Université de Montréal

Latent Data Augmentation and Modular Structure for Improved Generalization

par

Alex Lamb

Département d'informatique et de recherche opérationnelle
Faculté des arts et des sciences

Thèse présentée en vue de l'obtention du grade de
Philosophiæ Doctor (Ph.D.)
en Discipline

August 22, 2022

Université de Montréal

Faculté des arts et des sciences

Cette thèse intitulée

Latent Data Augmentation and Modular Structure for Improved Generalization

présentée par

Alex Lamb

a été évaluée par un jury composé des personnes suivantes :

Ioannis Mitliagkas
(président-rapporteur)

Yoshua Bengio
(directeur de recherche)

Dhanya Sridhar
(membre du jury)

Rob Fergus, NYU
(examinateur externe)

(représentant du doyen de la FESP)

Sommaire

Cette thèse explore la nature de la généralisation dans l'apprentissage en profondeur et plusieurs contextes dans lesquels elle échoue. En particulier, les réseaux de neurones profonds peuvent avoir du mal à se généraliser dans des contextes avec des données limitées, une supervision insuffisante, des dépendances à longue portée difficiles ou une structure et des sous-systèmes complexes.

Cette thèse explore la nature de ces défis pour la généralisation en apprentissage profond et présente plusieurs algorithmes qui cherchent à relever ces défis.

Dans le premier article, nous montrons comment l'entraînement avec des états cachés interpolés peut améliorer la généralisation et la calibration en apprentissage profond. Nous introduisons également une théorie montrant comment notre algorithme, que nous appelons Manifold Mixup, conduit à un aplatissement des représentations cachées par classe, ce qui peut être vu comme une compression de l'information dans les états cachés.

Le deuxième article est lié au premier et montre comment des exemples interpolés peuvent être utilisés pour un apprentissage semi-supervisé. Outre l'interpolation des exemples d'entrée, les prédictions interpolées du modèle sont utilisées comme cibles pour ces exemples. Cela améliore les résultats sur les benchmarks standard ainsi que sur les problèmes de jouets 2D classiques pour l'apprentissage semi-supervisé.

Le troisième article étudie comment un réseau de neurones récurrent peut être divisé en plusieurs modules avec des paramètres différents et des états cachés bien séparés, ainsi qu'un mécanisme de concurrence limitant la mise à jour des états cachés à un sous-ensemble des modules les plus pertinents sur un pas de temps spécifique. . Cela améliore la généralisation systématique lorsque la distribution des modèles est modifiée entre les phases de entraînement et d'évaluation. Il améliore également la généralisation dans l'apprentissage par renforcement.

Dans le quatrième article, nous montrons que l'attention peut être utilisée pour contrôler le flux d'informations entre les couches successives des réseaux profonds. Cela permet à chaque couche de ne traiter que le sous-ensemble des sorties des couches précédemment calculées qui sont les plus pertinentes. Cela améliore la généralisation sur les tâches de raisonnement relationnel ainsi que sur les tâches de classification de référence standard.

Mots-clés: Apprentissage en profondeur, traitement du langage naturel, apprentissage des représentations, modèles génératifs, modélisation du langage

Summary

This thesis explores the nature of generalization in deep learning and several settings in which it fails. In particular, deep neural networks can struggle to generalize in settings with limited data, insufficient supervision, challenging long-range dependencies, or complex structure and subsystems.

This thesis explores the nature of these challenges for generalization in deep learning and presents several algorithms which seek to address these challenges.

In the first article, we show how training with interpolated hidden states can improve generalization and calibration in deep learning. We also introduce a theory showing how our algorithm, which we call Manifold Mixup, leads to a flattening of the per-class hidden representations, which can be seen as a compression of the information in the hidden states.

The second article is related to the first and shows how interpolated examples can be used for semi-supervised learning. In addition to interpolating the input examples, the model's interpolated predictions are used as targets for these examples. This improves results on standard benchmarks as well as classic 2D toy problems for semi-supervised learning.

The third article studies how a recurrent neural network can be divided into multiple modules with different parameters and well separated hidden states, as well as a competition mechanism restricting updating of the hidden states to a subset of the most relevant modules on a specific time-step. This improves systematic generalization when the pattern distribution is changed between the training and evaluation phases. It also improves generalization in reinforcement learning.

In the fourth article, we show that attention can be used to control the flow of information between successive layers in deep networks. This allows each layer to only process the subset of the previously computed layers' outputs which are most relevant. This improves generalization on relational reasoning tasks as well as standard benchmark classification tasks.

Keywords: Deep Learning, Natural Language Processing, Representation Learning, Generative Models, Language Modeling

Contents

List of tables

List of figures

List of Acronyms and Abbreviations

BPTT	Backpropagation Through Time
CNN	Convolutional Neural Network
ERM	Empirical Risk Minimization
GAN	Generative Adversarial Network
GRU	Gated Recurrent Unit
LSTM	Long Short-Term Memory
MLE	Maximum Likelihood Estimation
MLP	Multi-Layer Perceptron
NLP	Natural Language Processing
NMT	Neural Machine Translation
RNN	Recurrent Neural Network
SGD	Stochastic Gradient Descent
BPTT	Backpropagation Through Time

VAE	Variational Autoencoder

Acknowledgements

I'd like to acknowledge Anirudh Goyal, Rosemary Nan Ke, Chiheb Trabelsi, Michael Mozer, Vikas Verma, Sherjil Ozair, Riashat Islam, Kenji Kawaguchi, Tarin Clanuwat, Asanobu Kitamoto, Yoshua Bengio, Aaron Courville, Martin Arjovsky, Kari Torkkola, Dominique Perrault-Joncas, Amir Najafi, Ishmael Belghazi, and Mark Dredze for their enormous help in the research which allowed me to write this thesis.

Chapter 1

Introduction

State-of-the-art AI systems achieve excellent results on supervised learning tasks where lots of cleanly labeled data is available and the training and evaluation datasets are sampled from the same distribution. While such systems have achieved impressive results, they fall short of some of the more ambitious goals of AI research: learning agents which can be dropped into a completely new environment and immediately gain an understanding of their situation from only a few unlabeled observations. Such an agent would not only learn quickly, but would also adapt quickly when the environment changes while still correctly reasoning about long-range dependencies between events. The aim of the work in this thesis is to work towards developing such a learning system, and to do so concretely, while closely interacting with real-world problems where these issues are salient.

Many of today's machine learning systems perform well, but only when trained on very large and cleanly labeled datasets. By contrast, humans are able to learn from data which is almost entirely unlabeled, and learn new concepts with only a handful of examples. One widely explored way to improve the data-efficiency of machine learning techniques is by training with augmented data. For example in images, it is common to include cropped or rotated images in the training set. While this improves performance, it has a few deficiencies. One is that the changes are fairly superficial, and do not do much to add diversity to the data. Another is that it requires domain knowledge to hand-craft these augmentations, which are not available in fields where humans may not have good intuitions. An algorithm presented in this thesis that was developed to help address this challenge, Manifold Mixup (Verma et al., 2018) aims to improve data efficiency by training on synthetic examples which combine the hidden state of multiple different examples. This algorithm also has an intriguing theory, in which training on combinations of hidden states pushes them onto a lower-dimensional manifold, which itself has been studied in a follow-up paper (Roth et al., 2020). Manifold Mixup has had a direct follow-up which achieved state-of-the-art results on few-shot learning, in which a model must learn from only a handful of labeled examples (Mangla et al., 2019).

Another idea for how to improve the data efficiency of deep networks is semi-supervised learning, in which a model is trained on both a small amount of labeled data and a large volume of unlabeled data. One motivating strategy for semi-supervised learning is the cluster assumption, which is that the decision boundary should only pass through low density regions of the data distribution. An algorithm that my colleagues and I developed,

Interpolation Consistency Training (ICT), sought to utilize this assumption by training on linear interpolations from multiple unlabeled examples (Verma et al., 2019a). This approach does not require any hand-crafted data augmentations, and has been influential both within academia and in real-world applications. Another follow-up from our group, GraphMix (Verma et al., 2019b) seeks to extend this idea to node-classification in graph structured data, and has state-of-the-art results on some of the most widely studied graph datasets.

Another approach to improving sample efficiency is to train a generative models on the hidden states of deep networks and thus learn to project back to the points that the model understands. Our algorithm for this, which we called State-Reification Networks (ICML 2019 oral, 5% acceptance rate, first author) achieved substantial improvements in data efficiency as well as robustness [14].

An impressive aspect of human learning is the ability to gain an understanding of the world without any labels or direct supervision. Unsupervised Learning is a challenging and open problem, yet offers the promise of machine learning systems which can improve and gain understanding purely by interacting with natural data from the real world.

A fundamental limitation in most machine learning systems is that they rely on the assumption that training and test samples are "independent and identically distributed" (the i.i.d. assumption). On the other hand, humans clearly have a more powerful ability to generalize in a systematic way, including to examples which have zero probability under the training distribution.

One branch of my research explores how we can improve the generalization performance in deep learning models by decomposing the knowledge into independent mechanisms or modules. The idea of an independent mechanism, which is taken from the causality literature, is a distribution over some random variables which remain unchanged when other unrelated aspects of the world are changed. As an example, if we have a room with two different televisions, we know that what is playing on one television will have almost no relation to what is playing on the other television, and moreover is usually not effected by the people in the room watching the television (they could turn the television off or change the channel, but usually have no effect). The idea that the world consists of multiple elements which interact sparingly but operate independently by default is intuitive and is applicable to many settings, yet today's deep learning algorithms nearly always employ the opposite inductive bias: that all aspects of the world always interact so long as they are temporally and spatially adjacent. For example, a recurrent neural network and a convolutional neural network, by default, share information between all different hidden units at a given position. Having parts of the state remain perfectly well-separated requires learning a parameter vector where nearly every element is exactly zero, which the learning procedure has no inductive bias to do. As a result, these models often learn a separation between different independent mechanisms just well enough to fit the training set, yet fail to capture this separation when the data distribution changes, leading to poor systematic generalization.

We investigated this problem concretely in the Recurrent Independent Mechanisms (RIMs) paper (Goyal et al., 2019), in which we showed that there are several extremely simple sequence problems in which today's state-of-the-art sequence models fail to achieve

systematic generalization. To address this, we proposed an architecture in which the hidden state and parameters of a recurrent model are broken into several modules (capturing independent mechanisms) which are encouraged to specialize over distinct input patterns and only sparingly share information amongst themselves via a bottleneck of attention. The depth of this line of research is demonstrated by a direct follow-up on Bi-Directional Recurrent Independent Mechanisms (Mittal et al., 2020) as well as a paper on learning how to share the recurrent parameters between mechanisms dynamically (Goyal et al., 2020). While this work has inspired a significant amount of research on new algorithms, there is still a great deal of room to develop a more rigorous theory and to apply these improved techniques to achieve impact on real-world problems.

1.1. Thesis Overview

The thesis contains 4 articles that focus on latent interpolation and modular structure for generalization in deep learning.

Chapter 2 starts with an introduction to the field of Machine Learning and how maximum likelihood estimation can be used to frame supervised learning in a probabilistic framework. It then discusses challenges in generalization in deep learning and an overview of regularizers which can help to improve generalization. It also discusses a few architectures, and how they relate to modularity and generalization.

Chapter 4 presents a latent state interpolation algorithm which provably flattens the learned representations onto a lower-dimensional subspace. Despite requiring almost no additional computation, it substantially improves results on standard benchmarks like CIFAR-10, SVHN, and Tiny-Imagenet. It also shows substantial improvement on test set likelihood.

Chapter 6 explores the use of interpolations for semi-supervised learning. It uses mixes of the model's own predictions as targets for training mixed data points.

Chapter 8 shows how breaking a recurrent neural network into multiple modules with separate hidden states and separate parameters can substantially improve systematic generalization. It also explores a top-down competition mechanism, where the modules which have the strongest drive to read from the input become selectively activated. It dramatically improves specialization over distinct sub-patterns in data by allowing modules to remain dormant on time steps where they are not relevant.

Chapter 10 The Neural Function Modules (NFM) technique explored a new way of passing information between different layers in deep networks by using attention. In the NFM technique, each layer produces a query vector and attends over all previously computed layers, as input for the layer. This allows for better specialization of the layers and allows information to be routed around layers for which it is not relevant. This led to improved results on classification benchmarks and relational reasoning.

Chapter 11 presents an overall conclusion of the contributions in this thesis and presents some shortcomings of our work.

Chapter 2

Background

2.1. Probabilistic Machine Learning

Machine learning systems are able to make predictions or produce actions by automatically learning from data and experience. Mitchell et al. (1997) define a learning machine as an algorithm that improves with experience on a particular task, given a certain performance measure to evaluate the system. In this thesis, we are interested in questions of how machine learning systems can generalize either when trained with limited data or limited task diversity.

We often assume the ability to independently sample a certain finite number of datapoints from the same unknown data distribution. Some of that data is partitioned for our systems to learn on and the rest is reserved for testing how it generalizes on unseen examples. Generalizing to data drawn from the same distribution is a fairly common evaluation setup used in most machine learning systems. Most theoretical guarantees on the generalization properties of machine learning systems have required this assumption. This is however being questioned more as our systems do not generalize systematically and perform poorly when dealing with out-of-distribution data. For the sake of this introduction however, we will focus on the case where our data is independent and identically distributed (iid).

2.1.1. Supervised Learning

Supervised Learning serves as the fundamental framework for most of the work in this thesis and much of machine learning research as a whole. It consists of learning to predict output targets from inputs given a labeled dataset of input/output pairs, where all pairs are assumed to be sampled uniformly from an unknown distribution. An appealing aspect of supervised learning is that the learned model's accuracy on held-out samples is an objective measure of success. Additionally, the correct outputs for the model are provided in the dataset, which makes credit assignment relatively straightforward. Compare this to playing a video game, where the correct actions are never given to the player.

Supervised learning entails learning a map from inputs to well-defined outputs collected in a dataset of known input/output pairs. Some examples of supervised learning include image classification (where the image is the input and the class-label is the output) and speech recognition (where the speech signal is the input and the corresponding text is the output).

The goal of supervised learning is to learn a function f_θ which maps an example's input to its output. These functions f_θ map a d-dimensional, typically real-valued vector input $\mathbf{x} \in \mathbb{R}^d$ from the space of possible inputs X to a discrete scalar output label y from the

space of k possible outputs Y. We assume our data comes from an unknown joint probability distribution over inputs and outputs $p(\mathbf{x}, y)$ where $\mathbf{x} \in X$ and $y \in Y$ and we would like to learn a function $f_\theta : \mathbb{R}^d \to \mathbb{R}^k$ that maps X to an estimate of $P(Y|X)$. We observe examples from this distribution $((\mathbf{x_1}, y_1) \dots (\mathbf{x_N}, y_N))$. The subscript θ is used to indicate that our function is parameterized. This doesn't always have to be the case, and there exist "non-parameteric" models in machine learning, but in the context of this thesis that focuses on neural networks, we are interested in learning parametric functions. There are many possible functions that can map from X to a distribution over Y, but we are often interested in learning one that makes as few mistakes as possible. We define a loss function $L(\hat{y}, y)$ that measures the penalty incurred when classifying y as $\hat{y}$ and we try to optimize f_θ to minimize this loss given a finite amount of data. We can use this loss to define the empirical risk associated with a particular function as the expected loss over the entire training set $R_{emp}(X, Y, f_\theta) = \frac{1}{N} \sum_{i=1}^{N} L(f_\theta(\mathbf{x_i}), y_i)$. For a given dataset, there may be many functions $f \in F$ in the hypothesis space, that achieve the same or even 0 empirical risk on the training data subset, but generalize poorly to unseen examples. In such cases, we want to bias our learners to certain solutions that may have better generalization properties such as those with low parameter norms with L1/L2 regularization, with max-margin methods (Vapnik, 1963), or by using dropout (Srivastava et al., 2014). Empirical Risk Minimization (ERM) tries to find $\text{argmin}_{f \in F} R_{emp}(f)$.

$f_\theta(\mathbf{x})$ outputs a probability distribution over possible class labels and the loss function used is typically the negative log-likelihood of the correct class $-\log q_\theta(Y = y|\mathbf{x})$. Intuitively, we wish to maximize the probability the model assigns to the correct class, and the use of the log causes a very high loss when a small probability is assigned to the correct class. The frequentist approach to parameter estimation termed Maximum Likelihood Estimation (MLE) finds a setting of parameters that maximizes the likelihood of the observed data. This decomposes into a product over the examples in the case where their conditional distributions are independent and identically distributed (the i.i.d. assumption): $q_\theta(Y|X) = \prod_{i=1}^{N} q_\theta(y_i|\mathbf{x_i})$. Since the logarithm is a monotonically increasing function, we can maximize the $\log q_\theta(Y|X)$ and break the product into a bunch of sums $\log q_\theta(Y|X) = \sum_{i=1}^{N} \log q_\theta(y_i|\mathbf{x_i})$. To turn it into a minimization problem, we can minimize the negative log-likelihood of the data, showing that empirical risk minimization (ERM) with the negative log-likelihood loss corresponds to maximum-likelihood estimation of the parameters of the model. At inference/test time, the model outputs $\text{argmax}_{\tilde{y}} q_\theta(\tilde{y}|\mathbf{x}_i)$.

2.1.2. Distributions and Likelihood

So far, we have discussed generative modeling in qualitative terms. We want models which can simulate from the dynamics of the world. We want models that can synthesize realistic looking data. However, before going further it is useful to understand the probabilistic interpretation of generative model, which gives a formal framework for studying generative models. The essential idea is that we treat observations from the world as samples from a distribution $x, y \sim p(x, y)$. For example, we could consider the distribution over all human faces which can occur in reality to be $p(x)$ and consider each face as a sample with an associated label $y \sim p(y|x)$. If we have access to a recorded dataset (for example a set of

faces), we may also choose to treat these points as a finite collection of samples from this distribution.

At the same time, we can interpret our generative model as an estimating distribution $q_\theta(y|x)$, which is described by a set of parameters θ. Then we can frame generative modeling as trying to ensure that $p(y|x)$ and $q_\theta(y|x)$ become as similar as possible. Statistical divergences give a natural mathematical framework for this. A divergence is a function $D(p||q) : S \times S \to R$ taking two distributions p and q over a space of distributions S as inputs, with the properties:

$$D(p||q) \geq 0.0 \tag{2.1}$$

$$D(p||q) = 0.0 \iff p = q \tag{2.2}$$

Notably, there is no symmetry assumption, so in general $D(p||q) \neq D(q||p)$. The probabilistic approach to generative modeling frames learning as an optimization problem where the loss corresponds to a given divergence.

$$\mathcal{L}(\theta) = \underset{\theta}{\operatorname{argmin}}\, D(p(y|x)||q_\theta(y|x)). \tag{2.3}$$

2.1.3. Maximum Likelihood and KL-Divergence

We will now show how minimizing KL-divergence can be applied to the supervised learning setting introduced previously. In particular, we assume the ability to sample from $p(x, y)$ and wish to learn $q_\theta(y|x)$.

What is the right algorithm for finding a distribution $q_\theta(y|x)$ which minimizes a divergence between itself and $p(y|x)$? Before selecting the type of divergence to minimize, a natural question is to consider what types of expressions we are capable of optimizing, and work backwards to find a suitable divergence. In general, we only have access to samples from the distribution $p(x, y)$ and not any additional information about the distribution. At the same time, $q_\theta(y|x)$ is a model that we control, so it's reasonable to believe that we'll be able to design it so that it has a density that we can compute as well as the ability to draw samples.

The KL-divergence can be rewritten as an expression in which the only term that depends on the parameters is an expectation on $q_\theta(y|x)$ over samples from $p(x, y)$. Beginning with two distributions $p(x, y)$ (the empirical distribution) and $q_\theta(y|x)$ (the model distribution), we write the KL-divergence:

$$D_{KL}(p(y|x)||q_\theta(y|x)) = \mathop{\mathbb{E}}_{x\sim p(x)}\left[\int p(y|x)\log p(y|x)dy - \int p(y|x)\log q_\theta(y|x)dy\right] \tag{2.4}$$

$$= \mathop{\mathbb{E}}_{x\sim p(x)} \int p(y|x)\log\frac{p(y|x)}{q_\theta(y|x)}dy \tag{2.5}$$

$$= \mathop{\mathbb{E}}_{x,y\sim p(x,y)}\left[\log\frac{p(y|x)}{q_\theta(y|x)}\right] \tag{2.6}$$

$$= \mathop{\mathbb{E}}_{x,y\sim p(x,y)}[\log p(y|x) - \log q_\theta(y|x)] \tag{2.7}$$

$$= H_p(y|x) - \mathop{\mathbb{E}}_{x,y\sim p(x,y)}[\log q_\theta(y|x)] \tag{2.8}$$

Then we can show the maximum likelihood estimation for a set of N data points.

$$\theta^* = \arg\max_\theta \prod_{i=1}^{N} q_\theta(y_i|x_i) \tag{2.9}$$

$$= \arg\max_\theta \sum_{i=1}^{N} \log q_\theta(y_i|x_i) \tag{2.10}$$

$$= \arg\min_\theta \frac{1}{N} \sum_{i=1}^{N} -\log q_\theta(y_i|x_i) \tag{2.11}$$

$$\sim= \arg\min_\theta \mathop{\mathbb{E}}_{x_i,y_i\sim p(x,y)}[-\log q_\theta(y|x)] \tag{2.12}$$

The objective for maximum likelihood is maximizing the log-density $\log(q_\theta(y|x))$ over real data points sampled from the distribution $p(x, y)$ can be written as:

$$D_{KL}(p(y|x)||q(y|x)) = \mathop{\mathbb{E}}_{x\sim p(x)} \left[\int p(y|x) \log p(y|x) dy - \int p(y|x) \log q_\theta(y|x) dy \right] \tag{2.13}$$

$$= -H(p(y|x)) + \mathrm{CE}(p(y|x), q_\theta(y|x)) \tag{2.14}$$

Thus we can see that the KL-divergence decomposes into two terms 2.13: a cross-entropy term (likelihood) and a term for the entropy of the true data distribution. Because the entropy of the true data distribution doesn't depend on the estimator, the KL-divergence can be minimized by maximizing likelihood. Another useful consequence of this is that the entropy of the true data distribution can be estimated by such a generative model if it maximizes likelihood.

2.1.4. ERM and probabilistic framing of the learning problem

The empirical risk minimization consists of minimizing a loss function over a set of examples x with labels y sampled from a generally unknown distribution $p(x, y)$:

$$\theta^* = \mathop{\mathrm{argmin}}_\theta \sum_{x,y\sim p(x,y)} L(y, f(x)). \tag{2.15}$$

One general way of motivating empirical risk minimization is through maximum likelihood. Essentially, it can be shown that maximizing a model's likelihood on some data distribution is equivalent to minimizing an expectation of loss over samples from the data distribution, which can then be empirically estimated with a sum. The objective for maximum likelihood is maximizing the log-density $log(q_\theta(x))$ over real data points sampled from the distribution $p(x)$. The use of maximum likelihood training for generative models (such as sequence models) is an unconditional generalization of the use of conditional maximum likelihood for supervised learning.

2.1.5. Teacher Forcing

A specific variant of the maximum likelihood principle can be used for training sequence models. This form of training is known as teacher forcing (Williams & Zipser, 1989b), due to the use of the ground-truth samples y_t being fed back into the model to be conditioned on for

$$p(\hat{y}_{1:T}|X,\Theta) = \prod_{t=1}^{T} p(\hat{y}_t|y_{1:t-1},X,\Theta) \qquad p(\hat{y}_{1:T}|X,\Theta) = \prod_{t=1}^{T} p(\hat{y}_t|\hat{y}_{1:t-1},X,\Theta)$$

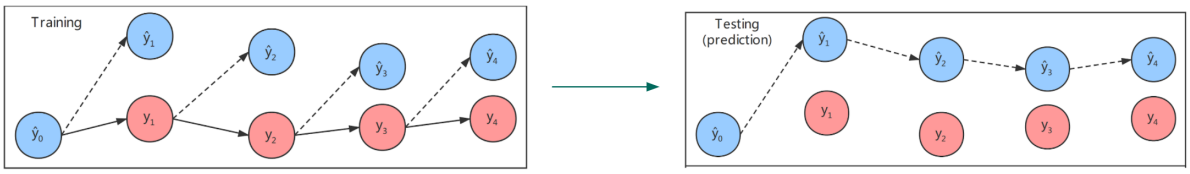


Fig. 1. The teacher forcing objective uses the ground-truth sequence during training (left), and uses generated sequence during sampling (right).

the prediction of later outputs. These fed back samples force the next-step-ahead predictions to stay close to the specific ground-truth sequence being trained on. The teacher forcing procedure can be formally justified by using the chain rule of probability. For example, in the case of three variables, this factorization is: $p(y_1, y_2, y_3) = p(y_3|y_1, y_2)p(y_2|y_1)p(y_1)$.

When performing prediction, the ground-truth sequence is not available conditioning and we sample from the joint distribution over the sequence by sampling each y_t from its conditional distribution given the previously generated samples. This procedure can result in poor generation over long sequences as small prediction errors compound over many steps of generation. This can lead to poor prediction performance as the sequence of previously generated samples diverges from observed sequences seen during training. This procedure is shown in Figure 1.

2.2. Regularization and Semi-Supervised Learning

Several techniques have been proposed to help improve generalization performance, which is especially useful when the amount of labeled data is limited. Some of these regularizations on a simple 2D spiral classification task are shown in Figure 2.

2.2.1. Vicinal Risk Minimization

(Chapelle et al., 2001) introduced the notion of "Vicinal Risk Minimization" (VRM), in which the loss is defined around the "vicinity" of training examples rather than just at the training examples themselves. (Chapelle et al., 2001) proposed to use gaussian kernels surrounding individual examples to define these vicinities. (Tong, 2000) proved that the VRM logistic regression using gaussian vicinity approaches the hard-margin SVM classifier as $\sigma \to 0$. This existing theory explores how training with synthetic examples changes the characteristics of the resulting model.

