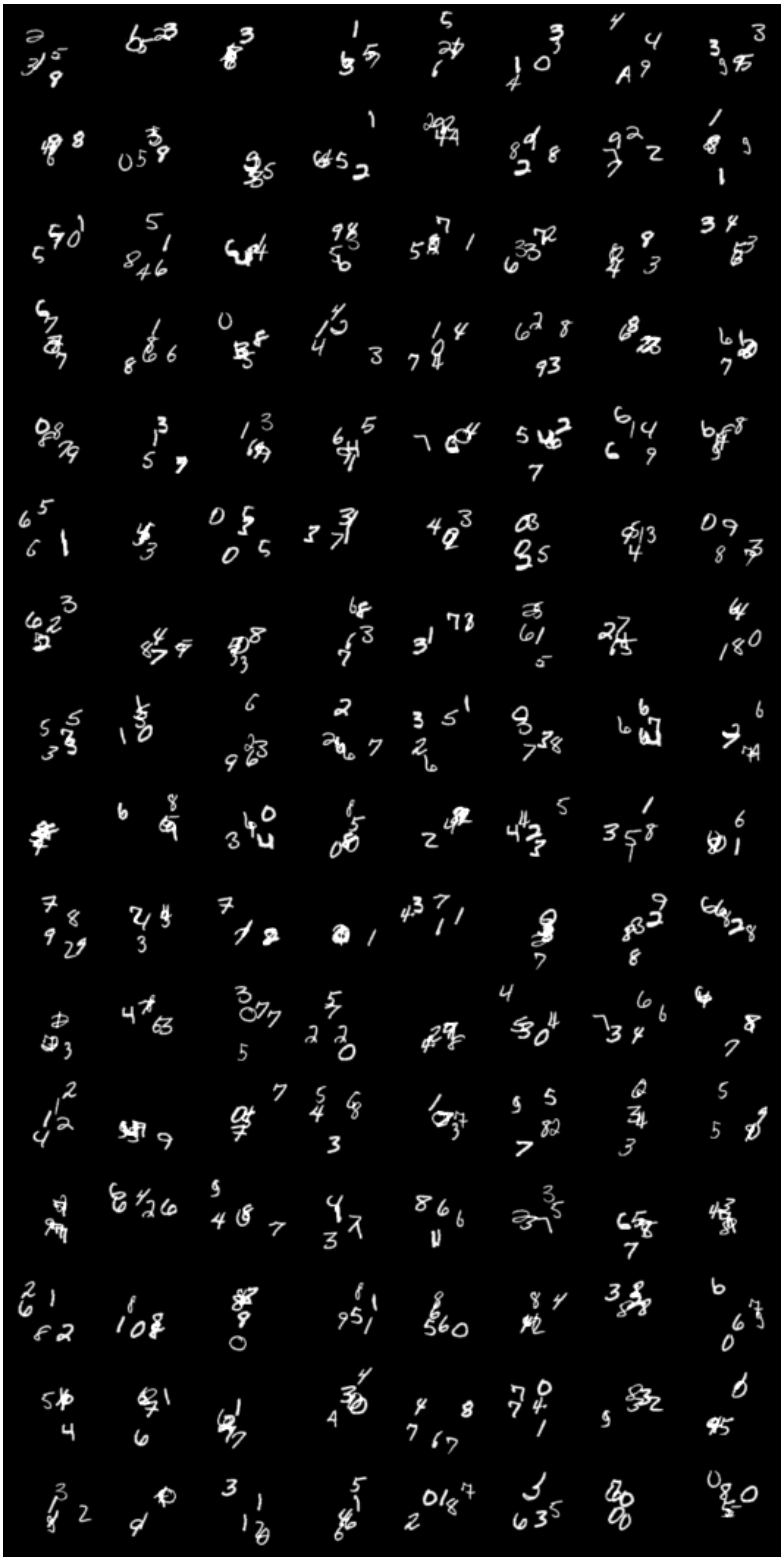


Figure 3: Examples of the stacked mnist digit dataset with 5 digits per image.

C Relational Reasoning

Figure 2 shows a sample (image, question) from the Sort-of-CLEVR dataset. Each image is paired with 10 relational and 10 non-relational questions. We use the exact CNN baseline architecture from [Antoniou et al., 2018] along with the same experimental setup. We train all models for 250 epochs with the Adam optimizer with a learning rate of 0.001.

D Computational Resources

Resources Used: It takes about 4 days to train the proposed model on Atari RL benchmark task for 50M timesteps.