2.2.2. Hand-Crafted Data Augmentation

The size of a dataset can be increased by augmenting it with artificially constructed variants of the existing samples. For example, in a task where there is rotational invariance, augmenting the dataset by including rotated versions of the original samples, could lead to

a vastly larger augmented dataset. On the other hand, if the task does not have rotational invariance, then including rotated samples could degrade model performance. Training with hand-crafted data augmentation has a long and complex history in deep learning and machine learning in general. The largest strength of these methods is that they can perform quite well if the transformations which are appropriate for a task are well understood beforehand. For example, in image classification, it is typically (but not always) the case that images have horizontal symmetry. In such a case the correct prediction should not depend on whether it appears on the left or right side of an image. However in most natural images, such as scenes or everyday objects, there is not a vertical symmetry.

2.2.3. Weight Noise and Weight Decay

The idea of adding noise to the parameters of a deep network has a long history in neural network applications (An, 1996; Steijvers & Grünwald, 1996). Other variants have also been explored, such as adding Gaussian noise to the gradients of deep neural networks while training them (Neelakantan et al., 2015).

2.2.4. Dropout

(Srivastava et al., 2014) showed that randomly turning off a fraction of the hidden units in a deep neural network can be an effective regularizer. While the units are turned off during training, all the units are kept on during inference/prediction, with the units multiplied by $1/(1-p)$, where p is the probability of dropping an individual unit to compensate for the expected effect on the mean from dropping units.

2.2.5. Batch Normalization

(Ioffe & Szegedy, 2015a) explored the use of training deep neural networks with normalization of the mean and variance of units in the forward pass, while using additional learned parameters to control the mean and variance following standardization. The use of finite batches for normalization during training has been shown to achieve a regularization effect (Luo et al., 2018a).

2.2.6. Mixup

The mixup technique is straightforward to implement and understand, and at the same time is almost entirely model-agnostic.

At a high-level, mixup considers training with synthetic examples defined as linear combinations of examples from the training data.

$$\begin{aligned} \tilde{x} &= \lambda x_i + (1-\lambda)x_j, && \text{where } x_i, x_j \text{ are input vectors} \\ \tilde{y} &= \lambda y_i + (1-\lambda)y_j, && \text{where } y_i, y_j \text{ are one-hot targets} \end{aligned}$$

(x_i, y_i) and (x_j, y_j) are two examples drawn at random from the training data, and $\lambda \sim p(\lambda)$. As a simple starting point (which as we shall see, is surprisingly effective) we may use $p(\lambda) = U(0, 1)$.

We quickly note that in the particular case of the cross-entropy loss, averaging the soft-targets of two examples is equivalent to averaging the loss for two examples. Mixup is a work which strongly influences several of the new contributions in this thesis.

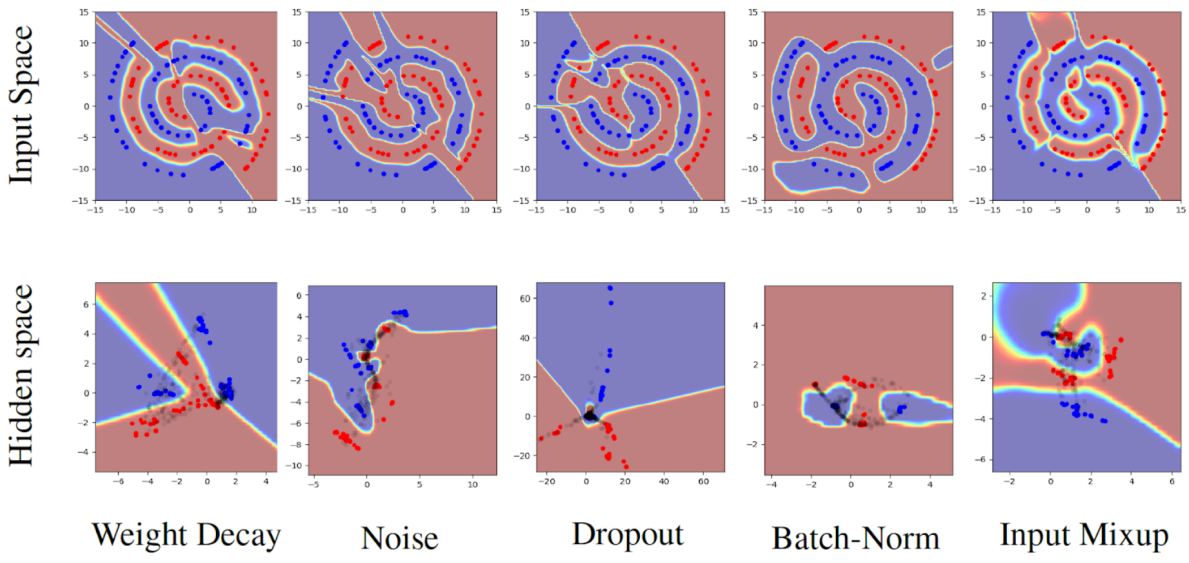


Fig. 2. A classification task with two interleaved spirals and a limited number of labeled examples (Figure from (Verma et al., 2018) with permission). The unregularized model makes many mistakes and is highly confident in those mistaken predictions. This high confidence also occurs when analyzing predictions in the hidden space.

2.3. Semi-Supervised Learning.

In many application domains, such as computer vision, speech, natural language understanding and medical diagnosis, Deep Neural Networks achieve human-level or even super-human-level prediction accuracy by leveraging a large collection of labeled data in a fully-supervised learning setup. However, there exits a caveat: Creating large collections of labeled data is often expensive or difficult. In this thesis, semi-supervised learning is the primary focus of the Interpolation Consistency Training technique (section 2.3.3), whereas Recurrent Independent Mechanisms (section 8.2) and Neural Function Modules (section 9.1) present general architectures which could be used for semi-supervised learning.

Current Deep Learning based Semi-Supervised Learning approaches can be broadly categorized into two streams (1) Deep Generative Modeling based approaches (Dumoulin et al., 2016b) and (2) Consistency Regularization based approaches (Verma et al., 2019a).

While the Deep Generative Modeling based approaches have achieved better results earlier, the recent progress in Semi-Supervised Learning has been primarily driven by the Consistency Regularization based methods. In this Chapter, in Section 2.3.1, we introduce the underlying assumptions and motivations of the Consistency Regularization approaches. Next, we describe some of the previous state-of-the-art Semi-Supervised Learning approaches based on Consistency Regularization.

2.3.1. Consistency Regularization

Consistency Regularization for semi-supervised learning is motivated by the low-density separation assumption (Chapelle et al., 2010). We first describe this assumption and then

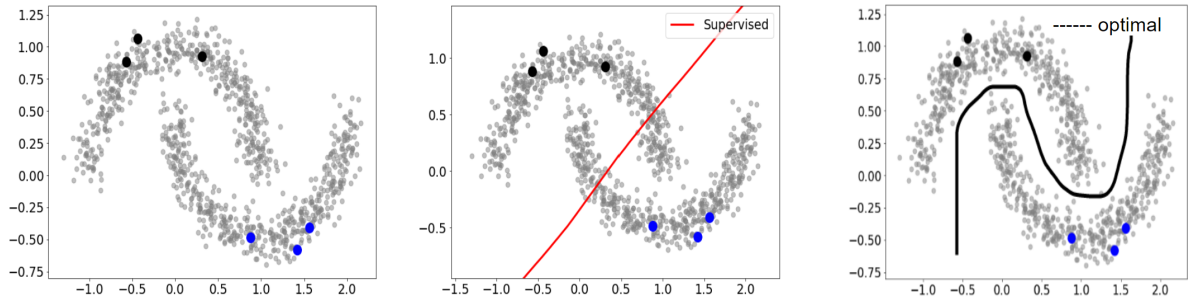


Fig. 3. On the two moons problem (left) unlabeled data is abundant but only six labeled examples are provided. A purely supervised solution fails to separate the two moons (center), while a decision boundary which avoids regions of high-density is able to separate the two moons (right).

present Consistency Regularization based Deep Semi-supervised Learning methods which utilize the cluster assumption.

The low-density separation assumption for classification problems assumes that the decision boundary is likely to traverse low-density regions in the sample space. This can also be equivalently seen as assuming that samples of the same class are connected in densely connected clusters: two samples in the same cluster will be classified to the same class if there is a densely populated path connecting samples from the same class.

Consistency-based semi-supervised learning methods enforce the low density separation assumption by encouraging that the output of the model does not change under small perturbations. The choice of these "small perturbations" requires careful design - but may include small amounts of Gaussian noise or random translation/cropping of images (data augmentation). By small perturbation, we mean perturbations that do not change the class-related semantics of the sample.

We can mathematically define self-supervised consistency regularization by assuming that $x \in \mathcal{D}_{UL}$ be an unlabeled sample, $\hat{x}$ is a perturbed version of the sample, and $f_\theta(x)$ is the prediction function parameterized by θ. Then we use a distance metric $D(a, b)$ which measures the difference between two predictions In practice this is often mean-squared error but could also be cross-entropy loss. Using these terms, the semi-supervised consistency regularization $R_\theta(x)$ can be defined as:

$$R_\theta(x) = D(f_\theta(x), f_\theta(\hat{x})) \tag{2.16}$$

The loss term used to train the prediction function $f_\theta(x)$ is obtained by adding the consistency regularizer of Equation 2.16 to the supervised classification loss weighted by a hyperparameter λ, which is often increased over the course of training. The consistency-based approach to semi-supervised learning has motivated a family of recent state-of-the-art methods, which includes temporal ensembling (Laine & Aila, 2016), Virtual Adversarial Training (VAT) (Miyato et al., 2018a), and the Mean-Teacher algorithm (Tarvainen & Valpola, 2017b).

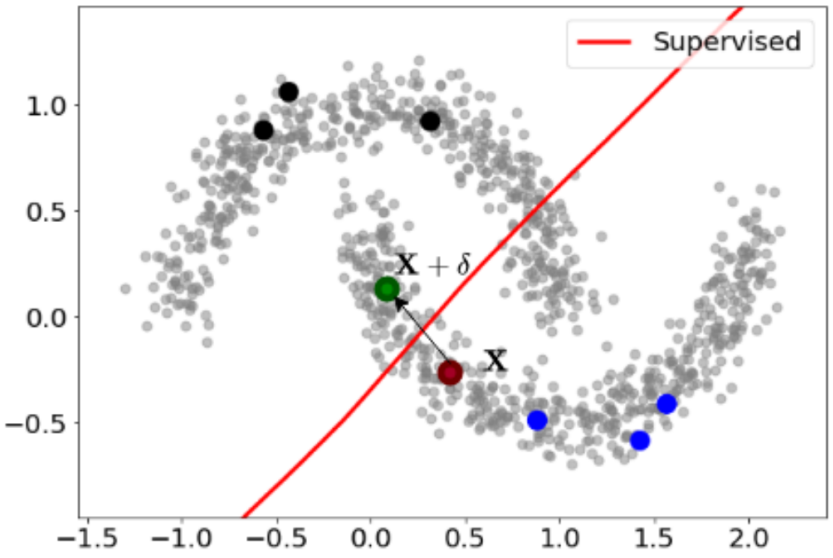


Fig. 4. On the two moons problem, a handful of labeled examples are provided which by themselves suggest the optimal classifier is linear. Consistency regularization on the unlabeled points gives error on perturbed points which changes the decision boundary so that it does not pass through either of the two moons.

2.3.2. Virtual Adversarial Training

The hand-crafted stochastic perturbations used by (Sajjadi et al., 2016; Laine & Aila, 2016), such as i.i.d. Gaussian noise, might be inefficient in high dimensions, as only a tiny fraction of input perturbations will cross the decision boundary. Virtual adversarial training (VAT) seeks to alleviate this problem by finding an adversarial perturbation which causes maximum change to the model's predictions for a given input sample. The Virtual Adversarial Training (VAT) is motivated by Adversarial Training (Goodfellow et al., 2015; Szegedy et al., 2014). An adversarial perturbation r_{adv} in VAT is computed as:

$$r_{adv} = \underset{r,\quad ||r||_2 \leq \varepsilon}{\operatorname{argmax}} D(f_\theta(x), f_\theta(x+r)). \tag{2.17}$$

VAT uses a scalar hyperparameter ε which specifies the upper bound on the norm of the adversarial perturbation. VAT (Miyato et al., 2018a) approximates the computation of r_{adv} as follows based on a single iteration of the power-method:

$$r_{adv} \approx \varepsilon \frac{g}{\|g\|_2} \tag{2.18}$$

$$\text{where } g = \nabla_r D(f_\theta(x), f_\theta(x+r))\Big|_{r=\xi d} \tag{2.19}$$

In this expression $\xi \neq 0$ is a scalar hyperparameter whose value is set to be very small (such as $1e-6$) and d is a randomly sampled unit vector. We cannot set $\xi = 0$ because $D(f_\theta(x), f_\theta(x)) = 0$, $g = 0$, and thus r_{adv} becomes undefined.

Using this approximate adversarial perturbation r_{adv} for a sample x, the consistency regularization in VAT is computed as:

$$R_\theta(x) = D(f_\theta(x), f_\theta(x + r_{adv})) \tag{2.20}$$

2.3.3. Improving Targets with Ensembling or Mean Teacher

Consistency-based semi-supervised learning methods involve encouraging the network's outputs to be unchanged by applying some perturbation. Many of these perturbations can be applied in the input space (such as with gaussian noise or image augmentations) or the hidden states, but another form of perturbation can be considered in the space of the parameters. One way to get these perturbed target networks involves using the network from previous iterations during training. In the temporal ensembling technique (Laine & Aila, 2016), one uses the running average of the model's own predictions as targets. This technique was further refined and simplified in the Mean-Teacher algorithm, in which targets are constructed from a network which is parameterized by an exponential moving average of the model's parameters. Intriguingly, despite the general non-convexity of deep neural networks, a network constructed by averaging parameter values from multiple iterates tends to be more accurate than the network from the most recent iterate (Tarvainen & Valpola, 2017b).

2.4. Neural Architectures

2.4.1. Linear Models

The foundational building block of deep neural networks is a linear model: $\mathbf{y} = \mathbf{W}\mathbf{x}$ where $\mathbf{x} \in \mathbb{R}^{d_{in}}$ and $\mathbf{W} \in \mathbb{R}^{d_{in} \times d_{out}}$, where $\mathbf{W}$ is a matrix of learnable parameters. Intuitively every parameter in a linear model's weight matrix learns how strongly (and in what direction) a single input variable changes a single output variable.

2.4.2. Deep Neural Networks

The simplest deep neural networks consist of multiple alternating layers which each consist of a learnable linear projection followed by a (generally) fixed non-linearity:

$$\mathbf{h}_j^l = f\Big(\sum_{i=1}^{d_{in}^l} \mathbf{W}_{ij}^l \mathbf{h}_i^{l-1} + \mathbf{b}_i^l\Big) \tag{2.21}$$

If we use matrix notation, we can remove the subscripts and simplify this expression to:

$$\mathbf{h}^l = f(\mathbf{W}^l \mathbf{h}^{l-1} + \mathbf{b}^l) \tag{2.22}$$

In the above equation, the non-linearity is referred to as $f()$. Many different non-linearities have been shown to work well, but the ReLU non-linearity is simple and remains in wide usage: $f(a) = \max(a, 0)$.

The necessity of the non-linearity is intriguing. Initially, we observe that any product of multiple weight matrices can be equivalently written as a single weight matrix: $\mathbf{W} = \mathbf{W}_1\mathbf{W}_2...\mathbf{W}_n$. Thus a deep network without the use of a non-linearity is no more expressive than a linear model.

Neural networks with a single hidden layer are universal function approximators, but the width of the hidden layer may need to be very large to handle complex problems. (Htad, 1987) showed limited expressiveness for shallow networks.

A simple problem which highlights the importance of non-linear hidden layers is the *XOR* problem in which there are two binary-valued input variables and a single binary-valued output variables. The dataset consists of four examples, with $(0, 1)$ and $(1, 0)$ having the output of 1 and $(0, 0)$ and $(1, 1)$ having the output of 0. A linear model cannot solve this problem because the relationship between the first input variable and the output changes depending on the value of the second input variable.

A very simple solution for the *XOR* problem using a single hidden layer can be derived by realizing that a linear solution is permitted if we only consider the examples: $(0, 1)$, $(0, 0)$, and $(1, 0)$, which is simply $y = x_1 + x_2$. However this solution fails for $(1, 1)$. However, if we construct a hidden unit which "activates" only on $(1, 1)$, we can use it to fix the partially-correct linear solution. Using the relu activation $\text{relu}(x) = max(x, 0)$, we can construct $h_1 = \text{relu}(2 * x_1 + 2 * x_2 - 3)$, which will take a value of 0 for the examples the linear model already solves, but will take a value of 1 for the input $(1, 1)$. We can then solve *XOR* with: $y = x_1 + x_2 - h_1$.

One generalization of *XOR* to more than two input variables illustrates the advantage of using deep neural networks with more than a single hidden layer. For example, suppose we have a list of N binary variables $x_1, x_2, ..., x_N$ and wish to classify if the number of inputs which are 1 is even ($y = 0$) or odd ($y = 1$). An inductive solution to this problem involves determining if the first j of the digits are even or odd, which we can call y_j. We can then solve for y_{j+1} as $\text{XOR}(y_j, 1 - x_j)$. Since we solved *XOR* previously with a single hidden layer network, this suggests that we can solve this N step parity problem with a network using N layers.

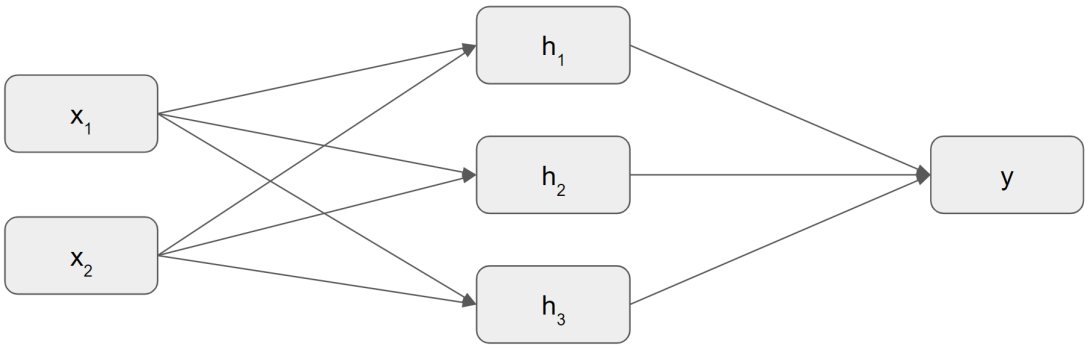


Fig. 5. A multi-layer perceptron with 2 input neurons, 3 hidden neurons, a single hidden layer and 1 output neuron.

In the case of a supervised binary classification problem, the network produces a single scalar output (like shown in Figure 5) which is typically soft-thresholded by a sigmoid function $\sigma(\cdot)$ to produce values in the closed interval $[0, 1]$:

$$\sigma(\mathbf{x}) = \frac{1}{1 + e^{-\mathbf{x}}}. \tag{2.23}$$

To handle the case with multiple classes, the probability of a class j is typically computed using the softmax function using the weights and hidden states of the final layer n in the network. We will use $\mathbf{y}$ to make it clear that it is the output of the network, but it can be thought of as activations of the final layer $\mathbf{h}^n$ as well:

$$y_j = \exp\big(\sum_i \mathbf{W}^n_{ij}\mathbf{h}^{n-1}_i + \mathbf{b}^n_j\big) / \Big(\sum_{j'} \exp\big(\sum_i \mathbf{W}^n_{ij'}\mathbf{h}^{n-1}_i + \mathbf{b}^n_{j'}\big)\Big). \tag{2.24}$$

These basic elements of learned linear weight matrices and elementwise non-linear activation functions serve as the basic foundations for deep neural networks.

2.4.3. Training Neural Networks with Backpropagation

In the simplest case, we want to train a neural network to minimize error on an i.i.d dataset of examples $((\mathbf{x}_1, y_1) \dots (\mathbf{x}_N, y_N))$. When using the negative log-likelihood loss function, ERM corresponds to maximum-likelihood estimation of the parameters of the model. In this section, we will discuss by far the most popular approach to training deep neural networks, the backpropagation algorithm (Rumelhart et al., 1986; Werbos, 1974).

This technique involves computing gradients of the network's loss with respect to the parameters $\frac{\partial \mathcal{L}}{\partial \theta}$, which can then be used to incrementally update θ in the direction which locally reduces the loss:

$$\theta_t = \theta_{t-1} - \varepsilon \frac{\partial \mathcal{L}}{\partial \theta}. \tag{2.25}$$

This gradient descent algorithm is provably justified for convex optimization problems, such as optimizing linear models with mean square error as the loss function. While neural networks are generally non-convex, it has been found empirically that gradient descent can still work well for a wide range of neural network architectures. However, this success depends on the details of the architecture and the initialization scheme and is far from guaranteed. For example, if a neural network is initialized with all parameters set to zero, the gradient is zero, and training stagnates - so it is generally essential to initialize the network with random-valued parameters.

A following question is how gradients can be computed for gradient descent. By far the most popular algorithm for this is backpropagation, which is a special case of reverse-mode automatic differentiation (for a review of automatic differentiation techniques we refer the reader to Margossian (2019)). The backpropagation algorithm is the application of the chain rule of calculus to neural networks. In its first stage, the gradient with respect to an intermediate hidden layer is computed as: $\frac{\partial \mathcal{L}}{\partial h_i} = \frac{\partial \mathcal{L}}{\partial h_{i+1}} \frac{\partial h_{i+1}}{\partial h_i}$, which can further be reduced to multiplying by the transposed weight matrix and multiplying by the point-wise derivative of the activation function. The gradient with respect to the weight matrices (and hence parameters) can then be computed based on the hidden states and the gradients with respect to the hidden states.

2.4.4. Stochastic Gradient Descent

On small datasets, it is often reasonable to compute the gradient across all the examples in the dataset and use this full gradient for each update. However, for large datasets, this is clearly sub-optimal, as the gradient on a small subset of the dataset may be very similar to the gradient on the full dataset. As a result, the number of updates done for a fixed amount

of computation would scale poorly in the size of the dataset. To illustrate this, we could imagine that with our computation budget we are able to do N full gradient descent updates. If we were to modify our dataset by duplicating every example K times, then we would only be able to do $\frac{N}{K}$ updates, which eventually would shrink to be less than 1 (meaning that we'd never do a single update). We can instead use a random subset of the examples to compute an estimate of the gradient (referred to as a *stochastic gradient*). The model takes many stochastic gradient steps to converge to a point estimate of the parameters that achieves low error. The SGD update rule at step t for a parameter $\mathbf{W}^l$, after seeing example $(\mathbf{x}, \mathbf{y})$, can be written as

$$\mathbf{W}_t^l = \mathbf{W}_{t-1}^l - \alpha \frac{\partial \mathcal{L}(\mathbf{x}, \mathbf{y}; \theta_{\mathbf{t-1}})}{\partial \mathbf{W}_{t-1}^l} \tag{2.26}$$

Where we use a learning rate α to control how large of a step to take in the direction of the gradient. α can be fixed over the course of training or changed adaptively as a function of t. Often we want to decay α near the end of training and we also may want α to be small near the beginning of training to improve stability.

2.4.5. Convolutional Neural Networks

We've previously discussed the fully-connected neural network architecture (often called a Multi-Layer Perceptron or MLP), which is very general but has some interesting limitations. Perhaps the most fundamental one is that there is no structure to the hidden state to associate different hidden units with different positions. Convolutional neural networks address this by replacing a single monolithic hidden state $h \in \mathbb{R}^d$ by a structured hidden state $h \in \mathbb{R}^{d \times h \times w}$ where h and w are the height and width of the structured hidden state respectively. The key idea of convolutional networks is that the weights also have a spatial structure $\mathbf{W} \in \mathbb{R}^{k \times c \times \Delta x \times \Delta y}$ where k is the number of output units per position and c is the number of input units per position. These weight tensors are then convolved with the input tensor to produce a structured linear operation. An illustration of the receptive field of a convolutional neural network is shown in Figure 6. This linear operation can then be followed by elementwise non-linearities (such as the ReLU function) as well as downsampling or upsampling operations to change the spatial structure of the hidden state. We refer readers to Dumoulin & Visin (2016) for a more detailed discussion on convolutional layers.

2.4.6. Recurrent Neural Networks

Another form of structure that can be added to deep neural networks involves applying the same function to a hidden state multiple times. Such an architecture is called a *Recurrent Neural Network* and the most basic variant can be written as: $\mathbf{h}_t = F(\mathbf{h}_{t-1}, x, \theta)$ where F is our function parameterized by θ (Rumelhart et al., 1986). This kind of repeated computation could be significantly more parameter-efficient and could be useful in problems which require multiple functionally similar processing steps.

We can generalize this further to consider an architecture in which a sequence of inputs $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ is transformed it into a sequence of vector representations $\mathbf{h} = (\mathbf{h}_1, \mathbf{h}_2, \ldots, \mathbf{h}_n)$. We use subscripts to denote the position of an element within the sequence. This more general form of recurrent neural network can be written as: $\mathbf{h}_t = F(\mathbf{h}_{t-1}, \mathbf{x}_t, \theta)$ (see Figure 7). In this setup, each step in the repeated computation is provided with a single position in

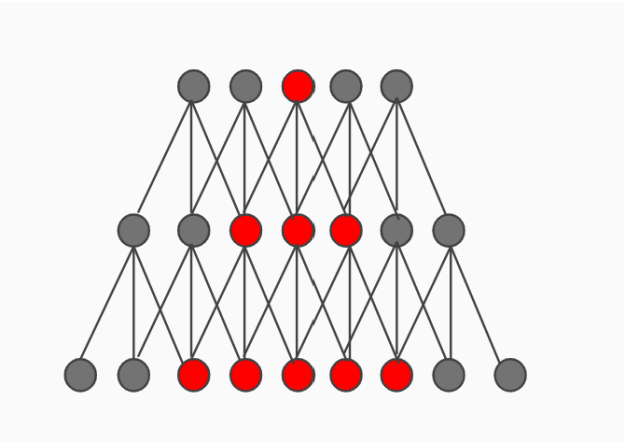


Fig. 6. An illustration of the receptive field of a convnet.

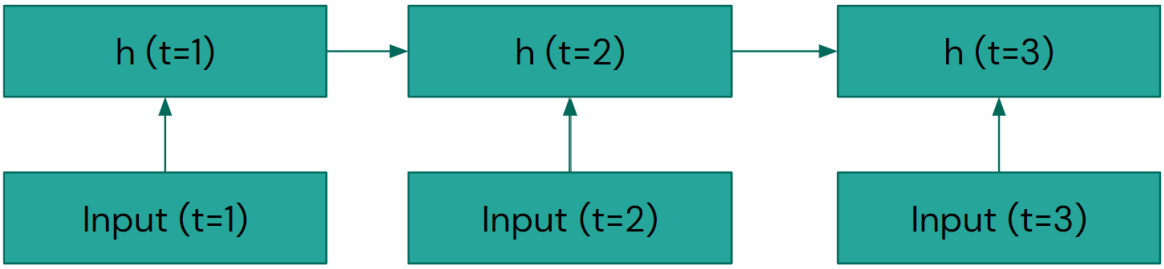


Fig. 7. A simple example of a recurrent neural network with a single hidden layer and a sequence length of three.

an input sequence. This is a widely used architecture for processing time series or other sequences such as text and audio.

A simple choice for the recurrent function F is to use a learned linear function of the input and the hidden state followed by an elementwise non-linearity. We can then write this as: $\mathbf{h}_t = \tanh(\mathbf{W}\mathbf{h}_{t-1} + \mathbf{U}\mathbf{x}_t + \mathbf{b})$. This is parameterized by a recurrent weight matrix $\mathbf{W}$ and input weight matrix $\mathbf{U}$ and a bias term b that are shared across time. If the inputs $\mathbf{x}_i \in \mathbb{R}^{d_{in}}$ are continuous vector valued inputs, then $\mathbf{U} \in \mathbb{R}^{d_{in} \times d_{hid}}$ and $\mathbf{W} \in \mathbb{R}^{d_{hid} \times d_{hid}}$ ($\mathbf{W}$ doesn't necessarily need to be a square matrix, but for the sake of notational convenience, we will assume it is). The hidden state of the recurrent network may then be processed by a linear layer or an MLP to produce an output from the network, such as: $y_t = \mathbf{W}_{o2} \tanh(\mathbf{W}_{o1}\mathbf{h}_t)$

Recurrent networks can be trained using backpropagation, in much the same way as feedforward networks like MLPs are trained. These RNNs can be unrolled in time (Figure 7) into a deep feedforward model with shared weights and the same principles used to compute gradients in Section 2.4.3 can be used while adding gradients across multiple positions to account for weight sharing. This algorithm is commonly referred to as Backpropagation Through Time (BPTT) (Rumelhart et al., 1986; Werbos, 1990) due to its original usage for time-series, but the concept applies to any type of sequential data.

While exact gradients can be computed via backpropagation through time, the algorithm has a significant challenge in practice. If we backpropagate gradients for T steps, then in

the absence of an activation function (the linear-RNN case), the backward gradient contains a term W^T, which has the norm: $||W||^T$. If the norm of W is larger than 1, the resulting gradient tends to be very large for even moderately long time horizons. This is referred to colloquially as *Exploding Gradients*. If the norm of $||W||$ is smaller than 1, the resulting gradient will tend to be small, which is referred to as *Vanishing Gradients* (Hochreiter, 1991; Bengio et al., 1994; Pascanu et al., 2013). To illustrate this problem more concretely, if we consider 200 time steps, then if the weight norm is 1.05, the resulting $||W||^T$ is about 15000. If the weight norm is 0.95, the resulting $||W||^T$ is about 10^{-5}. So even if the weight norm is only slightly deviated from 1, the resulting gradients are very badly scaled. Adding an activation function which has very small or zero gradients at many points can help address the exploding gradient problem at the expense of making the vanishing gradient problem more severe.

The simplest solution to this is *Truncated Backpropagation through Time* which simply stops computing gradients after a certain number of steps K (even if we compute the hidden state of the recurrent neural network for more than K steps). This clearly solves the gradient scaling issue yet also means that the model fails to systematically learn *Long Range Dependencies* (Bengio et al., 1994; Pascanu et al., 2013).

A simple way of reducing the impact of *exploding gradients* is gradient clipping, where the magnitude of the gradients with respect to every parameter is clipped to a maximum value (Pascanu et al., 2012). This is a poor solution in principle since it changes the expected value of the gradients. However, if the gradients only occasionally become very large, this gradient clipping strategy can prevent these very large gradients from ruining the progress of the training algorithm.

A highly principled approach to addressing both vanishing and exploding gradients relies on the concept of unitary matrices from linear algebra. A unitary matrix is any matrix which has a norm $||W|| = 1$ (in the case where W is real-valued, it is called an orthogonal matrix). A very natural way to address the vanishing and exploding gradients problem is to make W such a unitary matrix. This was shown to be trainable on long-sequences with well-scaled gradients (Arjovsky et al., 2015), but requires a fairly complicated parameterization to make W exactly unitary by construction.

A more practical and more widely used solution for addressing vanishing and exploding gradients is to construct a recurrent neural network in which information is added into the hidden state on each recurrent time step after passing through a *Gating Function*. Gated recurrent networks were first presented in the form of the Long Short-Term Memory (LSTM) architecture (Hochreiter & Schmidhuber, 1997a), which was simplified to the Gated-Recurrent Unit (GRU) architecture (Cho et al., 2014).

The GRU introduces two gates $z_t = \sigma(W_z x_t + U_z h_{t-1})$ and $r_t = \sigma(W_r x_t + U_r h_{t-1})$, which both output a score from 0 to 1 for each unit. A pre-gated update value for the recurrent state is computed as: $\tilde{h}_t = \tanh(W x_t + U(r_t \odot h_t))$. The update for the recurrent state is: $h_t = (1 - z_t)h_{t-1} + z_t \tilde{h}_t$. Intuitively, z_t controls whether each unit keeps its value from the previous time-step or takes an updated value. r_t controls which units are used to compute that updated value for the recurrent hidden state.

We can gain more insights into the training dynamics of the GRU when z_t and r_t are saturated. If we always have $z_t = 1$ and $r_t = 1$, then the GRU becomes identical to a vanilla RNN, and it has the exact same training dynamics and resulting instabilities. On the other hand, when $z_t = 0$, then $h_t = h_{t-1}$ and the gradient is passed backward without modification. When $r_t = 0$, we can make updates to the hidden state but only using the input and an

additive dependence on the previous hidden state (weighted by z_t). For example if $z_t = 0.5$ and $r_t = 0$, the GRU behaves as an additive integrator of a non-linear function of the input value: $\tanh(Wx_t)$.

The resulting parameter gradients from a GRU network can still be an exponential of $||W||$, but it is no longer simply $||W||^T$. Rather, it depends on the value of z_t and r_t, with lower values for these gates making the gradient less sensitive to the value of $||W||$. Intuitively, if a unit is only relevant on a small number of time steps and the computation between units is well-separated (the latter is unlikely to exactly hold in practice), then the gradient for that unit may effectively only go through a few applications of the recurrent weight matrix. For this reason, these gated recurrent networks tend to be fairly stable to train on sequences of substantial length (thousands of time steps), although if all the gates are saturated to 1 then it can still suffer from vanishing or exploding gradients, so gradient clipping can help if this saturated condition occurs infrequently.

2.4.7. Attention

Attention (as it is most widely used in deep learning) was introduced for the purpose of improving the capture of long-range dependencies and reducing the information bottleneck in sequence models (Bahdanau et al., 2014). The essential idea is to learn an input-dependent weighting to directly control how information is shared between pairs of positions. This can improve long-range dependencies by allowing information to directly flow from a distant position, rather than being preserved across many steps by a gated recurrent network. It can reduce the bottleneck by allowing a position to depend on many other positions, rather than the single most recent recurrent hidden state.

The most general interface for attention considers three matrices (each consisting of multiple positions each with its own representation vector) as input: queries $Q \in \mathrm{R}^{N_q \times d_q}$, keys $K \in \mathrm{R}^{N_k \times d_q}$, and values $V \in \mathrm{R}^{N_k \times d_v}$. The basic concept of attention (Bahdanau et al., 2014) is that by taking the dot product of all combinations of position-vectors in Q and K will yield a matrix of affinities $\alpha \in \mathrm{R}^{N_q \times N_k}$ between the queries and keys. This is then normalized such that every querying position $1, 2, ..., N_q$ will have an affinity over the keys which sums to 1, which is accomplished using a softmax function. This normalized affinity score is then multiplied by V (effectively weighting over positions in V) to yield an output matrix $A \in \mathrm{N_q} \times \mathrm{d_v}$.

Attention can be used to complement a gated recurrent network (Bahdanau et al., 2014), by allowing information to flow between distant positions. (Vaswani et al., 2017) found that attention can be used in the absence of any recurrent architecture in the *Transformer*. This has a significant scaling advantage since it removes the sequential processing between positions required for recurrent networks. At the same time, a single layer of attention has much weaker processing capabilities compared to a single recurrent layer, so Transformers typically require many attention layers to achieve good performance. Additionally, the attention operation is permutation invariant, so ordering information needs to be provided through position encoding (either sinusoids with different periods or separate learned parameters per position).

Chapter 3

Prologue to the first article

3.1. Article Details

Manifold Mixup: Better representations by interpolating hidden states. Alex Lamb*, Vikas Verma*, Chris Beckham, Amir Najafy, Ioannis Mitliagkas, Aaron Courville, Yoshua Bengio.

Manifold Mixup is an algorithm for improving the robustness and generalization performance of deep neural networks. It involves interpolating hidden states between different examples in deep neural networks while also interpolating the losses of these examples. Additionally, the layer to apply interpolation to is randomly selected on each training iteration from a set of eligible layers. A direct goal of manifold mixup is to encourage the model to perform well on combinations of semantically meaningful attributes not observed during training. Improving the smoothness and increasing the margin of the classifier are further motivations. Another advantage of the Manifold Mixup technique that we did not anticipate before starting the project, is that Manifold Mixup encourages the hidden representations to lie on a lower-dimensional linear subspace. In practice, the hidden representations become much more compressed in networks trained with manifold mixup.

3.2. Personal Contribution

This project began when Alex Lamb, Vikas Verma, and Aaron Courville began discussing the idea. In the initial idea, I was interested in using a mixup-inspired regularization for adversarially learned inference (Dumoulin et al., 2016a). However, as we began experimenting and found positive results, we realized that the impact could be broader if we applied the technique to the more general supervised learning case. I wrote some of the initial code and wrote much of the paper, and played a major role in the initial brainstorming. I also had the initial idea for the "flattening hidden states" theory, although the actual specific theory and math in the paper was developed by Amir Najafy. Chris Beckham designed and ran the GAN experiments, where Manifold Mixup was applied to the discriminator of a GAN (Goodfellow et al., 2014) network. Ioannis Mitliagkas, Aaron Courville, and Yoshua Bengio all played important advising roles. I asked Vikas Verma and received permission to use this paper in my thesis.

3.3. Context

The goal of this work was to develop a way of encouraging deep networks to train with distinct combinations of semantic attributes, and thus reduce the chance of evaluating on an example with an unseen combination during training. We used the technique of interpolating target values that was used in the original mixup paper (Zhang et al., 2018c). Some further inspiration comes from the techniques which apply regularization to the hidden representations learned by deep neural networks. For example, state-reification networks (Lamb et al., 2019) showed that autoencoders trained in the hidden representations of deep networks can improve adversarial robustness in both convnets and RNNs.

3.4. Contributions

The Manifold Mixup paper showed that it's possible to train with interpolated hidden states from multiple examples and that this can often outperform mixup in the input space. We also showed that manifold mixup has a unique effect on the hidden states: pushing them onto a lower dimensional linear subspace. The Manifold Mixup technique significantly improves classification accuracy and calibration despite not requiring significant additional computation. Improved classification results are reported on CIFAR-10, CIFAR-100, and SVHN. An interesting aspect of the results is that the negative log-likelihood on the test set shows a much larger improvement than the test accuracy, suggesting that the manifold mixup classifier is less confident on examples where it makes incorrect predictions.

3.5. Research Impact

Manifold Mixup has seen excellent adoption in the applied research community, with many successful follow-up papers in speech, vision, language, and medical imaging. Manifold Mixup was applied to achieve state-of-the-art results in the few-shot learning domain (Mangla et al., 2019). Fair Mixup (Chuang & Mroueh, 2021) found that Manifold Mixup achieves competitive results on fairness benchmarks focusing on spurious correlations in face-attribute classification. In particular, Manifold Mixup often outperforms techniques specifically designed to improve fairness. (Moysset & Messina, 2019) showed that Manifold Mixup substantially improves generalization performance in handwriting recognition. Manifold Mixup has also been used to significantly improve robustness in medical polyp segmentation, where the size of training datasets tends to be very small (Guo et al., 2021). The theory developed in the work has also seen a few intriguing follow-ups, for example, (Roth et al., 2020) showed that the compression effect of manifold mixup is harmful for metric learning. There have also been a few exciting follow-ups on the technique. PatchUp (Faramarzi et al., 2020) explored a variation on manifold mixup where the mixing is done over randomly selected patches, rather than being performed as linear interpolation over the entire hidden state. An extension on graphs (Verma et al., 2019b) explored linearly interpolating the hidden states of nodes in graph neural networks. Adversarial Mixup Resynthesizers (Beckham et al., 2019) showed that when latent-space interpolation is used in training adversarial autoencoders, the visual quality of latent-interpolations can be greatly improved. In that case, an adversarially trained discriminator (Goodfellow et al., 2014) was used to encourage the decoded reconstructions from these interpolated latent states to appear more similar to real examples.

Chapter 4

Manifold Mixup

4.1. Introduction

Deep neural networks are the backbone of state-of-the-art systems for computer vision, speech recognition, and language translation (LeCun et al., 2015). However, these systems perform well only when evaluated on instances very similar to those from the training set. When evaluated on slightly different distributions, neural networks often provide incorrect predictions with strikingly high confidence. This is a worrying prospect, since deep learning systems are being deployed in settings where data may be subject to distributional shifts. Adversarial examples (Szegedy et al., 2014) are one such failure case: deep neural networks with nearly perfect performance provide incorrect predictions with very high confidence when evaluated on perturbations imperceptible to the human eye. Adversarial examples are a serious hazard when deploying machine learning systems in security-sensitive applications. More generally, deep learning systems quickly degrade in performance as the distributions of training and testing data differ slightly from each other (Ben-David et al., 2010).

In this paper, we realize several troubling properties concerning the hidden representations and decision boundaries of state-of-the-art neural networks. First, we observe that the decision boundary is often sharp and close to the data. Second, we observe that the vast majority of the hidden representation space corresponds to high confidence predictions, both on and off of the data manifold.

Motivated by these intuitions we propose *Manifold Mixup* (Section 4.2), a simple regularizer that addresses several of these flaws by training neural networks on linear combinations of hidden representations of training examples. Previous work, including the study of analogies through word embeddings (e.g. king $-$ man $+$ woman $\approx$ queen), has shown that interpolations are an effective way of combining factors (Mikolov et al., 2013). Since high-level representations are often low-dimensional and useful to linear classifiers, linear interpolations of hidden representations should explore meaningful regions of the feature space effectively. To use combinations of hidden representations of data as novel training signal, we also perform the same linear interpolation in the associated pair of one-hot labels, leading to mixed examples with soft targets.

To start off with the right intuitions, Figure 1 illustrates the impact of *Manifold Mixup* on a simple two-dimensional classification task with small data. In this example, vanilla training of a deep neural network leads to an irregular decision boundary (Figure 1a), and a complex arrangement of hidden representations (Figure 1d). Moreover, every point in both the raw (Figure 1a) and hidden (Figure 1d) data representations is assigned a prediction with very

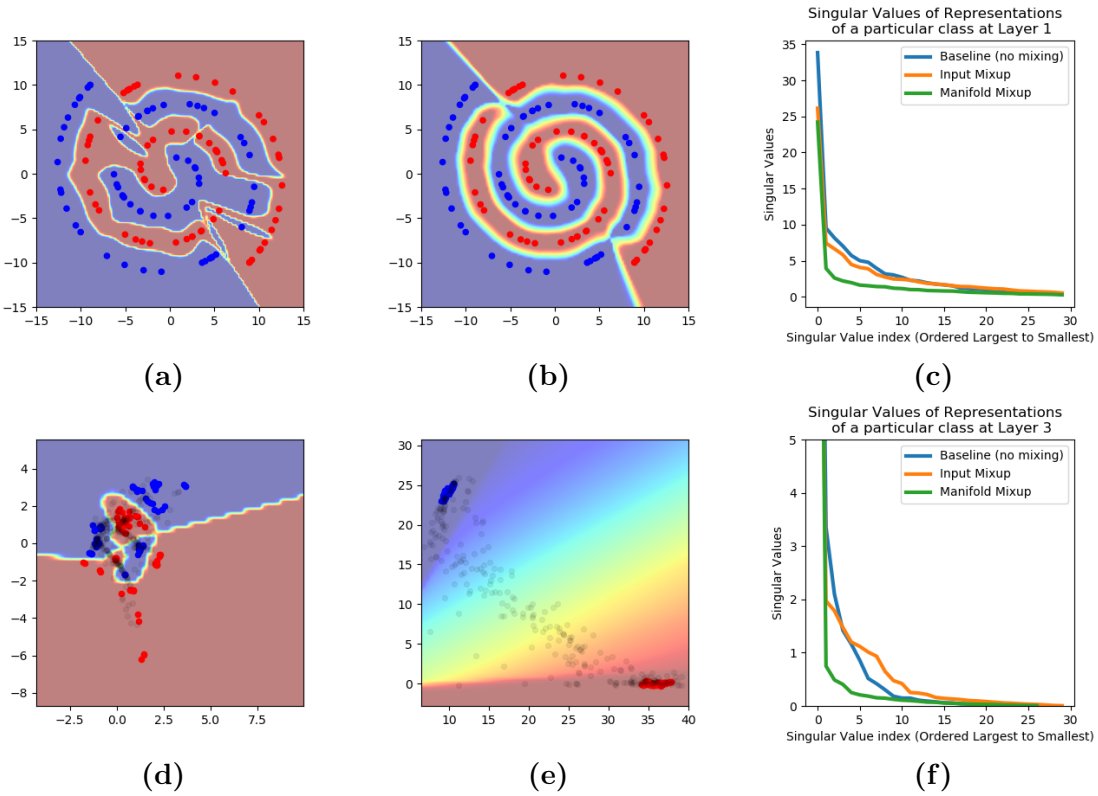


Fig. 1. An experiment on a network trained on the 2D spiral dataset with a 2D bottleneck hidden representation in the middle of the network. Manifold mixup has three effects on learning when compared to vanilla training. First, it smoothens decision boundaries (from a. to b.). Second, it improves the arrangement of hidden representations and encourages broader regions of low-confidence predictions (from d. to e.). Black dots are the hidden representation of the inputs sampled uniformly from the range of the input space. Third, it flattens the representations (c. at layer 1, f. at layer 3) of all the examples belonging to a particular class. Figure 2 shows that these effects are not accomplished by other well-studied regularizers (input mixup, weight decay, dropout, batch normalization, and adding noise to the hidden representations).

high confidence. This includes points (depicted in black) that correspond to inputs off the data manifold! In contrast, training the same deep neural network with *Manifold Mixup* leads to a smoother decision boundary (Figure 1b) and a simpler (linear) arrangement of hidden representations (Figure 1e). In sum, the representations obtained by *Manifold Mixup* have two desirable properties: the class-representations are flattened into a minimal amount of directions of variation, and all points in-between these flat representations, most unobserved during training and off the data manifold, are assigned low-confidence predictions.

This example conveys the central message of this paper:

Manifold Mixup improves the hidden representations and decision boundaries of neural networks at multiple layers.

More specifically, *Manifold Mixup* improves generalization in deep neural networks because it:

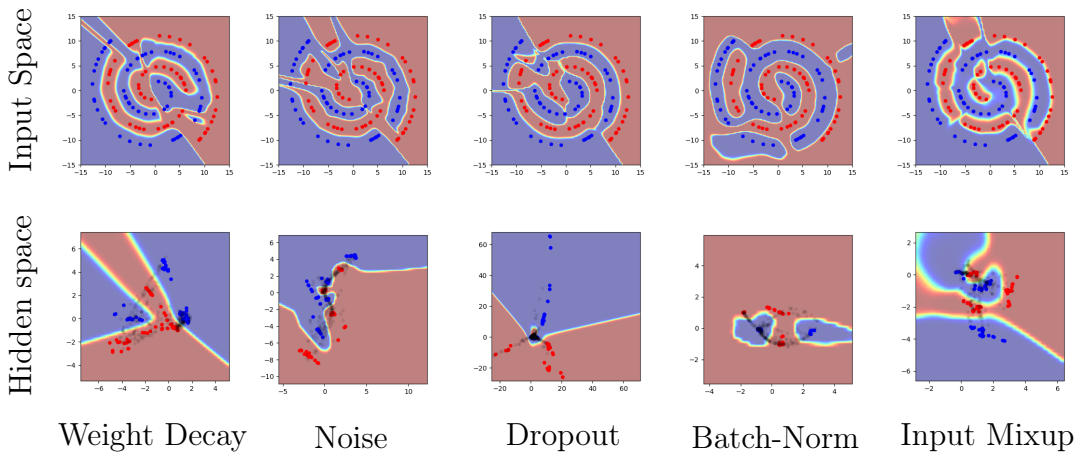


Fig. 2. The same experimental setup as Figure 1, but using a variety of competitive regularizers. This shows that the effect of concentrating the hidden representation for each class and providing a broad region of low confidence between the regions is not accomplished by the other regularizers (although input space mixup does produce regions of low confidence, it does not flatten the class-specific state distribution). Noise refers to gaussian noise in the input layer, dropout refers to dropout of 50% in all layers except the bottleneck itself (due to its low dimensionality), and batch normalization refers to batch normalization in all layers.

- Leads to smoother decision boundaries that are further away from the training data, at multiple levels of representation. Smoothness and margin are well-established factors of generalization (Bartlett & Shawe-taylor, 1998; Lee et al., 1995).
- Leverages interpolations in deeper hidden layers, which capture higher level information (Zeiler & Fergus, 2013) to provide additional training signal.
- Flattens the class-representations, reducing their number of directions with significant variance (Section 4.3). This can be seen as a form of compression, which is linked to generalization by a well-established theory (Tishby & Zaslavsky, 2015; Shwartz-Ziv & Tishby, 2017) and extensive experimentation (Alemi et al., 2017; Belghazi et al., 2018; Goyal et al., 2018; Achille & Soatto, 2018).

Throughout a variety of experiments, we demonstrate four benefits of *Manifold Mixup*:

- Better generalization than other competitive regularizers (such as Cutout, Mixup, AdaMix, and Dropout) (Section 4.5.1).
- Improved log-likelihood on test samples (Section 4.5.1).
- Increased performance at predicting data subject to novel deformations (Section 4.5.2).
- Improved robustness to single-step adversarial attacks. This is the evidence that *Manifold Mixup* pushes the decision boundary away from the data in some directions (Section 4.5.3). This is not to be confused with full adversarial robustness, which is defined in terms of moving the decision boundary away from the data in *all* directions.

4.2. Manifold Mixup

Consider training a deep neural network $f(x) = f_k(g_k(x))$, where g_k denotes the part of the neural network mapping the input data to the hidden representation at layer k, and f_k denotes the part mapping such hidden representation to the output $f(x)$. Training f using

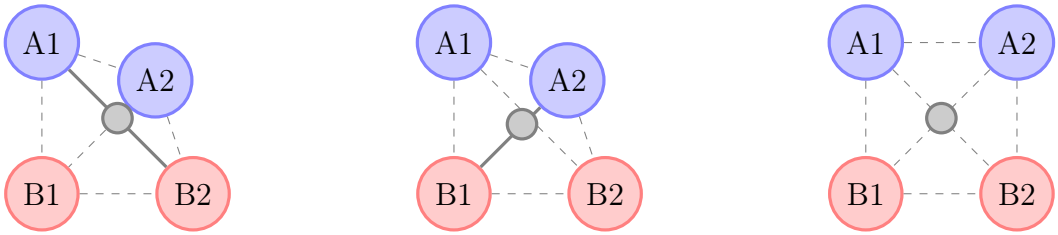


Fig. 3. Illustration on why Manifold Mixup learns flatter representations. The interpolation between A1 and B2 in the left panel soft-labels the black dot as 50% red and 50% blue, regardless of being very close to a blue point. In the middle panel a different interpolation between A2 and B1 soft-labels the same point as 95% blue and 5% red. However, since *Manifold Mixup learns* the hidden representations, the pressure to predict consistent soft-labels at interpolated points causes the states to become flattened (right panel). The effect of this flattening is that all points from the blue class would be forced onto a line, reducing the number of directions of variability from two to one (and thus making the representation flatter).

Manifold Mixup is performed in five steps. First, we select a random layer k from a set of eligible layers $\mathcal{S}$ in the neural network. This set may include the input layer $g_0(x)$. Second, we process two random data minibatches (x, y) and (x', y') as usual, until reaching layer k. This provides us with two intermediate minibatches $(g_k(x), y)$ and $(g_k(x'), y')$. Third, we perform Input Mixup (Zhang et al., 2018c) on these intermediate minibatches. This produces the mixed minibatch:

$$(\tilde{g}_k, \tilde{y}) := (\text{Mix}_\lambda(g_k(x), g_k(x')), \text{Mix}_\lambda(y, y')),$$

where $\text{Mix}_\lambda(a, b) = \lambda \cdot a + (1 - \lambda) \cdot b$. Here, (y, y') are one-hot labels, and the mixing coefficient $\lambda \sim \text{Beta}(\alpha, \alpha)$ as proposed in mixup (Zhang et al., 2018c). For instance, $\alpha = 1.0$ is equivalent to sampling $\lambda \sim U(0, 1)$. Fourth, we continue the forward pass in the network from layer k until the output using the mixed minibatch $(\tilde{g}_k, \tilde{y})$. Fifth, this output is used to compute the loss value and gradients that update all the parameters of the neural network.

Mathematically, *Manifold Mixup* minimizes:

$$L(f) = \mathop{\mathbb{E}}_{(x,y)\sim P} \mathop{\mathbb{E}}_{(x',y')\sim P} \mathop{\mathbb{E}}_{\lambda\sim\text{Beta}(\alpha,\alpha)} \mathop{\mathbb{E}}_{k\sim\mathcal{S}} \ell(f_k(\text{Mix}_\lambda(g_k(x), g_k(x'))), \text{Mix}_\lambda(y, y')). \tag{4.1}$$

Some implementation considerations. We backpropagate gradients through the entire computational graph, including those layers before the mixup layer k (Section 4.5.1 explore this issue in more detail). In the case where $\mathcal{S} = \{0\}$, *Manifold Mixup* reduces to the original mixup algorithm of Zhang et al. (2018c).

While one could try to reduce the variance of the gradient updates by sampling a random (k, λ) per example, we opted for the simpler alternative of sampling a single (k, λ) per minibatch, which in practice gives the same performance. As in Input Mixup, we use a single minibatch to compute the mixed minibatch. We do so by mixing the minibatch with copy of itself with shuffled rows.

A significant limitation of Mixup (Zhang et al., 2018c) is that it makes strong assumptions about the data - in particular that interpolations must be consistent with the data distribution to prevent underfitting. For example, fitting mixup on a circle or 2d spiral in a low-dimensional space could lead to significant underfitting. *Manifold Mixup* relaxes this assumption by interpolating in the hidden states, which can be learned to ensure that these interpolations

are consistent. The implications of forcing this consistency is the focus of the following section.

4.3. Manifold Mixup Flattens Representations

We turn to the study of how *Manifold Mixup* impacts the hidden representations of a deep neural network. At a high level, *Manifold Mixup* jointly flattens the representations of all examples belonging to a specific class. This flattening effect happens on a per-class basis, and we refer to the representations of all examples belong to a particular class as *class-specific representations*. More specifically, this flattening reduces the number of directions with significant variance (akin to reducing their number of principal components).

In the sequel, we first prove a theory (Section 4.3.1) that characterizes this behavior precisely under idealized conditions. Second, we show that this flattening also happens in practice, by performing the SVD of class-specific representations of neural networks trained on real datasets (Section 4.3.2). Finally, we discuss why the flattening of class-specific representations is a desirable property (Section 4.3.3).

4.3.1. Theory

We start by characterizing how the representations of a neural network are changed by *Manifold Mixup*, under a simplifying set of assumptions. More concretely, we will show that if one performs mixup in a sufficiently deep hidden layer in a neural network, then the loss can be driven to zero if the dimensionality of that hidden layer $\dim(\mathcal{H})$ is greater than the number of classes d. As a consequence of this, the resulting representations for that class will have $\dim(\mathcal{H}) - d + 1$ dimensions.

A more intuitive and less formal version of this argument is shown in Figure 3. Intuitively, we show a sufficient condition for the mixup objective to achieve zero loss (given zero loss is also achievable on the original dataset). As a starting point, we observe that a linear function achieves this since $w(\lambda x_1 + (1-\lambda)x_2) = \lambda w x_1 + (1-\lambda) w x_2$. We can then study the linear algebra of this linear function and the representations being passed into this linear function, and then study the necessary rank of the representation matrix.

To this end, assume that $\mathcal{X}$ and $\mathcal{H}$ denote the input and representation spaces, respectively. We denote the label-set by $\mathcal{Y}$ and let $\mathcal{Z} = \mathcal{X} \times \mathcal{Y}$. Let $\mathcal{G} \subseteq \mathcal{H}^{\mathcal{X}}$ denote the set of functions realizable by the neural network, from the input to the representation. Similarly, let $\mathcal{F} \subseteq \mathcal{Y}^{\mathcal{H}}$ be the set of all functions realizable by the neural network, from the representation to the output.

We are interested in the solution of the following problem in some asymptotic regimes:

$$J(P) = \inf_{g \in \mathcal{G}, f \in \mathcal{F}} \mathop{\mathbb{E}}_{(x,y),(x',y'),\lambda} \ell(f(\text{Mix}_\lambda(g(x), g(x'))), \text{Mix}_\lambda(y, y')). \tag{4.2}$$

More specifically, let P_D be the empirical distribution defined by a dataset $D = \{(x_i, y_i)\}_{i=1}^n$. Then, let $f^\star \in \mathcal{F}$ and $g^\star \in \mathcal{G}$ be the minimizers of (4.2) for $P = P_D$. Also, let $\mathcal{G} = \mathcal{H}^{\mathcal{X}}$, $\mathcal{F} = \mathcal{Y}^{\mathcal{H}}$, and $\mathcal{H}$ be a vector space. These conditions (Cybenko, 1989) state that the mappings realizable by large neural networks are dense in the set of all continuous bounded functions. In this case, we show that the minimizer $f^\star$ is a linear function from $\mathcal{H}$ to $\mathcal{Y}$. In this case,

the objective (4.2) can be rewritten as:

$$J(P_D) = \inf_{h_1,\ldots,h_n \in \mathcal{H}} \frac{1}{n\,(n-1)} \sum_{i \neq j}^{n} \left\{ \inf_{f \in \mathcal{F}} \int_0^1 \ell(f(\mathrm{Mix}_\lambda(h_i, h_j)), \mathrm{Mix}_\lambda(y_i, y_j))\, p(\lambda) \mathrm{d}\lambda \right\},$$

where $h_i = g(x_i)$. Let $\mathcal{H}$ be a vector space of dimension $\dim(\mathcal{H})$, and let $d \in \mathbb{N}$ to represent the number of classes contained in some dataset D. If $\dim(\mathcal{H}) \geq d-1$, then $J(P_D) = 0$ and the corresponding minimizer $f^\star$ is a linear function from $\mathcal{H}$ to $\mathbb{R}^d$.

Proof. First, we observe that the following statement is true if $\dim(\mathcal{H}) \geq d-1$:

$$\exists\ A, H \in \mathbb{R}^{\dim(\mathcal{H}) \times d}, b \in \mathbb{R}^d : A^\top H + b 1_d^\top = I_{d \times d},$$

where $I_{d\times d}$ and 1_d denote the d-dimensional identity matrix and all-one vector, respectively. In fact, $b1_d^\top$ is a rank-one matrix, while the rank of identity matrix is d. So, $A^\top H$ only needs rank $d-1$.

Let $f^\star(h) = A^\top h + b$ for all $h \in \mathcal{H}$. Let $g^\star(x_i) = H_{\zeta_i,:}$ be the ζ_i-th column of H, where $\zeta_i \in \{1, \ldots, d\}$ stands for the class-index of the example x_i. These choices minimize (4.2), since:

$$\ell(f^\star(\mathrm{Mix}_\lambda(g^\star(x_i), g^\star(x_j))), \mathrm{Mix}_\lambda(y_i, y_j)) =$$
$$\ell(A^\top \mathrm{Mix}_\lambda(H_{\zeta_i,:}, H_{\zeta_j,:}) + b, \mathrm{Mix}_\lambda(y_{i,\zeta_i}, y_{j,\zeta_j})) = \ell(u, u) = 0.$$

The result follows from $A^\top H_{\zeta_i,:} + b = y_{i,\zeta_i}$ for all i. □

Furthermore, if $\dim(\mathcal{H}) > d-1$, then data points in the representation space $\mathcal{H}$ have some degrees of freedom to move independently.
Consider the setting in Theorem 4.3.1 with $\dim(\mathcal{H}) > d-1$. Let $g^\star \in \mathcal{G}$ minimize (4.2) under $P = P_D$. Then, the representations of the training points $g^\star(x_i)$ fall on a $(\dim(\mathcal{H}) - d + 1)$-dimensional subspace.

Proof. From the proof of Theorem 4.3.1, $A^\top H = I_{d\times d} - b1_d^\top$. The r.h.s. of this expression is a rank-$(d-1)$ matrix for a properly chosen b. Thus, A can have a null-space of dimension $\dim(\mathcal{H}) - d + 1$. This way, one can assign $g^\star(x_i) = H_{\zeta_i,:} + e_i$, where $H_{\zeta_i,:}$ is defined as in the proof of Theorem 4.3.1, and e_i are arbitrary vectors in the null-space of A, for all $i = 1, \ldots, n$. □

This result implies that if the *Manifold Mixup* loss is minimized, then the representation of each class lies on a subspace of dimension $\dim(\mathcal{H}) - d + 1$. In the extreme case where $\dim(\mathcal{H}) = d-1$, each class representation will collapse to a single point, meaning that hidden representations would not change in any direction, for each class-conditional manifold. In the more general case with larger $\dim(\mathcal{H})$, the majority of directions in $\mathcal{H}$-space will be empty in the class-conditional manifold.

4.3.2. Empirical Investigation of Flattening

We now show that the "flattening" theory that we have just developed also holds for real neural networks networks trained on real data. To this end, we trained a collection of fully-connected neural networks on the MNIST dataset using multiple regularizers, including *Manifold Mixup*. When using *Manifold Mixup*, we mixed representations at a single, fixed hidden layer per network. After training, we performed the Singular Value Decomposition (SVD) of the hidden representations of each network, and analyzed their spectrum decay.

More specifically, we computed the largest singular value per class, as well as the sum of the all other singular values. We computed these statistics at the first hidden layer for all networks and regularizers. For the largest singular value, we obtained: 51.73 (baseline), 33.76 (weight decay), 28.83 (dropout), 33.46 (input mixup), and 31.65 (manifold mixup). For the sum of all the other singular values, we obtained: 78.67 (baseline), 73.36 (weight decay), 77.47 (dropout), 66.89 (input mixup), and 40.98 (manifold mixup). Therefore, weight decay, dropout, and input mixup all reduce the largest singular value, but only *Manifold Mixup* achieves a reduction of the sum of the all other singular values (e.g. flattening).

4.3.3. Why is Flattening Representations Desirable?

We have presented evidence to conclude that *Manifold Mixup* leads to flatter class-specific representations, and that such flattening is not accomplished by other regularizers.

But why is this flattening desirable? First, it means that the hidden representations computed from our data occupy a much smaller volume. Thus, a randomly sampled hidden representation within the convex hull spanned by the data in this space is more likely to have a classification score with lower confidence (higher entropy). Second, compression has been linked to generalization in the information theory literature (Tishby & Zaslavsky, 2015; Shwartz-Ziv & Tishby, 2017). Third compression has been been linked to generalization empirically in the past by work which minimizes mutual information between the features and the inputs as a regularizer (Belghazi et al., 2018; Alemi et al., 2017; Achille & Soatto, 2018).

4.4. Related Work

Regularization is a major area of research in machine learning. *Manifold Mixup* is a generalization of Input Mixup, the idea of building random interpolations between training examples and perform the same interpolation for their labels (Zhang et al., 2018c; Tokozume et al., 2018b).

Intriguingly, our experiments show that *Manifold Mixup* changes the representations associated to the layers before and after the mixing operation, and that this effect is crucial to achieve good results (Section 4.5.1). This suggests that *Manifold Mixup* works differently than Input Mixup.

Another line of research closely related to *Manifold Mixup* involves regularizing deep networks by perturbing their hidden representations. These methods include dropout (Hinton et al., 2012), batch normalization (Ioffe & Szegedy, 2015b), and the information bottleneck (Alemi et al., 2017). Notably, (Hinton et al., 2012) and (Ioffe & Szegedy, 2015b) demonstrated that regularizers that work well in the input space can also be applied to the hidden layers of a deep network, often to further improve results. We believe that *Manifold Mixup* is a complimentary form of regularization.

Zhao & Cho (2018) explored improving adversarial robustness by classifying points using a function of the nearest neighbors in a fixed feature space. This involves applying mixup between each set of nearest neighbor examples in that feature space. The similarity between (Zhao & Cho, 2018) and *Manifold Mixup* is that both consider linear interpolations of hidden representations with the same interpolation applied to their labels. However, an important difference is that *Manifold Mixup* backpropagates gradients through the earlier parts of the network (the layers before the point where mixup is applied), unlike (Zhao & Cho, 2018). In Section 4.3 we explain how this discrepancy significantly affects the learning process.

Table 1. Classification errors on (a) CIFAR-10 and (b) CIFAR-100. We include results from (Zhang et al., 2018c)† and (Guo et al., 2016)‡. Standard deviations over five repetitions.

PreActResNet18	Test Error (%)	Test NLL
No Mixup	4.83 ± 0.066	0.190 ± 0.003
AdaMix‡	3.52	NA
Input Mixup†	4.20	NA
Input Mixup ($\alpha = 1$)	3.82 ± 0.048	0.186 ± 0.004
Manifold Mixup ($\alpha = 2$)	$\underline{2.95 \pm 0.046}$	$\underline{0.137 \pm 0.003}$
PreActResNet34		
No Mixup	4.64 ± 0.072	0.200 ± 0.002
Input Mixup ($\alpha = 1$)	2.88 ± 0.043	0.176 ± 0.002
Manifold Mixup ($\alpha = 2$)	$\underline{2.54 \pm 0.047}$	$\underline{0.118 \pm 0.002}$
Wide-Resnet-28-10		
No Mixup	3.99 ± 0.118	0.162 ± 0.004
Input Mixup ($\alpha = 1$)	2.92 ± 0.088	0.173 ± 0.001
Manifold Mixup ($\alpha = 2$)	$\underline{2.55 \pm 0.024}$	$\underline{0.111 \pm 0.001}$

(a) CIFAR-10

PreActResNet18	Test Error (%)	Test NLL
No Mixup	24.01 ± 0.376	1.189 ± 0.002
AdaMix‡	20.97	n/a
Input Mixup†	21.10	n/a
Input Mixup ($\alpha = 1$)	22.11 ± 0.424	1.055 ± 0.006
Manifold Mixup ($\alpha = 2$)	$\underline{20.34 \pm 0.525}$	$\underline{0.912 \pm 0.002}$
PreActResNet34		
No Mixup	23.55 ± 0.399	1.189 ± 0.002
Input Mixup ($\alpha = 1$)	20.53 ± 0.330	1.039 ± 0.045
Manifold Mixup ($\alpha = 2$)	$\underline{18.35 \pm 0.360}$	$\underline{0.877 \pm 0.053}$
Wide-Resnet-28-10		
No Mixup	21.72 ± 0.117	1.023 ± 0.004
Input Mixup ($\alpha = 1$)	18.89 ± 0.111	0.927 ± 0.031
Manifold Mixup ($\alpha = 2$)	$\underline{18.04 \pm 0.171}$	$\underline{0.809 \pm 0.005}$

(b) CIFAR-100

AdaMix (Guo et al., 2018a) is another related method which attempts to learn better mixing distributions to avoid overlap. AdaMix performs interpolations only on the input space, reporting that their method degrades significantly when applied to hidden layers. Thus, AdaMix may likely work for different reasons than *Manifold Mixup*, and perhaps the two are complementary. AgrLearn (Guo et al., 2018b) adds an information bottleneck layer to the output of deep neural networks. AgrLearn leads to substantial improvements, achieving 2.45% test error on CIFAR-10 when combined with Input Mixup (Zhang et al., 2018c). As AgrLearn is complimentary to Input Mixup, it may be also complimentary to *Manifold Mixup*. Wang et al. (2018a) proposed an interpolation exclusively in the output space, does not backpropagate through the interpolation procedure, and has a very different framing in terms of the Euler-Lagrange equation (Equation 2) where the cost is based on unlabeled data (and the pseudolabels at those points) and the labeled data provide constraints.

4.5. Experiments

We now turn to the empirical evaluation of *Manifold Mixup*. We will study its regularization properties in supervised learning (Section 4.5.1), as well as how it affects the robustness of neural networks to novel input deformations (Section 4.5.2), and adversarial examples (Section 4.5.3).

4.5.1. Generalization on Supervised Learning

We train a variety of residual networks (He et al., 2016b) using different regularizers: no regularization, AdaMix, Input Mixup, and *Manifold Mixup*. We follow the training procedure of (Zhang et al., 2018c), which is to use SGD with momentum, a weight decay

Table 2. Classification errors and neg-log-likelihoods on SVHN. We run each experiment five times.

PreActResNet18	Test Error (%)	Test NLL
No Mixup	2.89 ± 0.224	0.136 ± 0.001
Input Mixup ($\alpha = 1$)	2.76 ± 0.014	0.212 ± 0.011
Manifold Mixup ($\alpha = 2$)	2.27 ± 0.011	0.122 ± 0.006
PreActResNet34		
No Mixup	2.97 ± 0.004	0.165 ± 0.003
Input Mixup ($\alpha = 1$)	2.67 ± 0.020	0.199 ± 0.009
Manifold Mixup ($\alpha = 2$)	2.18 ± 0.004	0.137 ± 0.008
Wide-Resnet-28-10		
No Mixup	2.80 ± 0.044	0.143 ± 0.002
Input Mixup ($\alpha = 1$)	2.68 ± 0.103	0.184 ± 0.022
Manifold Mixup ($\alpha = 2$)	2.06 ± 0.068	0.126 ± 0.008

Table 3. Accuracy on TinyImagenet.

PreActResNet18	top-1	top-5
No Mixup	55.52	71.04
Input Mixup ($\alpha = 0.2$)	56.47	71.74
Input Mixup ($\alpha = 0.5$)	55.49	71.62
Input Mixup ($\alpha = 1.0$)	52.65	70.70
Input Mixup ($\alpha = 2.0$)	44.18	68.26
Manifold Mixup ($\alpha = 0.2$)	58.70	73.59
Manifold Mixup ($\alpha = 0.5$)	57.24	73.48
Manifold Mixup ($\alpha = 1.0$)	56.83	73.75
Manifold Mixup ($\alpha = 2.0$)	48.14	71.69

of 10^{-4}, and a step-wise learning rate decay. We show results for the CIFAR-10 (Table 1a), CIFAR-100 (Table 1b), SVHN (Table 2), and TinyImageNET (Table 3) datasets. *Manifold Mixup* outperforms vanilla training, AdaMix, and Input Mixup across datasets and model architectures. Furthermore, *Manifold Mixup* leads to models with significantly better Negative Log-Likelihood (NLL) on the test data. In the case of CIFAR-10, *Manifold Mixup* models achieve as high as 50% relative improvement of test NLL.

As a complimentary experiment to better understand why *Manifold Mixup* works, we zeroed gradient updates immediately after the layer where mixup is applied. On the dataset CIFAR-10 and using a PreActResNet18, this led to a 4.33% test error, which is worse than our results for Input Mixup and *Manifold Mixup*, yet better than the baseline. Because *Manifold Mixup* selects the mixing layer at random, each layer is still being trained even when zeroing gradients, although it will receive less updates. This demonstrates that *Manifold Mixup* improves performance by updating the layers both before and after the mixing operation.

We also compared *Manifold Mixup* against other strong regularizers. For each regularizer, we selected the best hyper-parameters using a validation set. The training of PreActResNet50 on CIFAR-10 for 600 epochs led to the following test errors (%): no regularization (4.96 ± 0.19), Dropout (5.09 ± 0.09), Cutout (Devries & Taylor, 2017) (4.77 ± 0.38), Mixup (4.25 ± 0.11), and Manifold Mixup (3.77 ± 0.18). (Note that the results in Table 1 for PreActResNet were run for 1200 epochs, and therefore are not directly comparable to the numbers in this paragraph.)

To provide further evidence about the quality of representations learned with *Manifold Mixup*, we applied a k-nearest neighbour classifier on top of the features extracted from a PreActResNet18 trained on CIFAR-10. We achieved test errors of 6.09% (vanilla training), 5.54% (Input Mixup), and 5.16% (*Manifold Mixup*).

Finally, Table 6 and Table 5 show the sensitivity of *Manifold Mixup* to the hyper-parameter α and the set of eligible layers $\mathcal{S}$. (These results are based on training a PreActResNet18 for 2000 epochs, so these numbers are not exactly comparable to the ones in Table 1.) This shows that *Manifold Mixup* is robust with respect to choice of hyper-parameters, with improvements for many choices.

Table 4. Test accuracy on novel deformations. All models trained on normal CIFAR-100.

Deformation	No Mixup	Input Mixup ($\alpha = 1$)	Input Mixup ($\alpha = 2$)	*Manifold Mixup* ($\alpha = 2$)
Rotation U(−20°,20°)	52.96	55.55	56.48	<u>60.08</u>
Rotation U(−40°,40°)	33.82	37.73	36.78	<u>42.13</u>
Shearing U(−28.6°, 28.6°)	55.92	58.16	60.01	<u>62.85</u>
Shearing U(−57.3°, 57.3°)	35.66	39.34	39.7	<u>44.27</u>
Zoom In (60% rescale)	12.68	<u>13.75</u>	13.12	11.49
Zoom In (80% rescale)	47.95	52.18	50.47	<u>52.70</u>
Zoom Out (120% rescale)	43.18	60.02	61.62	<u>63.59</u>
Zoom Out (140% rescale)	19.34	41.81	42.02	<u>45.29</u>

Table 5. Test accuracy *Manifold Mixup* for different sets of eligible layers $\mathcal{S}$ on CIFAR.

$\mathcal{S}$	CIFAR-10	CIFAR-100
$\{0, 1, 2\}$	<u>97.23</u>	79.60
$\{0, 1\}$	96.94	78.93
$\{0, 1, 2, 3\}$	96.92	<u>80.18</u>
$\{1, 2\}$	96.35	78.69
$\{0\}$	96.73	78.15
$\{1, 2, 3\}$	96.51	79.31
$\{1\}$	96.10	78.72
$\{2, 3\}$	95.32	76.46
$\{2\}$	95.19	76.50
$\{\}$	95.27	76.40

Table 6. Test accuracy (%) of Input Mixup and *Manifold Mixup* for different α on CIFAR-10.

α	Input Mixup	*Manifold Mixup*
0.5	96.68	<u>96.76</u>
1.0	96.75	<u>97.00</u>
1.2	96.72	<u>97.03</u>
1.5	96.84	<u>97.10</u>
1.8	96.80	<u>97.15</u>
2.0	96.73	<u>97.23</u>

4.5.2. Generalization to Novel Deformations

To further evaluate the quality of representations learned with *Manifold Mixup*, we train PreActResNet34 models on the normal CIFAR-100 training split, but test them on novel (not seen during training) deformations of the test split. These deformations include random rotations, random shearings, and different rescalings. Better representations should generalize to a larger variety of deformations. Table 4 shows that networks trained using *Manifold Mixup* are the most able to classify test instances subject to novel deformations, which suggests the learning of better representations.

4.5.3. Robustness to Adversarial Examples

Adversarial robustness is related to the position of the decision boundary relative to the data. Because *Manifold Mixup* only considers some directions around data points (those corresponding to interpolations), we would not expect the model to be robust to adversarial attacks that consider any direction around each example. However, since *Manifold Mixup* expands the set of examples seen during training, an intriguing hypothesis is that these expansions overlap with the set of possible adversarial examples, providing some degree of

Table 7. Test accuracy on white-box FGSM adversarial examples on CIFAR-10 and CIFAR-100 (using a PreActResNet18 model) and SVHN (using a WideResNet20-10 model). We include the results of (Madry et al., 2018)†.

CIFAR-10	FGSM
No Mixup	36.32
Input Mixup ($\alpha = 1$)	71.51
Manifold Mixup ($\alpha = 2$)	<u>77.50</u>
PGD training (7-steps)†	56.10
CIFAR-100	**FGSM**
Input Mixup ($\alpha = 1$)	40.7
Manifold Mixup ($\alpha = 2$)	44.96
SVHN	**FGSM**
No Mixup	21.49
Input Mixup ($\alpha = 1$)	56.98
Manifold Mixup ($\alpha = 2$)	65.91
PGD training (7-steps)†	<u>72.80</u>

defense. If this hypothesis is true, *Manifold Mixup* would force adversarial attacks to consider a wider set of directions, leading to a larger computational expense for the attacker. To explore this, we consider the Fast Gradient Sign Method (FGSM, Goodfellow et al., 2015), which constructs adversarial examples in one single step, thus considering a relatively small subset of directions around examples. The performance of networks trained using *Manifold Mixup* against FGSM attacks is given in Table 7. One challenge in evaluating robustness against adversarial examples is the "gradient masking problem", in which a defense succeeds only by reducing the quality of the gradient signal. (Athalye et al., 2018) explored this issue in depth, and proposed running an unbounded search for a large number of iterations to confirm the quality of the gradient signal. *Manifold Mixup* passes this sanity check. While we found that using *Manifold Mixup* improves the robustness to single-step FGSM attack (especially over Input Mixup), we found that *Manifold Mixup* did not significantly improve robustness against stronger, multi-step attacks such as PGD (Madry et al., 2018).

4.6. Connections to Neuroscience and Credit Assignment

We present an intriguing connection between *Manifold Mixup* and a challenging problem in neuroscience. At a high level, we can imagine systems in the brain which compute predictions from a stream of changing inputs, and pass these predictions onto other modules which return some kind of feedback signal (Lee et al., 2015; Scellier & Bengio, 2017; Whittington & Bogacz, 2017; Bartunov et al., 2018). For instance, these feedback signals can be gradients or targets for prediction. There is a delay between the output of the prediction and the point in time in which the feedback can return to the system after having travelled across the brain. Moreover, this delay could be noisy and could differ based on the type of the prediction or other conditions in the brain, as well as depending on which paths are considered (there are many skip connections between areas). This means that it could be very difficult for a system

in the brain to establish a clear correspondence between its outputs and the feedback signals that it receives over time.

While it is preliminary, an intriguing hypothesis is that part of how systems in the brain could be working around this limitation is by averaging their states and feedback signals across multiple points in time. The empirical results from mixup suggest that such a technique may not just allow successful computation, but also act as a potent regularizer. *Manifold Mixup* strengthens this result by showing that the same regularization effect can be achieved from mixing in higher level hidden representations.

4.7. Conclusion

Deep neural networks often give incorrect, yet extremely confident predictions on examples that differ from those seen during training. This problem is one of the most central challenges in deep learning. We have investigated this issue from the perspective of the representations learned by deep neural networks. We observed that vanilla neural networks spread the training data widely throughout the representation space, and assign high confidence predictions to almost the entire volume of representations. This leads to major drawbacks since the network will provide high-confidence predictions to examples off the data manifold, thus lacking enough incentives to learn discriminative representations about the training data. To address these issues, we introduced *Manifold Mixup*, a new algorithm to train neural networks on interpolations of hidden representations. *Manifold Mixup* encourages the neural network to be uncertain across the volume of the representation space unseen during training. This leads to concentrating the representations of the real training examples in a low dimensional subspace, resulting in more discriminative features. Throughout a variety of experiments, we have shown that neural networks trained using *Manifold Mixup* have better generalization in terms of error and log-likelihood, as well as better robustness to novel deformations of the data and adversarial examples. Being easy to implement and incurring little additional computational cost, we hope that *Manifold Mixup* will become a useful regularization tool for deep learning practitioners.

Chapter 5

Prologue to the second article

5.1. Article Details

Interpolation consistency training for semi-supervised learning. Vikas Verma, Alex Lamb, Juho Kannala, Yoshua Bengio, David Lopez-Paz.

Interpolation Consistency Training (ICT) is a semi-supervised learning technique which tries to improve consistency using unlabeled samples. We proposed a technique in which we produce classification decisions (pseudolabels) for random pairs of unlabeled samples, and then train a linear combination of those samples with soft-targets given by interpolation of the associated pseudolabels. Additionally, we improve the accuracy of the pseudolabels by producing them with a network with a moving average of the parameters over the training iterates (Tarvainen & Valpola, 2017b).

5.2. Personal Contribution

. This algorithm was originally a part of the Manifold Mixup paper. After our first submission and rejection by a conference, we noticed that reviewers did not seem to place much emphasis on the semi-supervised learning result, so we decided to move them into their own paper. We felt that this would lead the idea to get more attention and also felt that it would make the description of the algorithm in both papers more straightforward and easier to read. David Lopez-Paz reached out to us to introduce the idea of running on the two moons semi-supervised learning dataset. Alex Lamb produced the initial successful results on the two moon dataset. Alex Lamb and Vikas Verma developed the basic idea together, while Vikas Verma did most of the experiments for the paper. I asked Vikas Verma and he gave me permission to use this paper in my thesis.

5.3. Context

Interpolation Consistency Training builds on the consistency-based approach to semi-supervised learning in which the area around unlabeled points (in some sense) are encouraged to have similar predictions. This algorithm was originally a part of the Manifold Mixup paper, where we found that using pseudolabels instead of the original labels also led to improvements. ICT is a way of combining mixup with semi-supervised learning, and yielded significant improvements on semi-supervised baselines.

Consistency-based semi-supervised learning serve as the foundation for this technique (Chapelle et al., 2009; Tarvainen & Valpola, 2017a). While virtual adversarial training (Miyato et al., 2018a) used adversarial perturbations, we found that using perturbations based on linear interpolations was faster and achieved better performance.

5.4. Contributions

Our results suggest that using a richer class of perturbations (as opposed to simple gaussian noise) improves performance on semi-supervised learning. Additionally, our perturbations are much more non-local than what had been considered previously, as they can consist of interpolations between completely different examples. We found that ICT solves the standard two moon semi-supervised learning task, in which there are two crescent-shaped clusters of data, with each only having three labeled examples. The simplest classifier which perfectly fits just the labeled data is a linear classifier, which fails to correctly classify the ends of the crescents. On the other hand, a classifier may obtain perfect accuracy by exploiting the cluster assumption, which ensures that the decision boundary does not pass through the cluster. The successful results on this task provide evidence that ICT succeeds by exploiting the cluster assumption. We also evaluated ICT on semi-supervised CIFAR-10 and SVHN benchmarks using a 13 layer convolutional neural network. We obtained state-of-the-art results at the time of the ICT paper's publication. We also found that the benefits from using ICT greatly exceeded those obtained by using mixup (Zhang et al., 2018c) or manifold mixup (Verma et al., 2018) on just the labeled data. We achieved these improvements using 1000, 2000, or 4000 labeled examples on CIFAR-10 and 250, 500, or 1000 labeled examples on SVHN. We also obtained state-of-the-art semi-supervised learning results on CIFAR-10 and SVHN when using the WideResNet architecture (Zagoruyko & Komodakis, 2016a).

5.5. Research Impact

The ICT technique has been used in many subsequent SOTA works in semi-supervised learning (Berthelot et al., 2019b,a). It has also been used in some real-world applications such as graph neural networks in GraphMix (Verma et al., 2019b) and object detection (Jeong et al., 2021). In GraphMix, node classification tasks were considered in which the entire dataset is a single large graph and each example is a node. A small fraction of the nodes are labeled while most nodes are unlabeled (and some labeled nodes are held out as evaluation data). In the GraphMix algorithm, the pseudolabels are produced using a moving-average network of the graph-neural-network, which has access to the entire graph. The mixup-process is done on a per-node basis, and this student network is tasked with predicting the pseudolabels generated using the entire graph. This achieved state-of-the-art results on several of the most widely studied node classification tasks.

Chapter 6

Interpolation Consistency Training

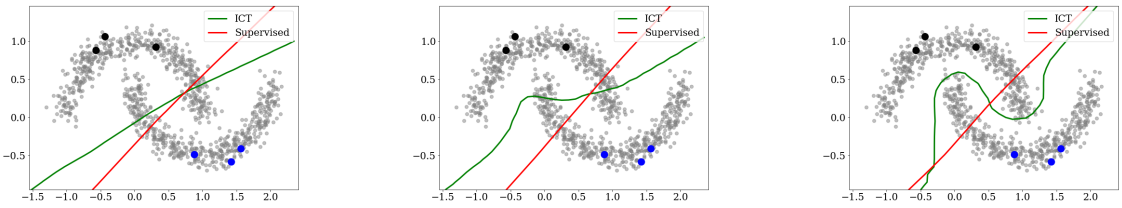
6.1. Introduction

Deep learning achieves excellent performance in supervised learning tasks where labeled data is abundant (LeCun et al., 2015). However, labeling large amounts of data is often prohibitive due to time, financial, and expertise constraints. As machine learning permeates an increasing variety of domains, the number of applications where unlabeled data is voluminous and labels are scarce increases: for instance, recognizing documents in extinct languages, where a machine learning system has access to a few labels, produced by highly-skilled scholars (Clanuwat et al., 2018).

The goal of Semi-Supervised Learning (SSL) (Chapelle et al., 2010) is to leverage large amounts of unlabeled data to improve the performance of supervised learning over small datasets. Often, SSL algorithms use unlabeled data to learn additional structure about the input distribution. For instance, the existence of cluster structures in the input distribution could hint the separation of samples into different labels. This is often called the *cluster assumption*: if two samples belong to the same cluster in the input distribution, then they are likely to belong to the same class. The cluster assumption is equivalent to the *low-density separation assumption*: the decision boundary should lie in the low-density regions. The equivalence is easy to infer: A decision boundary which lies in a high-density region, will cut a cluster into two different classes, requiring that samples from different classes lie in the same cluster; which is the violation of the *cluster assumption*. The *low-density separation assumption* has inspired many recent *consistency-regularization* semi-supervised learning techniques, including the Π-model (Sajjadi et al., 2016; Laine & Aila, 2016), temporal ensembling (Laine & Aila, 2016), VAT (Miyato et al., 2018a), and the Mean-Teacher (Tarvainen & Valpola, 2017b).

Consistency regularization methods for semi-supervised learning enforce the low-density separation assumption by encouraging invariant prediction $f(u) = f(u + \delta)$ for perturbations $u + \delta$ of unlabeled points u. Such consistency and small prediction error can be satisfied simultaneously if and only if the decision boundary traverses a low-density path.

Different consistency regularization techniques vary in how they choose the unlabeled data perturbations δ. One simple alternative is to use random perturbations δ. However, random perturbations are inefficient in high dimensions, as only a tiny proportion of input perturbations are capable of pushing the decision boundary into low-density regions. To alleviate this issue, Virtual Adversarial Training or VAT (Miyato et al., 2018a), searches for small perturbations δ that maximize the change in the prediction of the model. This involves



(a) After 100 updates **(b)** After 500 updates **(c)** After 1000 updates

Fig. 1. Interpolation Consistency Training (ICT) applied to the "two moons" dataset, when three labels per class (large dots) and a large amount of unlabeled data (small dots) is available. When compared to supervised learning (red), ICT encourages a decision boundary traversing a low-density region that would better reflect the structure of the unlabeled data. Both methods employ a multilayer perceptron with three hidden ReLU layers of twenty neurons.

computing the gradient of the predictor with respect to its input, which can be expensive for large neural network models. More specifically, the backpropagated gradients need to be computed twice: the first time to produce the adversarial perturbation and the second time to produce gradients for updating the parameters.

This additional computation makes VAT (Miyato et al., 2018a) and other related methods such as (Park et al., 2018) less appealing in situations where unlabeled data is available in large quantities. Furthermore, recent research has shown that training with adversarial perturbations can hurt generalization performance (Nakkiran, 2019; Tsipras et al., 2018).

To overcome the above limitations, we propose the Interpolation Consistency Training (ICT), an efficient consistency regularization technique for state-of-the-art semi-supervised learning. In a nutshell, ICT regularizes semi-supervised learning by encouraging consistent predictions $f(\alpha u_1 + (1-\alpha)u_2) = \alpha f(u_1) + (1-\alpha) f(u_2)$ at interpolations $\alpha u_1 + (1-\alpha)u_2$ of unlabeled points u_1 and u_2.

Our experimental results on the benchmark datasets CIFAR10 and SVHN and neural network architectures CNN-13 (Laine & Aila, 2016; Miyato et al., 2018a; Tarvainen & Valpola, 2017b; Park et al., 2018; Luo et al., 2018b) and WRN28-2 (Oliver et al., 2018) outperform (or are competitive with) the state-of-the-art methods. ICT is simpler and more computation efficient than several of the recent SSL algorithms, making it an appealing approach to SSL. Figure 1 illustrates how ICT learns a decision boundary traversing a low density region in the "two moons" problem.

6.2. Interpolation Consistency Training

Given a mixup (Zhang et al., 2018b) operation:

$$\text{Mix}_\lambda(a, b) = \lambda \cdot a + (1-\lambda) \cdot b,$$

Interpolation Consistency Training(ICT) trains a prediction model f_θ to provide consistent predictions at interpolations of unlabeled points:

$$f_\theta(\text{Mix}_\lambda(u_j, u_k)) \approx \text{Mix}_\lambda(f_{\theta'}(u_j), f_{\theta'}(u_k)),$$

where θ' is a moving average of θ (Figure 2). But, why do interpolations between unlabeled samples provide a good consistency perturbation for semi-supervised training?

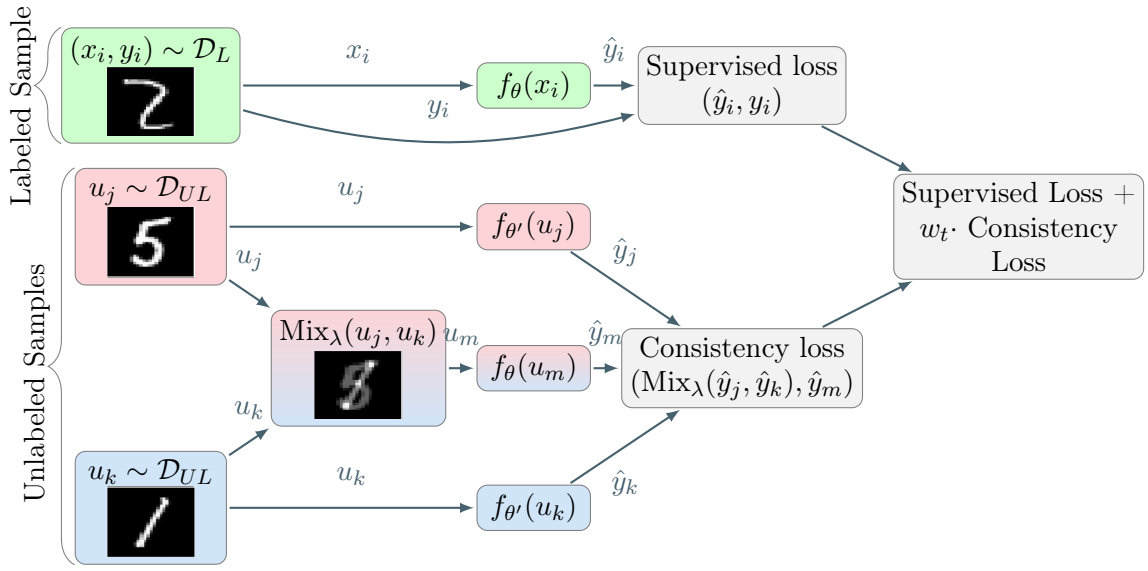


Fig. 2. Interpolation Consistency Training (ICT) learns a student network f_θ in a semi-supervised manner. To this end, ICT uses a mean-teacher $f_{\theta'}$, where the teacher parameters θ' are an exponential moving average of the student parameters θ. During training, the student parameters θ are updated to encourage consistent predictions $f_\theta(\text{Mix}_\lambda(u_j, u_k)) \approx \text{Mix}_\lambda(f_{\theta'}(u_j), f_{\theta'}(u_k))$, and correct predictions for labeled examples x_i.

To begin with, observe that the most useful samples on which the consistency regularization should be applied are the samples near the decision boundary. Adding a small perturbation δ to such low-margin unlabeled samples u_j is likely to push $u_j + \delta$ over the other side of the decision boundary. This would violate the *low-density separation assumption*, making $u_j + \delta$ a good place to apply consistency regularization. These violations do not occur at high-margin unlabeled points that lie far away from the decision boundary.

Back to low-margin unlabeled points u_j, how can we find a perturbation δ such that u_j and $u_j + \delta$ lie on opposite sides of the decision boundary? Although tempting, using random perturbations is an inefficient strategy, since the subset of directions approaching the decision boundary is a tiny fraction of the ambient space.

Instead, consider interpolations $u_j + \delta = \text{Mix}_\lambda(u_j, u_k)$ towards a second randomly selected unlabeled examples u_k. Then, the two unlabeled samples u_j and u_k can either:

(1) lie in the same cluster,

(2) lie in different clusters but belong to the same class,

(3) lie on different clusters and belong to the different classes.

Assuming the cluster assumption, the probability of (1) decreases as the number of classes increases. The probability of (2) is low if we assume that the number of clusters for each class is balanced. Finally, the probability of (3) is the highest. Then, assuming that one of (u_j, u_k) lies near the decision boundary (it is a good candidate for enforcing consistency), it is likely (because of the high probability of (3)) that the interpolation towards u_k points towards a region of low density, followed by the cluster of the other class. Since this is a good direction to move the decision, the interpolation is a good perturbation for consistency-based regularization.

Our exposition has argued so far that interpolations between random unlabeled samples are likely to fall in low-density regions. Thus, such interpolations are good locations where consistency-based regularization could be applied. But how should we label those interpolations? Unlike random or adversarial perturbations of single unlabeled examples u_j, our scheme involves two unlabeled examples (u_j, u_k). Intuitively, we would like to push the decision boundary as far as possible from the class boundaries, as it is well known that decision boundaries with large margin generalize better (Shawe-Taylor et al., 1996). In the supervised learning setting, one method to achieve large-margin decision boundaries is mixup (Zhang et al., 2018b). In mixup, the decision boundary is pushed far away from the class boundaries by enforcing the prediction model to change linearly in between samples. This is done by training the model f_θ to predict $\text{Mix}_\lambda(y, y')$ at location $\text{Mix}_\lambda(x, x')$, for random pairs of labeled samples $((x, y), (x', y'))$. Here we extend mixup to the semi-supervised learning setting by training the model f_θ to predict the "fake label" $\text{Mix}_\lambda(f_\theta(u_j), f_\theta(u_k))$ at location $\text{Mix}_\lambda(u_j, u_k)$. In order to follow a more conservative consistent regularization, we encourage the model f_θ to predict the fake label $\text{Mix}_\lambda(f_{\theta'}(u_j), f_{\theta'}(u_k))$ at location $\text{Mix}_\lambda(u_j, u_k)$, where θ' is a moving average of θ, also known as a *mean-teacher* (Tarvainen & Valpola, 2017b).

We are now ready to describe in detail the proposed Interpolation Consistency Training (ICT). Consider access to labeled samples $(x_i, y_i) \sim \mathcal{D}_L$, drawn from the joint distribution $P(X, Y)$. Also, consider access to unlabeled samples $u_j, u_k \sim \mathcal{D}_{UL}$, drawn from the marginal distribution $P(X) = \frac{P(X,Y)}{P(Y|X)}$ Our learning goal is to train a model f_θ, able to predict Y from X. By using stochastic gradient descent, at each iteration t, update the parameters θ to minimize

$$L = L_S + w(t) \cdot L_{US}$$

where L_S is the usual cross-entropy supervised learning loss over labeled samples $\mathcal{D}_L$, and L_{US} is our new interpolation consistency regularization term. These two losses are computed on top of (labeled and unlabeled) minibatches, and the ramp function $w(t)$ increases the importance of the consistency regularization term L_{US} after each iteration. To compute L_{US}, sample two minibatches of unlabeled points u_j and u_k, and compute their fake labels $\hat{y}_j = f_{\theta'}(u_j)$ and $\hat{y}_k = f_{\theta'}(u_k)$, where θ' is an moving average of θ (Tarvainen & Valpola, 2017b). Second, compute the interpolation $u_m = \text{Mix}_\lambda(u_j, u_k)$, as well as the model prediction at that location, $\hat{y}_m = f_\theta(u_m)$. Third, update the parameters θ as to bring the prediction $\hat{y}_m$ closer to the interpolation of the fake labels $\text{Mix}_\lambda(\hat{y}_j, \hat{y}_k)$. The discrepancy between the prediction $\hat{y}_m$ and $\text{Mix}_\lambda(\hat{y}_j, \hat{y}_k)$ can be measured using any loss; in our experiments, we use the mean squared error. Following (Zhang et al., 2018b), on each update we sample a random λ from $\text{Beta}(\alpha, \alpha)$.

In sum, the population version of our ICT term can be written as:

$$\mathcal{L}_{US} = \mathop{\mathbb{E}}_{u_j, u_k \sim P(X)} \mathop{\mathbb{E}}_{\lambda \sim \text{Beta}(\alpha,\alpha)} \ell(f_\theta(\text{Mix}_\lambda(u_j, u_k)), \text{Mix}_\lambda(f_{\theta'}(u_j), f_{\theta'}(u_k))) \tag{6.1}$$

ICT is summarized in Figure 2 and Algorithm 1.

6.3. Experiments

6.3.1. Datasets

We follow the common practice in semi-supervised learning literature (Laine & Aila, 2016; Miyato et al., 2018a; Tarvainen & Valpola, 2017b; Park et al., 2018; Luo et al., 2018b) and

Algorithm 1 The Interpolation Consistency Training (ICT) Algorithm

Require: $f_\theta(x)$: neural network with trainable parameters θ
Require: $f_{\theta'}(x)$ mean teacher with θ' equal to moving average of θ
Require: $\mathcal{D}_L(x, y)$: collection of the labeled samples
Require: $\mathcal{D}_{UL}(x)$: collection of the unlabeled samples
Require: α: rate of moving average
Require: $w(t)$: ramp function for increasing consistency regularization
Require: T: total number of iterations
Require: Q: random distribution on [0,1]
Require: $\text{Mix}_\lambda(a, b) = \lambda a + (1 - \lambda)b$.
  for $t = 1, \ldots, T$ **do**
    Sample $\{(x_i, y_i)\}_{i=1}^B \sim \mathcal{D}_L(x, y)$ ▷ Sample labeled minibatch
    $L_S = \text{CrossEntropy}(\{(f_\theta(x_i), y_i)\}_{i=1}^B)$ ▷ Supervised loss (cross-entropy)
    Sample $\{u_j\}_{j=1}^U, \{u_k\}_{k=1}^U \sim \mathcal{D}_{UL}(x)$ ▷ Sample two unlabeled minibatchs
    $\{\hat{y}_j\}_{j=1}^U = \{f_{\theta'}(u_j)\}_{j=1}^U$, $\{\hat{y}_k\}_{k=1}^U = \{f_{\theta'}(u_k)\}_{k=1}^U$ ▷ Compute psuedo labels
    Sample $\lambda \sim Q$ ▷ sample an interpolation coefficient
    $(u_m = \text{Mix}_\lambda(u_j, u_k), \hat{y}_m = \text{Mix}_\lambda(\hat{y}_j, \hat{y}_k))$ ▷ Compute interpolation
    $L_{US} = \text{ConsistencyLoss}(\{(f_\theta(u_m), \hat{y}_m)\}_{m=1}^U)$ ▷ e.g., mean squared error
    $L = L_S + w(t) \cdot L_{US}$ ▷ Total Loss
    $g_\theta \leftarrow \nabla_\theta L$ ▷ Compute Gradients
    $\theta' = \alpha\theta' + (1 - \alpha)\theta$ ▷ Update moving average of parameters
    $\theta \leftarrow \text{Step}(\theta, g_\theta)$ ▷ e.g. SGD, Adam
  end for
  return θ

conduct experiments using the CIFAR-10 and SVHN datasets, where only a fraction of the training data is labeled, and the remaining data is used as unlabeled data. We followed the standardized procedures laid out by (Oliver et al., 2018) to ensure a fair comparison.

The CIFAR-10 dataset consists of 60000 color images each of size 32×32, split between 50K training and 10K test images. This dataset has ten classes, which include images of natural objects such as cars, horses, airplanes and deer. The SVHN dataset consists of 73257 training samples and 26032 test samples each of size 32×32. Each example is a close-up image of a house number (the ten classes are the digits from 0-9).

We adopt the standard data-augmentation and pre-processing scheme which has become standard practice in the semi-supervised learning literature (Sajjadi et al., 2016; Laine & Aila, 2016; Tarvainen & Valpola, 2017b; Miyato et al., 2018a; Luo et al., 2018b; Athiwaratkun et al., 2019). More specifically, for CIFAR-10, we first zero-pad each image with 2 pixels on each side. Then, the resulting image is randomly cropped to produce a new 32×32 image. Next, the image is horizontally flipped with probability 0.5, followed by per-channel standardization and ZCA preprocessing. For SVHN, we zero-pad each image with 2 pixels on each side and then randomly crop the resulting image to produce a new 32×32 image, followed by zero-mean and unit-variance image whitening.

6.3.2. Models

We conduct our experiments using CNN-13 and Wide-Resnet-28-2 architectures. The CNN-13 architecture has been adopted as the standard benchmark architecture in recent state-of-the-art SSL methods (Laine & Aila, 2016; Tarvainen & Valpola, 2017b; Miyato et al., 2018a; Park et al., 2018; Luo et al., 2018b). We use its variant (i.e., without additive Gaussian noise in the input layer) as implemented in (Athiwaratkun et al., 2019). We also removed the Dropout noise to isolate the improvement achieved through our method. Other SSL methods in Table 1 and Table 2 use the Dropout noise, which gives them more regularizing capabilities. Despite this, our method outperforms other methods in several experimental settings.

(Oliver et al., 2018) performed a systematic study using Wide-Resnet-28-2 (Zagoruyko & Komodakis, 2016b), a specific residual network architecture, with extensive hyperparameter search to compare the performance of various consistency-based semi-supervised algorithms. We evaluate ICT using this same setup as a mean towards a fair comparison to these algorithms.

6.3.3. Implementation details

We used the SGD with nesterov momentum optimizer for all of our experiments. For the experiments in Table 1 and Table 2, we run the experiments for 400 epochs. For the experiments in Table 3, we run experiments for 600 epochs. The initial learning rate was set to 0.1, which is then annealed using the cosine annealing technique proposed in (Loshchilov & Hutter, 2016) and used by (Tarvainen & Valpola, 2017b). The momentum parameter was set to 0.9. We used an L2 regularization coefficient 0.0001 and a batch-size of 100 in our experiments.

In each experiment, we report mean and standard deviation across three independently run trials.

The consistency coefficient $w(t)$ is ramped up from its initial value 0.0 to its maximum value at one-fourth of the total number of epochs using the same sigmoid schedule of (Tarvainen & Valpola, 2017b). We used MSE loss for computing the consistency loss following (Laine & Aila, 2016; Tarvainen & Valpola, 2017b). We set the decay coefficient for the mean-teacher to 0.999 following (Tarvainen & Valpola, 2017b).

We conduct hyperparameter search over the two hyperparameters introduced by our method: the maximum value of the consistency coefficient $w(t)$ (we searched over the values in $\{1.0, 10.0, 20.0, 50.0, 100.0\}$) and the parameter α of distribution $Beta(\alpha, \alpha)$ (we searched over the values in $\{0.1, 0.2, 0.5, 1.0\}$). We select the best hyperparameter using a validation set of 5000 and 1000 labeled samples for CIFAR-10 and SVHN respectively. This size of the validation set is the same as that used in the other methods compared in this work.

We note the in all our experiments with ICT, to get the supervised loss, we perform the interpolation of labeled sample pair and their corresponding labels (as in *mixup* (Zhang et al., 2018b)). To make sure, that the improvements from ICT are not only because of the supervised *mixup* loss, we provide the direct comparison of ICT against supervised *mixup* and *Manifold Mixup* training in the Table 1 and Table 2.

6.3.4. Results

We provide the results for CIFAR10 and SVHN datasets using CNN-13 architecture in the Table 1 and Table 2, respectively.

Table 1. Error rates (%) on CIFAR-10 using CNN-13 architecture. We ran three trials for ICT.

Model	1k labeled 50k unlabeled	2k labeled 50k unlabeled	4k labeled 50k unlabeled
Supervised	39.95 ± 0.75	31.16 ± 0.66	21.75 ± 0.46
Supervised (Mixup)	36.48 ± 0.15	26.24 ± 0.46	19.67 ± 0.16
Supervised (Manifold Mixup)	34.58 ± 0.37	25.12 ± 0.52	18.59 ± 0.18
Π model (Laine & Aila, 2016)	31.65 ± 1.20	17.57 ± 0.44	12.36 ± 0.31
TempEns (Laine & Aila, 2016)	23.31 ± 1.01	15.64 ± 0.39	12.16 ± 0.24
Mean Teacher (Tarvainen & Valpola, 2017b)	21.55 ± 1.48	15.73 ± 0.31	12.31 ± 0.28
VAT (Miyato et al., 2018a)	–	–	$11.36 \pm$ NA
VAT+Ent (Miyato et al., 2018a)	–	–	$10.55 \pm$ NA
VAdD (Park et al., 2018)	–	–	11.32 ± 0.11
SNTG (Luo et al., 2018b)	18.41 ± 0.52	13.64 ± 0.32	10.93 ± 0.14
MT+ Fast SWA (Athiwaratkun et al., 2019)	$15.58 \pm$ NA	$11.02 \pm$ NA	$9.05 \pm$ NA
ICT	$\mathbf{15.48 \pm 0.78}$	$\mathbf{9.26 \pm 0.09}$	$\mathbf{7.29 \pm 0.02}$

Table 2. Error rates (%) on SVHN using CNN-13 architecture. We ran three trials for ICT.

Model	250 labeled 73257 unlabeled	500 labeled 73257 unlabeled	1000 labeled 73257 unlabeled
Supervised	40.62 ± 0.95	22.93 ± 0.67	15.54 ± 0.61
Supervised (Mixup)	33.73 ± 1.79	21.08 ± 0.61	13.70 ± 0.47
Supervised (Manifold Mixup)	31.75 ± 1.39	20.57 ± 0.63	13.07 ± 0.53
Π model (Laine & Aila, 2016)	9.93 ± 1.15	6.65 ± 0.53	4.82 ± 0.17
TempEns (Laine & Aila, 2016)	12.62 ± 2.91	5.12 ± 0.13	4.42 ± 0.16
MT (Tarvainen & Valpola, 2017b)	4.35 ± 0.50	4.18 ± 0.27	3.95 ± 0.19
VAT (Miyato et al., 2018a)	–	–	$5.42 \pm$ NA
VAT+Ent (Miyato et al., 2018a)	–	–	$3.86 \pm$ NA
VAdD (Park et al., 2018)	–	–	4.16 ± 0.08
SNTG (Luo et al., 2018b)	$\mathbf{4.29 \pm 0.23}$	$\mathbf{3.99 \pm 0.24}$	$\mathbf{3.86 \pm 0.27}$
ICT	4.78 ± 0.68	4.23 ± 0.15	3.89 ± 0.04

To justify the use of a SSL algorithm, one must compare its performance against the state-of-the-art supervised learning algorithm (Oliver et al., 2018). To this end, we compare our method against two state-of-the-art supervised learning algorithms (Zhang et al., 2018b; Verma et al., 2018), denoted as Supervised(Mixup) and Supervised(Manifold Mixup), respectively in

Table 3. Results on CIFAR10 (4000 labels) and SVHN (1000 labels) (in test error %). All results use the same standardized architecture (WideResNet-28-2). Each experiment was run for three trials. † refers to the results reported in (Oliver et al., 2018). We did not conduct any hyperparameter search and used the best hyperparameters found in the experiments of Table 1 and 2 for CIFAR10(4000 labels) and SVHN(1000 labels)

SSL Approach	CIFAR10 4000 labeled 50000 unlabeled	SVHN 1000 labeled 73257 unlabeled
Supervised †	20.26 ± 0.38	12.83 ± 0.47
Mean-Teacher †	15.87 ± 0.28	5.65 ± 0.47
VAT †	13.86 ± 0.27	5.63 ± 0.20
VAT-EM †	13.13 ± 0.39	5.35 ± 0.19
ICT	$\mathbf{7.66 \pm 0.17}$	$\mathbf{3.53 \pm 0.07}$

Table 1 and 2. ICT method passes this test with a wide margin, often resulting in a two-fold reduction in the test error in the case of CIFAR10 (Table 1) and a four-fold reduction in the case of SVHN (Table 2)

Furthermore, in Table 1, we see that ICT improves the test error of other strong SSL methods. For example, in the case of 4000 labeled samples, it improves the test error of best-reported method by $\sim 25\%$. The best values of the hyperparameter max-consistency coefficient for 1000, 2000 and 4000 labels experiments were found to be 10.0, 100.0 and 100.0 respectively and the best values of the hyperparameter α for 1000, 2000 and 4000 labels experiments were found to be 0.2, 1.0 and 1.0 respectively. In general, we observed that for less number of labeled data, lower values of max-consistency coefficient and α obtained better validation errors.

For SVHN, the test errors obtained by ICT are competitive with other state-of-the-art SSL methods (Table 2). The best values of the hyperparameters max-consistency coefficient and α were found to be 100 and 0.1 respectively, for all the ICT results reported in the Table 2.

(Oliver et al., 2018) performed extensive hyperparameter search for various consistency regularization SSL algorithm using the WRN-28-2 and they report the best test errors found for each of these algorithms. For a fair comparison of ICT against these SSL algorithms, we conduct experiments on WRN-28-2 architecture. The results are shown in Table 3. ICT achieves improvement over other methods both for the CIFAR10 and SVHN datasets.

We note that unlike other SSL methods of Table 1, Table 2 and Table 3, we do not use Dropout regularizer in our implementation of CNN-13 and WRN-28-2. Using Dropout along with the ICT may further reduce the test error.

6.3.5. Ablation Study

- Effect of not using the mean-teacher in ICT: We note that Π-model, VAT and VAdD methods in Table 1 and Table 2 do not use a mean-teacher to make predictions on the unlabeled data. Although the mean-teacher (Tarvainen & Valpola, 2017b) used in ICT does not incur any significant computation cost, one might argue that a more direct comparison with Π-model, VAT and VAdD methods requires not using

a mean-teacher. To this end, we conduct an experiment on the CIFAR10 dataset, without the mean-teacher in ICT, i.e. the prediction on the unlabeled data comes from the network $f_\theta(x)$ instead of the mean-teacher network $f_{\theta'}(x)$ in Equation 6.1. We obtain test errors of $19.56 \pm 0.56\%$, $14.35 \pm 0.15\%$ and $11.19 \pm 0.14\%$ for 1000, 2000, 4000 labeled samples respectively (We did not conduct any hyperparameter search for these experiments and used the best hyperparameters found in the ICT experiments of Table 1). This shows that even without a mean-teacher, ICT has major a advantage over methods such as VAT (Miyato et al., 2018a) and VAdD (Park et al., 2018) that it does not require an additional gradient computation yet performs on the same level of the test error.

- Effect of not having the *mixup* supervised loss: In Section 6.3.3, we noted that to get the supervised loss, we perform the interpolation of labeled sample pair and their corresponding labels (*mixup* supervised loss as in (Zhang et al., 2018b)). Will the performance of ICT be significantly reduced by not having the *mixup* supervised loss? We conducted experiments with ICT on both CIFAR10 and SVHN with the *vanilla* supervised loss. For CIFAR10, we obtained test errors of 14.86 ± 0.39, 9.02 ± 0.12 and 8.23 ± 0.22 for 1000, 2000 and 4000 labeled samples respectively. We did not conduct any hyperparameter search and used the best values of hyperparameters (max-consistency coefficient and α) found in the experiments of the Table 1. We observe that in the case of 1000 and 2000 labeled samples, there is no increase in the test error (w.r.t having the *mixup* supervised loss), whereas in the case of 4000 labels, the test error increases by approximately 1% . This suggests that, in the low labeled data regimes, not having the *mixup* supervised loss in the ICT does not incur any significant increase in the test error.

6.4. Related Work

This work builds on two threads of research: consistency-regularization for semi-supervised learning and interpolation-based regularizers.

On the one hand, consistency-regularization semi-supervised learning methods (Sajjadi et al., 2016; Laine & Aila, 2016; Tarvainen & Valpola, 2017b; Miyato et al., 2018a; Luo et al., 2018b; Athiwaratkun et al., 2019) encourage that realistic perturbations $u + \delta$ of unlabeled samples u should not change the model predictions $f_\theta(u)$. These methods are motivated by the *low-density separation assumption* (Chapelle et al., 2010), and as such push the decision boundary to lie in the low-density regions of the input space, achieving larger classification margins. ICT differs from these approaches in two aspects. First, ICT chooses perturbations in the direction of another randomly chosen unlabeled sample, avoiding expensive gradient computations. When interpolating between distant points, the regularization effect of ICT applies to larger regions of the input space.

On the other hand, interpolation-based regularizers (Zhang et al., 2018b; Tokozume et al., 2018a; Verma et al., 2018) have been recently proposed for supervised learning, achieving state-of-the-art performances across a variety of tasks and network architectures. While (Zhang et al., 2018b; Tokozume et al., 2018a) was proposed to perform interpolations in the input space, (Verma et al., 2018) proposed to perform interpolation also in the hidden space representations. Furthermore, in the unsupervised learning setting, (Berthelot* et al., 2019) proposes to measure the realism of latent space interpolations from an autoencoder to improve its training.

Other works have approached semi-supervised learning from the perspective of generative models. Some have approached this from a consistency point of view, such as (Lecouat et al., 2018), who proposed to encourage smooth changes to the predictions along the data manifold estimated by the generative model (trained on both labeled and unlabeled samples). Others have used the discriminator from a trained generative adversarial network (Goodfellow et al., 2014) as a way of extracting features for a purely supervised model (Radford et al., 2015). Still, others have used trained inference models as a way of extracting features (Dumoulin et al., 2016b).

6.5. Conclusion

Machine learning is having a transformative impact on diverse areas, yet its application is often limited by the amount of available labeled data. Progress in semi-supervised learning techniques holds promise for those applications where labels are expensive to obtain. In this paper, we have proposed a simple but efficient semi-supervised learning algorithm, Interpolation Consistency Training(ICT), which has two advantages over previous approaches to semi-supervised learning. First, it uses almost no additional computation, as opposed to computing adversarial perturbations or training generative models. Second, it outperforms strong baselines on two benchmark datasets, even without an extensive hyperparameter tuning. As for the future work, extending ICT to interpolations not only at the input but at hidden representations (Verma et al., 2018) could improve the performance even further. Another direction for future work is to better understand the theoretical properties of interpolation-based regularizers in the SSL paradigm.

Chapter 7

Prologue to the third article

7.1. Article Details

Recurrent Independent Mechanisms. Anirudh Goyal, Alex Lamb, Jordan Hoffmann, Shagun Sodhani, Sergey Levine, Yoshua Bengio, Bernhard Schölkopf.

Recurrent Independent Mechanisms (RIMs) is a new recurrent architecture which follows the same input-output interface as the popular Gated Recurrent Unit (GRU) or vanilla recurrent network. In RIMs, the hidden state is separated into K modules (each which of which is called a RIM). Each RIM is associated with its own parameters, which can also be seen as a block-sparse parameterization of the recurrent network. On each steps, the RIMs all read from the input values using attention. The K_A RIMs with the highest attention scores are *activated*, whereas the other RIMs are deactivated, and their values remain unchanged until the next step. The activated RIMs update their hidden states using their own independent recurrent dynamics and then share attention with the other activated RIMs using attention. In practice the fraction of RIMs to be activated is a fairly flexible hyperparameter, with 25% to 75% of K generally working well (with higher values often being easier to optimize, and lower values having better generalization).

7.2. Personal Contribution

Anirudh Goyal had the initial idea of separating a recurrent network into multiple blocks which are competitive. Alex Lamb had the specific idea for how to do competition (through use of the input attention mechanism) and wrote the initial implementation, along with the first experiments showing systematic generalization on the copying task. I also did a significant amount of the writing and creation of the figure. Anirudh Goyal did the experiments on reinforcement learning. I asked Anirudh Goyal and he agreed that I could use this paper in my thesis.

7.3. Context

This paper proposes a new architecture for learning more structured recurrent neural networks. The separation of the hidden state into multiple mechanisms which only communicate via attention is related to the idea of transformers, but an important difference is that the mechanisms are recurrent, have separate parameters, and compete to access the input on each step.

Separation of recurrent networks into multiple components has been studied before. In the IndRNN (Li et al., 2018b), the recurrent weight matrix is applied as an elementwise multiplication, keeping the recurrent state well separate, and allowing interaction between units to occur only in the processing between layers. This shares the idea of having a structured recurrent weight with RIMs, yet this factorized transition separates all units, rather than separating blocks of units.

The selective activation idea in RIMs has also been explored before in deep learning. Neural Module Networks (Andreas et al., 2016) also explored the idea of having of having structured neural networks with dynamic activation patterns, but they used a parse tree to determine this activation pattern rather than learning it end-to-end. Learning independent causal mechanisms has also been explored using competition between experts in image generation (Parascandolo et al., 2018b).

7.4. Contributions

This technique shows very strong systematic generalization on tasks which follow its assumptions closely. In particular, it shows improvement where the underlying data distribution is generated from many processes which only interact sparingly, and where only a subset are active on each time step. One example of where this improves results is in a sequence learning task where a sequence must be memorized by the model, retained for a certain number of "waiting" steps, and then produced as output. If the number of waiting steps changes from the training phase to the evaluation phase, then many models such as LSTM and GRU degrade catastrophically. However the RIMs model is able to keep some of its modules fully deactivated during this waiting phase, which causes information to be fully preserved, and thus perfect evaluation performance is achievable on this copying task. It also helps significantly on reinforcement learning, where we showed that using RIMs as a drop-in replacement for GRU networks significantly improved results on Atari (Mnih et al., 2013).

7.5. Research Impact

This paper has a few nice follow-ups which takes its concept of modularity further. It also has a few applied follow-ups which exploit its ability to perform systematic generalization (Bagherzadeh & Bergler, 2021).

A number of research follow-ups have been produced in our lab. One involves disentangling the separation between making the parameters modular and making the hidden states modular. In RIMs, we assumed that each module (or RIM) has its own associated parameters as well as its own well-separated hidden states. However we may alternatively wish for two modules to share the same parameters while having well-separated hidden states. This is motivated by the notion of object files and schemata from the cognitive science literature, as well as classes and objects in object-oriented programming. In this work we used a gumbel-softmax (Jang et al., 2016) to control the selection of what parameters to use for each module on each step (Goyal et al., 2020). Thus it could learn to have all modules use separate parameters (recovering RIMs as a special case) or have some of the modules use the same parameters. Another follow-up project applies a factorization of the hidden state and parameters of the model in the context of the transformers architecture (Lamb et al., 2021b).

Chapter 8

Recurrent Independent Mechanisms

8.1. Independent Mechanisms

Physical processes in the world often have a modular structure which human cognition appears to exploit, with complexity emerging through combinations of simpler subsystems. Machine learning seeks to uncover and use regularities in the physical world. Although these regularities manifest themselves as statistical dependencies, they are ultimately due to dynamic processes governed by causal physical phenomena. These processes are mostly evolving independently and only interact sparsely. For instance, we can model the motion of two balls as separate independent mechanisms even though they are both gravitationally coupled to Earth as well as (weakly) to each other. Only occasionally will they strongly interact via collisions.

The notion of independent or autonomous mechanisms has been influential in the field of causal inference. A complex generative model, temporal or not, can be thought of as the composition of *independent* mechanisms or "causal" modules. In the causality community, this is often considered a prerequisite for being able to perform localized interventions upon variables determined by such models (Pearl, 2009). It has been argued that the individual modules tend to remain robust or invariant even as other modules change, e.g., in the case of distribution shift (Schölkopf et al., 2012; Peters et al., 2017). This independence is not between the random variables being processed but between the description or parametrization of the mechanisms: learning about one should not tell us anything about another, and adapting one should not require also adapting another. One may hypothesize that if a brain is able to solve multiple problems beyond a single i.i.d. (independent and identically distributed) task, they may exploit the existence of this kind of structure by learning independent mechanisms that can flexibly be reused, composed and re-purposed.

In the dynamic setting, we think of an overall system being assayed as composed of a number of fairly independent subsystems that evolve over time, responding to forces and interventions. An agent needs not devote equal attention to all subsystems at all times: only those aspects that significantly interact need to be considered jointly when deciding or planning (Bengio, 2017). Such sparse interactions can reduce the difficulty of learning since few interactions need to be considered at a time, reducing unnecessary interference when a subsystem is adapted. Models learned this way may better capture the compositional generative (or causal) structure of the world, and thus better generalize across tasks where a (small) subset of mechanisms change while most of them remain invariant (Simon, 1991; Peters et al., 2017).The central question motivating our work is how a gradient-based deep

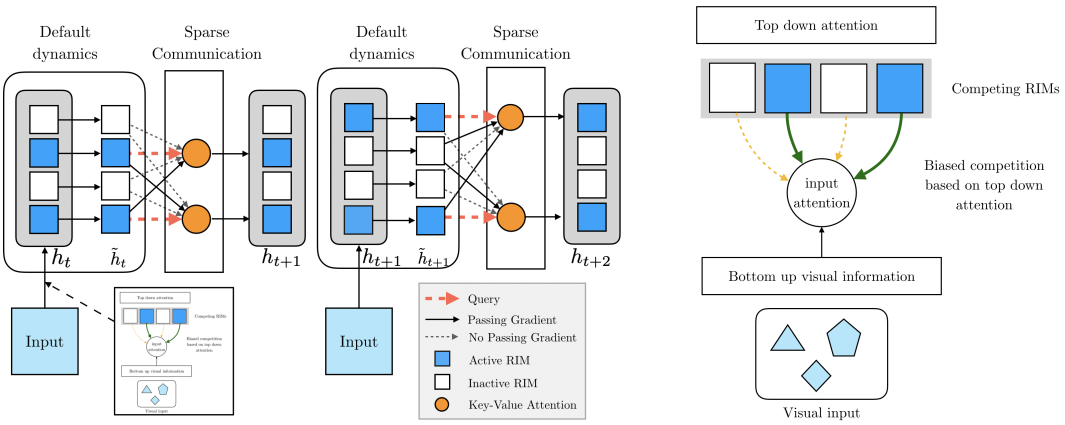


Fig. 1. Illustration of Recurrent Independent Mechanisms (RIMs). A single step under the proposed model occurs in four stages (left figure shows two steps). In the first stage, individual RIMs produce a query which is used to read from the current input. In the second stage, an attention based competition mechanism is used to select which RIMs to activate (right figure) based on encoded visual input (blue RIMs are active, based on an attention score, white RIMs remain inactive). In the third stage, individual activated RIMs follow their own default transition dynamics while non-activated RIMs remain unchanged. In the fourth stage, the RIMs sparsely communicate information between themselves, also using key-value attention.

learning approach can discover a representation of high-level variables which favour forming independent but sparsely interacting recurrent mechanisms in order to benefit from the modularity and independent mechanisms assumption.

Why do Models Succeed or Fail in Capturing Independent Mechanisms? While universal approximation theorems apply in the limit of large i.i.d. data sets, we are interested in the question of whether models can learn independent mechanisms from finite data in possibly changing environments, and how to implement suitable inductive biases. As the simplest case, we can consider training an RNN consisting of k completely independent mechanisms which operate on distinct time steps. How difficult would it be for an RNN (whether vanilla or LSTM or GRU) to correctly model that the true distribution has completely independent processes? For the hidden states to truly compartmentalize these different processes, a fraction $\frac{k-1}{k}$ of the connections would need to be set to exactly zero weight. This fraction approaches 100% as k approaches infinity. When sample complexity or out-of-distribution generalization matter, we argue that having an inductive bias which favors this form of modularity and dynamic recombination could be greatly advantageous, compared to static fully connected monolithic architectures.

Assumptions on the joint distribution of high level variables. The central question motivating our work is how a gradient-based deep learning approach can learn a representation of high level variables which favour learning independent but sparsely interacting recurrent mechanisms in order to benefit from such modularity assumption. The assumption about the joint distribution between the high-level variables is different from the assumption commonly found in many papers on disentangling factors of variation (Higgins et al., 2016; Burgess et al., 2018; Chen et al., 2018b), where the high level variables are assumed to be marginally independent of each other. We believe that these variables, (often named with

words in language), have highly structured dependencies supporting independent mechanisms assumption.

8.2. RIMs with Sparse Interactions

Our approach to modelling a dynamical system of interest divides the overall model into k small subsystems (or modules), each of which is recurrent in order to be able to capture the dynamics in the observed sequences. We refer to these subsystems as *Recurrent Independent Mechanisms (RIMs)*, where each RIM has distinct functions that are learned automatically from data[1]. We refer to RIM k at time step t as having vector-valued state $h_{t,k}$, where $t = 1, \ldots, T$. Each RIM has parameters θ_k, which are shared across all time steps.

At a high level (see Fig. 1), we want each RIM to have its own independent dynamics operating by default, and occasionally to interact with other relevant RIMs and selected elements of the encoded input. The total number of parameters can be kept small since RIMs can specialize on simple sub-problems, and operate on few key/value variables at a time selected using an attention mechanism, as suggested by the inductive bias from Bengio (2017). This specialization and modularization not only has computational and statistical advantages (Baum & Haussler, 1989), but also prevents individual RIMs from dominating the computation and thus facilitates factorizing the computation into easy to recombine but simpler elements. We expect this to lead to more robust systems than training one big homogeneous system (Schmidhuber, 2018). Moreover, modularity and the independent mechanisms hypothesis (Peters et al., 2017; Bengio et al., 2019) also has the desirable implication that a RIM should maintain its own independent functionality even as other RIMs are changed.

8.2.1. Key-Value Attention to Process Sets of Named Interchangeable Variables

Each RIM should be activated and updated when the input is relevant to it. We thus utilize competition to allocate representational and computational resources, using an attention mechanism which selects and then activates only a subset of the RIMs for each time step. As argued by (Parascandolo et al., 2018a), this tends to produce independence among learned mechanisms, provided the training data has been generated by a set of independent physical mechanisms. In contrast to (Parascandolo et al., 2018a), we use an *attention mechanism* for this purpose. The introduction of content-based soft-attention mechanisms (Bahdanau et al., 2014) has opened the door to neural networks which operate on *sets of typed interchangeable objects*. This idea has been remarkably successful and widely applied to most recent Transformer-style multi-head dot product self attention models (Vaswani et al., 2017; Santoro et al., 2018), achieving new state-of-the-art results in many tasks. Soft-attention uses the product of a *query* (or *read key*) represented as a matrix Q of dimensionality $N_r \times d$, with d the dimension of each key, with a set of N_o objects each associated with a *key* (or *write-key*) as a row in matrix K^T ($N_o \times d$), and after normalization with a softmax yields outputs in the convex hull of the *values* (or *write-values*) V_i (row i of

[1]Note that we are overloading the term *mechanism*, using it both for the mechanisms that make up the world's dynamics as well as for the computational modules that we learn to model those mechanisms. The distinction should be clear from context.

matrix V). The result is

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d}}\right)V,$$

where the softmax is applied to each row of its argument matrix, yielding a set of convex weights. As a result, one obtains a convex combination of the values in the rows of V. If the attention is focused on one element for a particular row (i.e., the softmax is saturated), this simply selects one of the objects and copies its value to row j of the result. Note that the d dimensions in the key can be split into *heads* which then have their own attention matrix and write values computed separately.

When the inputs and outputs of each RIM are a set of objects or entities (each associated with a key and value vector), the RIM processing becomes a generic object-processing machine which can operate on subsymbolic "variables" in a sense analogous to variables in a programming language: as interchangeable arguments of functions, albeit with a distributed representation both for they name or type and for their value. Because each object has a key embedding (which one can understand both as a name and as a type), the same RIM processing can be applied to any variable which fits an expected "distributed type" (specified by a query vector). Each attention head then corresponds to a typed argument of the function computed by the RIM. When the key of an object matches the query of head k, it can be used as the k-th input vector argument for the RIM. Whereas in regular neural networks (without attention) neurons operate on fixed variables (the neurons which are feeding them from the previous layer), the key-value attention mechanisms make it possible to select on the fly which variable instance (i.e. which entity or object) is going to be used as input for each of the arguments of the RIM dynamics, with a different set of query embeddings for each RIM head. These inputs can come from the external input or from the output of other RIMs.

8.2.2. Selective Activation of RIMs as a form of Top-Down Modulation

The proposed model learns to dynamically select those RIMs for which the current input is relevant. RIMs are triggered as a result of interaction between the current state of the RIM and input information coming from the environment. At each step, we select the top-k_A (out of k_T) RIMs in terms of their attention score for the real input. Intuitively, the RIMs must compete on each step to read from the input, and only the RIMs that win this competition will be able to read from the input and have their state updated. In our use of key-value attention, the queries come from the RIMs, while the keys and values come from the current input. This differs from the mechanics of Vaswani et al. (2017); Santoro et al. (2018), with the modification that the parameters of the attention mechanism itself are separate for each RIM rather than produced on the input side as in Transformers. The input attention for a particular RIM is described as follows.

The input x_t at time t is seen as a set of elements, structured as rows of a matrix. We first concatenate a row full of zerosSince the zeros contain no information, a RIM having high-attention weight on the zeros indicates that the input information on that time-step is not important for that RIM. This leads to:

$$X = \emptyset \oplus x_t. \tag{8.1}$$

As before, linear transformations are used to construct keys ($K = XW^e$, one per input element and for the null element), values ($V = XW^v$, again one per element), and queries ($Q = h_t W_k^q$, one per RIM attention head). W^v is a simple matrix mapping from an input element to the corresponding value vector for the weighted attention and W^e is similarly a weight matrix which maps the input to the keys. W_k^q is a per-RIM weight matrix which maps from the RIM's hidden state to its queries. $\oplus$ refers to the row-level concatenation operator. The attention thus is

$$A_k^{(in)} = \text{softmax}\left(\frac{h_t W_k^q (XW^e)^T}{\sqrt{d_e}}\right) XW^v, \text{ where } \theta_k^{(in)} = (W_k^q, W^e, W^v). \tag{8.2}$$

Based on the softmax values in (8.2), we select the top k_A RIMs (out of the total K RIMs) to be activated for each step, which have the least attention on the null input (and thus put the highest attention on the input), and we call this set $\mathcal{S}_t$. Since the queries depend on the state of the RIMs, this enables individual RIMs to attend only to the part of the input that is relevant for that particular RIM, thus enabling selective attention based on a *top-down attention* process (see. Fig 1). In practice, we use multiheaded attention, and multi-headed attention doesn't change the essential computation, but when we do use it for input-attention we compute RIM activation by averaging the attention scores over the heads.

For spatially structure input: All datasets we considered are temporal, yet there is a distinction between whether the input on each time step is highly structured (such as a video) or not (such as language modeling, where each step has a word or character). In the former case, we can get further improvements by making the activation of RIMs not just sparse across time but also sparse across the (spatial) structure. The input x_t at time t can be seen as an output of the encoder parameterized by a neural network (for ex. CNN in case of visual observations) i.e., $X = CNN(x_t)$. As before, linear transformations are used to construct position-specific input keys ($K = XW^e$), position-specific values ($V = XW^v$), and RIM specific queries ($Q = h_t W_k^q$, one per RIM attention head). The attention thus is

$$A_k^{(in)} = \text{softmax}\left(\frac{h_t W_k^q (XW^e)^T}{\sqrt{d_e}}\right) XW^v, \text{ where } \qquad \theta_k^{(in)} = (W_k^q, W^e, W^v). \tag{8.3}$$

In order for different RIMs to specialize on different spatial regions, we can use position-specific competition among the different RIMs. The contents of the attended positions are combined yielding a RIM-specific input. As before based on the softmax values in (8.3), we can select the top k_A RIMs (out of the total k_T RIMs) to be activated for each spatial position, which have the highest attention on that spatial position.

8.2.3. Independent RIM Dynamics

Now, consider the default transition dynamics which we apply for each RIM independently and during which no information passes between RIMs. We use $\tilde{h}$ for the hidden state after the independent dynamics are applied. The hidden states of RIMs which are not activated (we refer to the activated set as $\mathcal{S}_t$) remain unchanged, acting like untouched memory elements, i.e., $h_{t+1,k} = h_{t,k} \quad \forall k \notin \mathcal{S}_t$. Note that the gradient still flows through a RIM on a step where it is not activated. For the RIMs that are activated, we run a per-RIM independent transition dynamics. The form of this is somewhat flexible, but we opted to use either a GRU (Chung et al., 2015) or an LSTM (Hochreiter & Schmidhuber, 1997b). We generically refer to these independent transition dynamics as D_k, and we emphasize that each RIM has

its own separate parameters. Aside from being RIM-specific, the internal operation of the LSTM and GRU remain unchanged, and active RIMs are updated by

$$\tilde{h}_{t,k} = D_k(h_{t,k}) = LSTM(h_{t,k}, A_k^{(in)}; \theta_k^{(D)}) \quad \forall k \in \mathcal{S}_t$$

as a function of the attention mechanism $A_k^{(in)}$ applied on the current input, described in the previous sub-section.

8.2.4. Communication between RIMs

Although the RIMs operate independently by default, the attention mechanism allows sharing of information among the RIMs. Specifically, we allow the activated RIMs to read from all other RIMs (activated or not). The intuition behind this is that non-activated RIMs are not related to the current input, so their value needs not change. However they may still store contextual information relevant for activated RIMs later on. For this communication between RIMs, we use a residual connection as in (Santoro et al., 2018) to prevent vanishing or exploding gradients over long sequences. Using parameters $\theta_k^{(c)} = (\tilde{W}_k^q, \tilde{W}_k^e, \tilde{W}_k^v)$, we employ

$$Q_{t,k} = \tilde{W}_k^q \tilde{h}_{t,k}, \forall k \in \mathcal{S}_t \qquad K_{t,k} = \tilde{W}_k^e \tilde{h}_{t,k}, \forall k \qquad V_{t,k} = \tilde{W}_k^v \tilde{h}_{t,k}, \forall k$$

$$h_{t+1,k} = \text{softmax}\left(\frac{Q_{t,k}(K_{t,:})^T}{\sqrt{d_e}}\right) V_{t,:} + \tilde{h}_{t,k} \forall k \in \mathcal{S}_t.$$

We note that we can also consider sparsity in the communication attention such that a particular RIM only attends to sparse sub-set of other RIMs, and this sparsity is orthogonal to the kind used in input attention. In order to make the communication attention sparse, we can still use the same top-k attention.

Number of Parameters. RIMs can be used as a drop-in replacement for an LSTM/GRU layer. There is a subtlety that must be considered for successful integration. If the total size of the hidden state is kept the same, integrating RIMs drastically reduces the total number of recurrent parameters in the model (because of having a block-sparse structure). RIMs also adds new parameters to the model through the addition of the attention mechanisms although these are rather in small number.

Multiple Heads: Analogously to Vaswani et al. (2017); Santoro et al. (2018), we use multiple heads both for communication between RIMs as well as input attention (as in Sec 8.2.2) by producing different sets of queries, keys, and values to compute a linear transformation for each head (different heads have different parameters), and then applying the attention operator for each head separately in order to select conditioning inputs for the RIMs.

8.3. Related Work

Neural Turing Machine (NTM) and Relational Memory Core (RMC): the NTM (Graves et al., 2014) updates independent memory cells using an attention mechanism to perform targeted read and write operations. RIMs share a key idea with NTMs: that input information should only impact a sparse subset of the memory by default, while keeping most of the memory unaltered. RMC (Santoro et al., 2018) uses a multi-head attention mechanism to share information between multiple memory elements. We encourage the RIMs to remain separate as much as possible, whereas (Santoro et al., 2018) allow information between elements to flow on each step in an unsconstrained way. Instead, each RIM has its own default dynamics, while in RMC, all the processes interact with each other.

Separate Recurrent Models: EntNet (Henaff et al., 2016) and IndRNN (Li et al., 2018a) can be viewed as a set of separate recurrent models. In IndRNN, each recurrent unit has completely independent dynamics, whereas EntNet uses an independent gate for writing to each memory slot. RIMs use different recurrent models (with separate parameters), but we allow the RIMs to communicate with each other sparingly using an attention mechanism.

Modularity and Neural Networks: A network can be composed of several modules, each meant to perform a distinct function, and hence can be seen as a combination of experts (Jacobs et al., 1991; Bottou & Gallinari, 1991; Ronco et al., 1997; Reed & De Freitas, 2015; Andreas et al., 2016; Parascandolo et al., 2018a; Rosenbaum et al., 2017; Fernando et al., 2017; Shazeer et al., 2017; Kirsch et al., 2018; Rosenbaum et al., 2019) routing information through a gated activation of modules. These works generally assume that only a single expert is active at a particular time step. In the proposed method, multiple RIMs can be active, interact and share information.

Computation on demand: There are various architectures (El Hihi & Bengio, 1996; Koutnik et al., 2014; Chung et al., 2016; Neil et al., 2016; Jernite et al., 2016; Krueger et al., 2016) where parts of the RNN's hidden state are kept dormant at times. The major differences to our architecture are that (a) we modularize the dynamics of recurrent cells (using RIMs), and (b) we also control the inputs of each module (using transformer style attention), while many previous gating methods did not control the inputs of each module, but only whether they should be executed or not.

8.4. Experiments

The main goal of our experiments is to show that the use of RIMs improves generalization across changing environments and/or in modular tasks, and to explore how it does so. Our goal is not to outperform highly optimized baselines; rather, we want to show the versatility of our approach by applying it to a range of diverse tasks, focusing on tasks that involve a changing environment. We organize our results by the capabilities they illustrate: we address generalization based on temporal patterns, based on objects, and finally consider settings where both of these occur together.

8.4.1. RIMs improve generalization by specializing over temporal patterns

We first show that when RIMs are presented with sequences containing distinct and generally independent temporal patterns, they are able to specialize so that different RIMs are activated on different patterns. RIMs generalize well when we modify a subset of the patterns (especially those unrelated to the class label) while most recurrent models fail to generalize well to these variations.

Copying Task: First we turn our attention to the task of receiving a short sequence of characters, then receiving blank inputs for a large number of steps, and then being asked to reproduce the original sequence. We can think of this as consisting of two temporal patterns which are independent: one where the sequence is received and another "dormant" pattern where no input is provided. As an example of out-of-distribution generalization, we find that using RIMs, we can extend the length of this dormant phase from 50 during training to 200 during testing and retain perfect performance (Table 1), whereas baseline methods including LSTM, NTM, and RMC substantially degrade. In addition, we find that this result

Fig. 2. Visualizing Activation Patterns. For the copying task, one can see that the RIM activation pattern is distinct during the dormant part of the sequence in the middle (activated RIMs black, non-activated white). X-axis=time, Y-axis=RIMs activation bit.

Copying			**Train(50)**	**Test(200)**
k_{T}	k_{A}	h_{size}	CE	CE
RIMs 6	4	600	**0.00**	**0.00**
6	3	600	**0.00**	**0.00**
6	2	600	**0.00**	**0.00**
5	2	500	**0.00**	**0.00**
LSTM -	-	300	0.00	4.32
-	-	600	0.00	3.56
NTM -	-	-	0.00	2.54
RMC -	-	-	0.00	0.13
Transformers		-	0.00	0.54

Sequential MNIST			**16 x 16**	**19 x 19**	**24 x 24**
k_{T}	k_{A}	h_{size}	Accuracy	Accuracy	Accuracy
RIMs 6	6	600	85.5	56.2	30.9
6	5	600	88.3	43.1	22.1
6	4	600	**90.0**	**73.4**	**38.1**
LSTM -	-	300	86.8	42.3	25.2
-	-	600	84.5	52.2	21.9
EntNet	-	-	89.2	52.4	23.5
RMC -	-	-	89.58	54.23	27.75
DNC -	-	-	87.2	44.1	19.8
Transformers		-	**91.2**	51.6	22.9

Table 1. Performance on the copying task (left) and Sequential MNIST resolution generalization (right). While all of the methods are able to learn to copy for the length seen during training, the RIMs model generalizes to sequences longer than those seen during training whereas the LSTM, RMC, and NTM degrade much more. On sequential MNIST, both the proposed and the Baseline models were trained on 14x14 resolution but evaluated at different resolutions (averaged over 3 trials).

is robust to the number of RIMs used as well as to the number of RIMs activated per-step. Our ablation results show that all major components of the RIMs model are necessary to achieve this generalization. This is evidence that RIMs can specialize over distinct patterns in the data and improve generalization to settings where these patterns change.

Sequential MNIST Resolution Task: RIMs are motivated by the hypothesis that generalization performance can benefit from modules which only activate on relevant parts of the sequence. For further evidence that RIMs can achieve this out-of-distribution, we consider the task of classifying MNIST digits as sequences of pixels (Krueger et al., 2016) and assay generalization to images of resolutions different from those seen during training. Our intuition is that the RIMs model should have distinct subsets of the RIMs activated for pixels with the digit and empty pixels. RIMs should generalize better to higher resolutions by keeping RIMs dormant which store pixel information over empty regions of the image.

Results: Table 1 shows the result of the proposed model on the Sequential MNIST Resolution Task. If the train and test sequence lengths agree, both models achieve comparable test set performance. However, RIMs model is relatively robust to changing the sequence length (by changing the image resolution), whereas the LSTM performance degraded more severely. This can be seen as a more involved analogue of the copying task, as MNIST digits contain large empty regions. It is essential that the model be able to store information

and pass gradients through these regions. The RIMs outperform strong baselines such as Transformers, EntNet, RMC, and (DNC) (Graves et al., 2016).

8.4.2. RIMs learn to specialize over objects and generalize between them

We have shown that RIMs can specialize over temporal patterns. We now turn our attention to assaying whether RIMs can specialize to objects, and show improved generalization to cases where we add or remove objects at test time.

Bouncing Balls Environment: We consider a synthetic "bouncing balls" task in which multiple balls (of different masses and sizes) move using basic Newtonian physics (Van Steenkiste et al., 2018). What makes this task particularly suited to RIMs is that the balls move independently most of the time, except when they collide. During training, we predict the next frame at each time step using teacher forcing (Williams & Zipser, 1989a). We can then use this model to generate multi-step rollouts. As a preliminary experiment, we train on sequences of length 51 (the previous standard), using a binary cross entropy loss when predicting the next frame. We consider LSTMs as baselines. We then produce rollouts, finding that RIMs are better able to predict future motion (Figure 3).

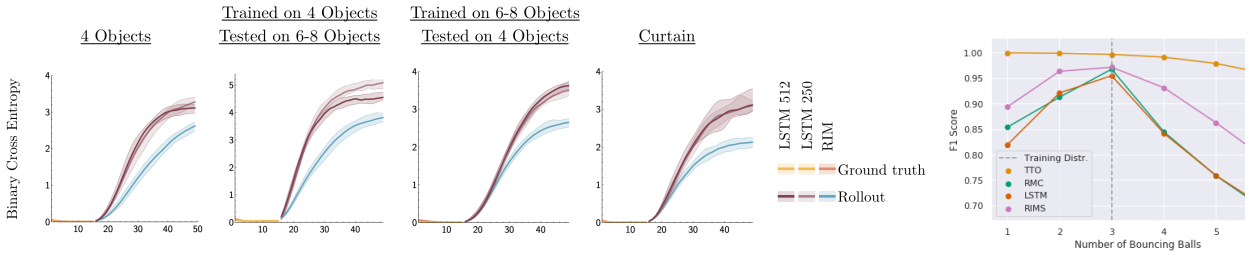


Fig. 3. Handling Novel Out-of-Distribution Variations. We study the performance of RIMs compared to an LSTM baseline (4 left plots). The first 15 frames of ground truth (yellow,orange) are fed in and then the system is rolled out for the next 35 time steps (blue,purple). During the rollout phase, RIMs perform better than the LSTMs in accurately predicting the dynamics of the balls as reflected by the lower Cross Entropy (CE) [blue for RIMs, purple for LSTMs]. Notice the substantially better out-of-distribution generalization of RIMs when testing on a number of objects different from the one seen during training. (2nd to 4th plot). **We also show (right plot) improved out-of-distribution generalization (F1 score) as compared to LSTM and RMC (Santoro et al., 2018) on another partial observation video prediction task.** X-axis = number of balls. For these experiments, the RIMs and baselines get an input image at each time step. Here, TTO refers to the time travelling oracle upper bound baseline, that does not model the dynamics, and has access to true dynamics.

We take this further by evaluating RIMs on environments where the test setup is different from the training setup. First we consider training with 4 balls and evaluating on an environment with 6-8 balls. Second, we consider training with 6-8 balls and evaluating with just 4 balls. Robustness in these settings requires a degree of invariance w.r.t. the number of balls.

In addition, we consider a task where we train on 4 balls and then evaluate on sequences where part the visual space is occluded by a "curtain." This allows us to assess the ability of

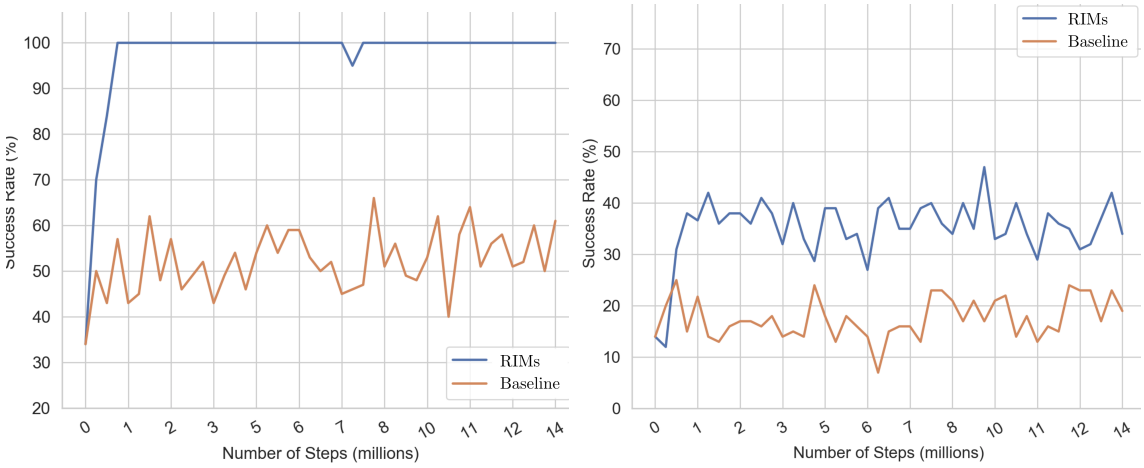


Fig. 4. Robustness to Novel Distractors:. Left: performance of the proposed method (blue) compared to an LSTM baseline (red) in solving the object picking task in the presence of distractors. Right: performance of proposed method and the baseline when novel distractors are added.

balls to be tracked (or remembered) through the occluding region. Our experimental results on these generalization tasks (Figure 3) show that RIMs substantially improve over an LSTM baseline. We found that increasing the capacity of the LSTM from 256 to 512 units did not substantially change the performance gap, suggesting that the improvement from RIMs is not primarily related to capacity.

Environment with Novel Distractors: We next consider an object-picking reinforcement learning task from BabyAI (Chevalier-Boisvert et al., 2018) in which an agent must retrieve a specific object in the presence of distractors. We use a partially observed formulation of the task, where the agent only sees a small number of squares ahead of it. These tasks are difficult to solve (Chevalier-Boisvert et al., 2018) with standard RL algorithms, due to (1) the partial observability of the environment and (2) the sparsity of the reward, given that the agent receives a reward only after reaching the goal. During evaluation, we introduce new distractors to the environment which were not observed during training.

Figure 4 shows that RIMs outperform LSTMs on this task (details in appendix). When evaluating with known distractors, the RIM model achieves perfect performance while the LSTM struggles. When evaluating in an environment with novel unseen distractors the RIM doesn't achieve perfect performance but strongly outperforms the LSTM. An LSTM with a single memory flow may struggle to keep the distracting elements separate from elements which are necessary for the task, while the RIMs model uses attention to control which RIMs receive information at each step as well as what information they receive (as a function of their hidden state). This "top-down" attention results in a diminished representation of the distractor, not only enhancing the target visual information, but also suppressing irrelevant information.

8.4.3. RIMs improve generalization in complex environments

We have investigated how RIMs use specialization to improve generalization to changing important factors of variation in the data. While these improvements have often been striking, it raises a question: what factors of variation should be changed between training

and evaluation? One setting where factors of variation change naturally is in reinforcement learning, as the data received from an environment changes as the agent learns and improves. We conjecture that when applied to reinforcement learning, an agent using RIMs may be able to learn faster as its specialization leads to improved generalization to previously unseen aspects of the environment. To investigate this we use an RL agent trained using Proximal Policy Optimization (PPO) (Schulman et al., 2017) with a recurrent network producing the policy. We employ an LSTM as a baseline, and compare results to the RIMs architecture. This was a simple drop-in replacement and did not require changing any of the hyperparameters for PPO. We experiment on the whole suite of Atari games and find that simply replacing the LSTM with RIMs greatly improves performance (Figure 5).

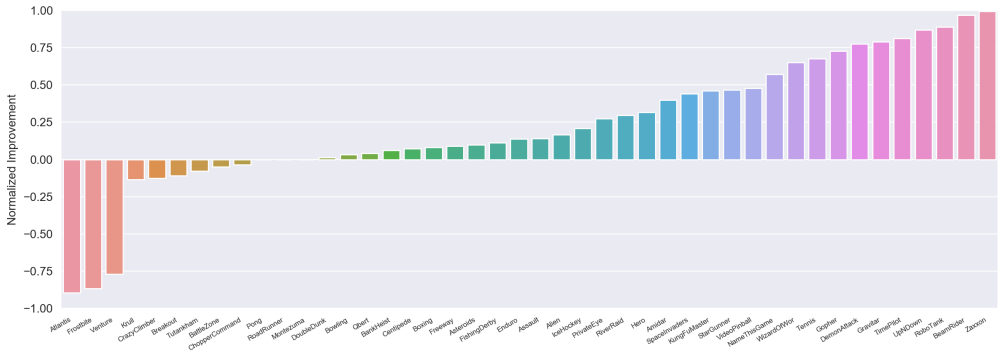


Fig. 5. RIMs-PPO relative score improvement over LSTM-PPO baseline (Schulman et al., 2017) across all Atari games averaged over 3 trials per game. In both cases, PPO was used with the exact same settings, and the only change is the choice of recurrent architecture.

There is also an intriguing connection between the selective activation in RIMs and the concept of affordances from cognitive psychology (Gibson, 1977; Cisek & Kalaska, 2010). To perform well in environments with a dynamic combination of risks and opportunities, an agent should be ready to adapt immediately, executing actions which are at least partially prepared. This suggests agents should process sensory information in a contextual manner, building representations of potential actions that the environment currently affords. For instance, in Demon Attack, one of the games where RIMs exhibit strong performance gains, the agent must quickly choose between targeting distant aliens to maximize points and avoiding fire from close-by aliens to avoid destruction (indeed both types of aliens are always present, but which is relevant depends on the player's position). We hypothesize that in cases like this, selective activation of RIMs allows the agent to rapidly adapt its information processing to the types of actions relevant to the current context.

8.4.4. Discussion and Ablations

Sparse Activation is necessary, but works for a wide range of hyperparameters: On the copying task, we tried a wide variety of sparsity levels for different numbers of RIMs, and found that using a sparsity level between 30% to 70% performed optimally, suggesting that the sparsity hyperparameter is fairly flexible. On Atari we found that using $k_A = 5$ slightly improved over results compared with $k_A = 4$, but both had similar performance across the vast majority of games.

Input-attention is necessary: We study the scenario where we remove the input attention process (i.e the top-down competition between different RIMs) but still allow the

RIMs ot communicate with attention. We found that this degraded results substantially on Atari but still outperformed the LSTM baseline.

Communication between RIMs improves performance: For copying and sequential MNIST, we performed an ablation where we remove the communication between RIMs and varied the number of RIMs and the number of activated RIMs. We found that the communication between RIMs is essential for good performance.

8.5. Conclusion

Many systems of interest comprise multiple dynamical processes that operate relatively independently and only occasionally have meaningful interactions. Despite this, most machine learning models employ the opposite inductive bias, i.e., that all processes interact. This can lead to poor generalization and lack of robustness to changing task distributions. We have proposed a new architecture, Recurrent Independent Mechanisms (RIMs), in which we learn multiple recurrent modules that are independent by default, but interact sparingly. For the purposes of this paper, we note that the notion of RIMs is not limited to the particular architecture employed here. The latter is used as a vehicle to assay and validate our overall hypothesis, but better architectures for the RIMs model can likely be found.

Chapter 9

Prologue to the fourth article

9.1. Article Details

Neural Function Modules with Sparse Arguments: A Dynamic Approach to Integrating Information across Layers. Alex Lamb, Anirudh Goyal, Agnieszka Słowik, Michael Mozer, Philippe Beaudoin, Yoshua Bengio. AISTATS 2021.

In Neural Function Modules (NFM) we showed that a network can benefit from using attention to look over the outputs of all previously computed layers. This makes the information flow through the network sparser and more dynamic, as each layer can focus on its own specialized processing. To make the set of layers that the network can attend over more interesting, we explore a multi-pass convolutional neural network, where we run a network for multiple passes and allow the network to attend over all previously computed layers from all of the prior passes.

9.2. Personal Contribution

Anirudh Goyal had the initial idea of having a neural network where each layer can attend to all previously computed layers. Alex Lamb produced the initial implementation of the module for doing this, and the initial experiments on the supervised learning datasets as well as the multiple mnist-digit experiments. Alex Lamb created the novel-occlusion dataset, ran the hyperparameter-effect analysis, and wrote most of the paper. Anirudh Goyal produced the GAN experiments and the Atari experiments. Aga Slowik produced the experiments on the relational reasoning dataset.

9.3. Context

Two ideas serve as the foundations and motivation for the NFM project. Residual networks (He et al., 2016a) introduced a type of additive skip connections into deep neural networks which allow for training of very deep neural networks. These skip connections allow each layer to specialize on learning a small additive change to the content of the hidden representation rather than forcing each layer to produce the hidden representation from scratch. The Densenet paper (Huang et al., 2017) explored an alternative type of skip connection in which layers are concatenated to the input of later layers in the network. The attention mechanism (Bahdanau et al., 2014; Vaswani et al., 2017) is a special type of layer which allows for information to be retrieved in a selective fashion. An outer-product between query and

key vectors is normalized to give attention-weights defining how each query attends over a set of values. When the attention-weights are fully saturated, a query can put 100% of its attention weight onto a single value, making the flow of information sparse. Sparse Attention Backtracking (Ke et al., 2018) showed that such a sparsity assumption can also be imposed explicitly in the architecture. Attention has been widely used to share information between semantically meaningful positions as well as between slots (Vaswani et al., 2017; Goyal et al., 2019).

9.4. Contributions

This paper explored using attention to share information between layers, giving networks a way of making layers more specialized. We achieved improved classification accuracy on CIFAR-10, CIFAR-100, Tiny-Imagenet, and Imagenet (Krizhevsky, 2009; Krizhevsky et al., 2012). We also integrated NFM into the generator of Generative Adversarial Networks (Goodfellow et al., 2014). Compared to a state-of-the-art InfoMax-GAN baseline (Kwot Sin Lee & Cheung, 2019), we achieved significant improvement in Inception Score (Salimans et al., 2016) and Frechet Inception Distance (Heusel et al., 2017a). We also constructed a novel stacked-MNIST task (LeCun & Cortes, 2005) in which multiple MNIST digits are super-imposed and the model is tasked with classifying which digits are present. This is framed as a multi-label binary classification task. We vary the number of digits present between training and evaluation. For example, during training we may have either three or five digits superimposed, but during evaluation four digits are superimposed. We found that NFM substantially improves systematic generalization this task. We also explored the Sort-of-Clevr relational reasoning benchmark (Santoro et al., 2018) in which simple questions must be answered from images (e.g. is the red ball to the left of the blue square). We achieved significant improvement using NFM on this task when compared to a convolutional neural network baseline. We also integrated NFM into the convolutional encoder in the Rainbow-DQN (Dabney et al., 2018; Hessel et al., 2018) and found that it improved results on four of five selected Atari games (Vezhnevets et al., 2016).

9.5. Research Impact

This paper showed significant improvements on relational reasoning tasks and decent improvements on standard image classification benchmarks. It also points to a way of sharing information across multiple layers that could be integrated into many different architectures, such as transformers or RNNs. The Head2Toe technique found that learning which layer's features to use for fine-tuning improved transfer learning performance (Evci et al., 2022). The Shared Workspace architecture (Goyal et al., 2021) writes to and reads from the same workspace on successive layers (when applied to transformers). In this way, it is related to how information is flexibly persisted across layers in NFM.

Chapter 10

Neural Function Modules

10.1. Introduction

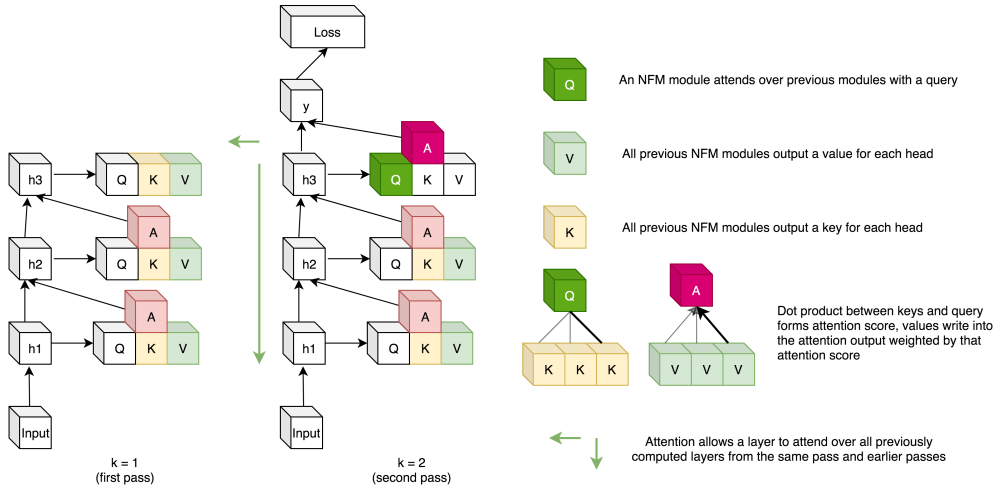


Fig. 1. An illustration of Neural Function Modules (NFM) where a network is ran over the input twice ($\mathcal{K} = 2$). A layer where NFM is applied sparsely attends over the set of previously computed layers, allowing better specialization. The attention over the layers from the previous pass acts as a form of top-down feedback.

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 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 10.4). 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.

10.2. 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

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., 2018b)
Combining Top-Down and Bottom-Up Feedback	Deep Boltzmann Machines (Salakhutdinov & Hinton, 2009)

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., 2017b; 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, 2020). 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 & Perona, 2017). Recently (Hu et al., 2018; Woo et al., 2018; Wang et al., 2018b; Chen et al., 2018c; Veit & 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 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.

10.3. 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.

10.3.1. 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.

10.3.2. 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 2.

10.3.2.1. 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; 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 2. 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

Algorithm 2 Neural Functional Module (NFM)

Input: An input x. A number of passes $\mathcal{K}$. A neural network with N modules for all $k \in \{1...\mathcal{K}\}$: $f_{\theta_k}^{(1)}$, $f_{\theta_k}^{(2)}$, $f_{\theta_k}^{(3)}$, ..., $f_{\theta_k}^{(N)}$
$\mathcal{M} := EmptyList$
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
    $A = Attention_{\tilde{\theta}_k}^{(i)}(K = \tilde{\mathcal{M}}, V = \tilde{\mathcal{M}}, Q = h^{(i-1)})$
    $h^{(i)} := f_{\theta_k}^{(i)}(residual = h^{(i-1)}, input = A)$
    $\mathcal{M}$.append($h^{(i)}$)
  end for
end for

the values are given dimension d_v. It is specific to the layer index i and the parameters for attention for the layer i: $(W_q, W_k, W_v, W_{o_1}, W_{o_2}, \gamma) = \tilde{\theta}_i^{(A)}$. 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., 2018a).
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.

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

If $\text{len}(h_i) = \text{len}(h_j)$, then the two modules are of the same scale, and no rescaling is performed. If $\text{len}(h_i) > \text{len}(h_j)$, 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 $\text{len}(h_i) < \text{len}(h_j)$, 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 $\frac{\text{len}(h_j)}{\text{len}(h_i)}$ as

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

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

separate positions for the attention. Thus when attending over a higher resolution module, NFM can pick specific positions to attend over.

10.3.3. 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.

10.4. Experiments

Our experiments have the following goals:

- 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.

10.4.1. 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

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., 2018b), SSGAN (Chen et al., 2019), and InfoMax-GAN (Kwot Sin Lee & 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

generating a part with particular properties. For example, if we want to draw a cat's leg, it should generally look exactly like the cat's other leg. Based on this intuition we integrated NFM into InfoMax GAN (Kwot Sin Lee & 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 & Town, 2020).

We used the (Lee & 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., 2017b) 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).

10.4.2. 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.

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 on number of digits: (1,3)		
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 on number of digits: (1,5)		
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

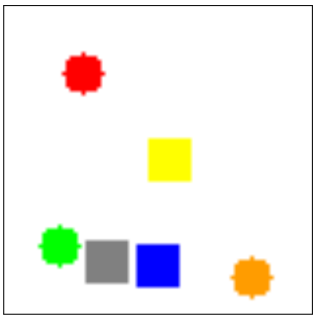
10.4.3. Relational Reasoning

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., 2017a) 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 generation follow description in the paper which introduced Sort-of-CLEVR (Santoro et al., 2017a). Similarly as in Section 10.4.2, 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%). Relational questions related to 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

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



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

Fig. 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

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., 2017a). 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.

10.4.4. 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., 2018c). 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 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 & Taylor, 2017).

10.4.5. 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).

10.4.6. Transformer-NFM for Language Modeling

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

10.4.7. 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).

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

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. We also looked at the relation between the number of model parameters, test accuracy and adding/removing NFM in an image classification task.

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 ± 3.5 whereas NFM achieves a much higher accuracy of 74.95 ± 1.58. When we doubled the number of channels and the number of layers the accuracy slightly improved to $\mathbf{76.5 \pm 1.1}$ (with NFM) and 69.21 ± 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 10.3.2.2) 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%

10.5. 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.

Chapter 11

Conclusion

The articles presented as a part of this thesis explore how latent data augmentation and modular structure can contribute to improved generalization in deep learning.

(1) **Training with interpolated hidden states** (Chapter 4): In Verma et al. (2018), we present a way of training deep networks on interpolations of hidden states in random layers, which we show leads to more compressed hidden representations in theory and in practice, as well as improved generalization in deep learning.

(2) **Interpolation Consistency Training for Semi-Supervised Learning** (Chapter 6): In Verma et al. (2019a), we present a new semi-supervised learning algorithm which operates in the consistency training framework, and trains on interpolations between different examples. The algorithm significantly improved over previous consistency-based semi-supervised learning algorithms which used Gaussian noise in the input space instead of interpolations.

(3) **Recurrent Independent Mechanisms** (Chapter 8): Deep neural networks achieve excellent ability to fit a fixed training set, yet often generalize poorly when the training set and testing set differ systematically. An assumption that can help to improve systematic generalization is the idea that the system under consideration consists of independent mechanisms, which retain a functional role as unrelated parts of the world are changed. We proposed a Recurrent Independent Mechanisms model (Goyal et al., 2019) in which the hidden state is divided into multiple mechanisms with well-separated states and parameters, and which communicate selectively via attention and compete to active. This greatly improves generalization on benchmarks where we vary some of the patterns between train and test time and also improves reinforcement learning performance.

(4) **Neural Function Modules** (Chapter 10): Typical layers in deep neural networks process the inputs in order, starting from the first layer and ending with the last layer. This may adversely affect the model's capacity, as it forces the layers to relate to each other in a fixed way, rather than dynamically selecting their inputs from the previously computed layers. To address this challenge, we proposed Neural Functional Modules Lamb et al. (2021a) which uses attention at every layer to attend over all the outputs from previously computed layers. This greatly improved performance on reasoning tasks and moderately improved performance on classification tasks.

References

Alessandro Achille and Stefano Soatto. Information dropout: Learning optimal representations through noisy computation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2018.

Alexander A. Alemi, Ian Fischer, Joshua V. Dillon, and Kevin Murphy. Deep variational information bottleneck. In *International Conference on Learning Representations*, 2017.

Guozhong An. The Effects of Adding Noise During Backpropagation Training on a Generalization Performance. *Neural Computation*, 8(3):643–674, 04 1996. ISSN 0899-7667. doi: 10.1162/neco.1996.8.3.643. URL https://doi.org/10.1162/neco.1996.8.3.643.

Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Dan Klein. Neural module networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 39–48, 2016.

Martín Arjovsky, Amar Shah, and Yoshua Bengio. Unitary evolution recurrent neural networks. *CoRR*, abs/1511.06464, 2015. URL http://arxiv.org/abs/1511.06464.

Anish Athalye, Nicholas Carlini, and David Wagner. Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. In Jennifer Dy and Andreas Krause (eds.), *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 274–283, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR. URL http://proceedings.mlr.press/v80/athalye18a.html.

Ben Athiwaratkun, Marc Finzi, Pavel Izmailov, and Andrew Gordon Wilson. There are many consistent explanations of unlabeled data: Why you should average. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=rkgKBhA5Y7.

Parsa Bagherzadeh and Sabine Bergler. Multi-input recurrent independent mechanisms for leveraging knowledge sources: Case studies on sentiment analysis and health text mining. In *Proceedings of Deep Learning Inside Out (DeeLIO): The 2nd Workshop on Knowledge Extraction and Integration for Deep Learning Architectures*, pp. 108–118, Online, June 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.deelio-1.11. URL https://aclanthology.org/2021.deelio-1.11.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

Dzmitry Bahdanau, Shikhar Murty, Michael Noukhovitch, Thien Huu Nguyen, Harm de Vries, and Aaron Courville. Systematic generalization: what is required and can it be learned? *arXiv preprint arXiv:1811.12889*, 2018.

Peter Bartlett and John Shawe-taylor. Generalization performance of support vector machines and other pattern classifiers, 1998.

Sergey Bartunov, Adam Santoro, Blake A. Richards, Geoffrey E. Hinton, and Timothy Lillicrap. Assessing the scalability of biologically-motivated deep learning algorithms and architectures. *submitted to ICLR'2018*, 2018.

Eric B Baum and David Haussler. What size net gives valid generalization? In *Advances in neural information processing systems*, pp. 81–90, 1989.

Christopher Beckham, Sina Honari, Alex Lamb, Vikas Verma, Farnoosh Ghadiri, R Devon Hjelm, and Christopher Pal. Adversarial mixup resynthesizers. *arXiv preprint arXiv:1903.02709*, 2019.

Ishmael Belghazi, Sai Rajeswar, Aristide Baratin, R. Devon Hjelm, and Aaron C. Courville. MINE: mutual information neural estimation. *CoRR*, abs/1801.04062, 2018. URL http://arxiv.org/abs/1801.04062.

Shai Ben-David, John Blitzer, Koby Crammer, Alex Kulesza, Fernando Pereira, and Jennifer Wortman Vaughan. A theory of learning from different domains. *Machine learning*, 79(1-2):151–175, 2010.

Yoshua Bengio. Deep learning of representations: Looking forward. In *International Conference on Statistical Language and Speech Processing*, pp. 1–37. Springer, 2013.

Yoshua Bengio. The consciousness prior. *arXiv preprint arXiv:1709.08568*, 2017.

Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.

Yoshua Bengio, Tristan Deleu, Nasim Rahaman, Rosemary Ke, Sébastien Lachapelle, Olexa Bilaniuk, Anirudh Goyal, and Christopher Pal. A meta-transfer objective for learning to disentangle causal mechanisms. *arXiv preprint arXiv:1901.10912*, 2019.

David Berthelot, Nicholas Carlini, Ekin D Cubuk, Alex Kurakin, Kihyuk Sohn, Han Zhang, and Colin Raffel. Remixmatch: Semi-supervised learning with distribution alignment and augmentation anchoring. *arXiv preprint arXiv:1911.09785*, 2019a.

David Berthelot, Nicholas Carlini, Ian Goodfellow, Nicolas Papernot, Avital Oliver, and Colin Raffel. Mixmatch: A holistic approach to semi-supervised learning. *arXiv preprint arXiv:1905.02249*, 2019b.

David Berthelot*, Colin Raffel*, Aurko Roy, and Ian Goodfellow. Understanding and improving interpolation in autoencoders via an adversarial regularizer. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=S1fQSiCcYm.

Léon Bottou and Patrick Gallinari. A framework for the cooperation of learning algorithms. In *Advances in neural information processing systems*, pp. 781–788, 1991.

Christopher P Burgess, Irina Higgins, Arka Pal, Loic Matthey, Nick Watters, Guillaume Desjardins, and Alexander Lerchner. Understanding disentangling in *beta*-vae. *arXiv preprint arXiv:1804.03599*, 2018.

O. Chapelle, B. Scholkopf, and A. Zien, Eds. Semi-supervised learning (chapelle, o. et al., eds.; 2006) [book reviews]. *IEEE Transactions on Neural Networks*, 20(3):542–542, 2009. doi: 10.1109/TNN.2009.2015974.

Olivier Chapelle, Jason Weston, Léon Bottou, and Vladimir Vapnik. Vicinal risk minimization. In T. K. Leen, T. G. Dietterich, and V. Tresp (eds.), *Advances in Neural Information Processing Systems 13*, pp. 416–422. MIT Press, 2001. URL http://papers.nips.cc/paper/1876-vicinal-risk-minimization.pdf.

Olivier Chapelle, Bernhard Schlkopf, and Alexander Zien. *Semi-Supervised Learning*. The MIT Press, 1st edition, 2010. ISBN 0262514125, 9780262514125.

Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (eds.), *Advances in Neural Information Processing Systems 31*, pp. 6571–6583. Curran Associates, Inc., 2018a. URL http://papers.nips.cc/paper/7892-neural-ordinary-differential-equations.pdf.

Ricky TQ Chen, Xuechen Li, Roger B Grosse, and David K Duvenaud. Isolating sources of disentanglement in variational autoencoders. In *Advances in Neural Information Processing Systems*, pp. 2610–2620, 2018b.

Ting Chen, Xiaohua Zhai, Marvin Ritter, Mario Lucic, and Neil Houlsby. Self-supervised gans via auxiliary rotation loss. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 12154–12163, 2019.

Zhourong Chen, Yang Li, Samy Bengio, and Si Si. Gaternet: Dynamic filter selection in convolutional neural network via a dedicated global gating network. *arXiv preprint arXiv:1811.11205*, 2018c.

Maxime Chevalier-Boisvert, Dzmitry Bahdanau, Salem Lahlou, Lucas Willems, Chitwan Saharia, Thien Huu Nguyen, and Yoshua Bengio. Babyai: First steps towards grounded language learning with a human in the loop. *arXiv preprint arXiv:1810.08272*, 2018.

Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. Generating long sequences with sparse transformers, 2019.

Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

Ching-Yao Chuang and Youssef Mroueh. Fair mixup: Fairness via interpolation. *arXiv preprint arXiv:2103.06503*, 2021.

Junyoung Chung, Kyle Kastner, Laurent Dinh, Kratarth Goel, Aaron C Courville, and Yoshua Bengio. A recurrent latent variable model for sequential data. In *Advances in neural information processing systems*, pp. 2980–2988, 2015.

Junyoung Chung, Sungjin Ahn, and Yoshua Bengio. Hierarchical multiscale recurrent neural networks. *arXiv preprint arXiv:1609.01704*, 2016.

Paul Cisek and John F Kalaska. Neural mechanisms for interacting with a world full of action choices. *Annual review of neuroscience*, 33:269–298, 2010.

Tarin Clanuwat, Mikel Bober-Irizar, Asanobu Kitamoto, Alex Lamb, Kazuaki Yamamoto, and David Ha. Deep learning for classical japanese literature. *arXiv preprint arXiv:1812.01718*, 2018.

George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

Will Dabney, Georg Ostrovski, David Silver, and Rémi Munos. Implicit quantile networks for distributional reinforcement learning. *arXiv preprint arXiv:1806.06923*, 2018.

Terrance Devries and Graham W. Taylor. Improved regularization of convolutional neural networks with cutout. *CoRR*, abs/1708.04552, 2017. URL http://arxiv.org/abs/1708.04552.

Terrance DeVries and Graham W Taylor. Improved regularization of convolutional neural networks with cutout. *arXiv preprint arXiv:1708.04552*, 2017.

Vincent Dumoulin and Francesco Visin. A guide to convolution arithmetic for deep learning. *arXiv preprint arXiv:1603.07285*, 2016.

Vincent Dumoulin, Ishmael Belghazi, Ben Poole, Olivier Mastropietro, Alex Lamb, Martin Arjovsky, and Aaron Courville. Adversarially learned inference. *arXiv preprint arXiv:1606.00704*, 2016a.

Vincent Dumoulin, Ishmael Belghazi, Ben Poole, Olivier Mastropietro, Alex Lamb, Martin Arjovsky, and Aaron Courville. Adversarially learned inference. *arXiv preprint arXiv:1606.00704*, 2016b.

Salah El Hihi and Yoshua Bengio. Hierarchical recurrent neural networks for long-term dependencies. In *Advances in neural information processing systems*, pp. 493–499, 1996.

Utku Evci, Vincent Dumoulin, Hugo Larochelle, and Michael C Mozer. Head2toe: Utilizing intermediate representations for better transfer learning. *arXiv preprint arXiv:2201.03529*, 2022.

Mojtaba Faramarzi, Mohammad Amini, Akilesh Badrinaaraayanan, Vikas Verma, and Sarath Chandar. Patchup: A regularization technique for convolutional neural networks. *arXiv preprint arXiv:2006.07794*, 2020.

Chrisantha Fernando, Dylan Banarse, Charles Blundell, Yori Zwols, David Ha, Andrei A Rusu, Alexander Pritzel, and Daan Wierstra. Pathnet: Evolution channels gradient descent in super neural networks. *arXiv preprint arXiv:1701.08734*, 2017.

Amber M Fyall, Yasmine El-Shamayleh, Hannah Choi, Eric Shea-Brown, and Anitha Pasupathy. Dynamic representation of partially occluded objects in primate prefrontal and visual cortex. *Elife*, 6:e25784, 2017.

James J Gibson. The theory of affordances. *Hilldale, USA*, 1(2), 1977.

I. J. Goodfellow, J. Shlens, and C. Szegedy. Explaining and Harnessing Adversarial Examples. In *International Conference on Learning Representations*, 2015.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pp. 2672–2680, 2014.

Anirudh Goyal, Riashat Islam, DJ Strouse, Zafarali Ahmed, Hugo Larochelle, Matthew Botvinick, Sergey Levine, and Yoshua Bengio. Transfer and exploration via the information bottleneck. 2018.

Anirudh Goyal, Alex Lamb, Jordan Hoffmann, Shagun Sodhani, Sergey Levine, Yoshua Bengio, and Bernhard Schölkopf. Recurrent independent mechanisms. *arXiv preprint arXiv:1909.10893*, 2019.

Anirudh Goyal, Alex Lamb, Phanideep Gampa, Philippe Beaudoin, Sergey Levine, Charles Blundell, Yoshua Bengio, and Michael Mozer. Object files and schemata: Factorizing declarative and procedural knowledge in dynamical systems. *arXiv preprint arXiv:2006.16225*, 2020.

Anirudh Goyal, Aniket Didolkar, Alex Lamb, Kartikeya Badola, Nan Rosemary Ke, Nasim Rahaman, Jonathan Binas, Charles Blundell, Michael Mozer, and Yoshua Bengio. Coordination among neural modules through a shared global workspace. *arXiv preprint arXiv:2103.01197*, 2021.

Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines. *arXiv preprint arXiv:1410.5401*, 2014.

Alex Graves, Greg Wayne, Malcolm Reynolds, Tim Harley, Ivo Danihelka, Agnieszka Grabska-Barwińska, Sergio Gómez Colmenarejo, Edward Grefenstette, Tiago Ramalho, John Agapiou, et al. Hybrid computing using a neural network with dynamic external memory. *Nature*, 538(7626):471, 2016.

Klaus Greff, Raphaël Lopez Kaufmann, Rishab Kabra, Nick Watters, Chris Burgess, Daniel Zoran, Loic Matthey, Matthew Botvinick, and Alexander Lerchner. Multi-object representation learning with iterative variational inference. *arXiv preprint arXiv:1903.00450*, 2019.

H. Guo, Y. Mao, and R. Zhang. MixUp as Locally Linear Out-Of-Manifold Regularization. *ArXiv e-prints*, September 2018a.

H. Guo, Y. Mao, and R. Zhang. Aggregated Learning: A Vector Quantization Approach to Learning with Neural Networks. *ArXiv e-prints*, July 2018b.

Hongyu Guo, Yongyi Mao, and Richong Zhang. MixUp as Locally Linear Out-Of-Manifold Regularization. *ArXiv e-prints*, 2016. URL https://arxiv.org/abs/1809.02499.

Xiaoqing Guo, Chen Yang, Yajie Liu, and Yixuan Yuan. Learn to threshold: Thresholdnet with confidence-guided manifold mixup for polyp segmentation. *IEEE Transactions on Medical Imaging*, 40(4):1134–1146, 2021. doi: 10.1109/TMI.2020.3046843.

Johan Hstad. *Computational Limitations of Small-Depth Circuits*. Mit Press, 1987.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016a.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *ECCV*, 2016b.

Mikael Henaff, Jason Weston, Arthur Szlam, Antoine Bordes, and Yann LeCun. Tracking the world state with recurrent entity networks. *arXiv preprint arXiv:1612.03969*, 2016.

Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017a. URL https://proceedings.neurips.cc/paper/2017/file/8a1d694707eb0fefe65871369074926d-Paper.pdf.

Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. In *Advances in neural information processing systems*, pp. 6626–6637, 2017b.

Irina Higgins, Loic Matthey, Arka Pal, Christopher Burgess, Xavier Glorot, Matthew Botvinick, Shakir Mohamed, and Alexander Lerchner. beta-vae: Learning basic visual concepts with a constrained variational framework. 2016.

Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *CoRR*, abs/1207.0580, 2012. URL http://arxiv.org/abs/1207.0580.

Sepp Hochreiter. Untersuchungen zu dynamischen neuronalen netzen. *Diploma, Technische Universität München*, 91(1), 1991.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9 (8):1735–1780, November 1997a. ISSN 0899-7667. doi: 10.1162/neco.1997.9.8.1735. URL http://dx.doi.org/10.1162/neco.1997.9.8.1735.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9 (8):1735–1780, 1997b.

Jie Hu, Li Shen, Samuel Albanie, Gang Sun, and Andrea Vedaldi. Gather-excite: Exploiting feature context in convolutional neural networks. In *Advances in Neural Information Processing Systems*, pp. 9401–9411, 2018.

Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4700–4708, 2017.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pp. 448–456. PMLR, 2015a.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, 2015b.

Robert A Jacobs, Michael I Jordan, Steven J Nowlan, Geoffrey E Hinton, et al. Adaptive mixtures of local experts. *Neural computation*, 3(1):79–87, 1991.

Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.

Stanisław Jastrzębski, Devansh Arpit, Nicolas Ballas, Vikas Verma, Tong Che, and Yoshua Bengio. Residual connections encourage iterative inference. *arXiv preprint arXiv:1710.04773*, 2017.

Jisoo Jeong, Vikas Verma, Minsung Hyun, Juho Kannala, and Nojun Kwak. Interpolation-based semi-supervised learning for object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 11602–11611, 2021.

Yacine Jernite, Edouard Grave, Armand Joulin, and Tomas Mikolov. Variable computation in recurrent neural networks. *arXiv preprint arXiv:1611.06188*, 2016.

Nan Rosemary Ke, Anirudh Goyal ALIAS PARTH GOYAL, Olexa Bilaniuk, Jonathan Binas, Michael C Mozer, Chris Pal, and Yoshua Bengio. Sparse attentive backtracking: Temporal credit assignment through reminding. In *Advances in neural information processing systems*, pp. 7640–7651, 2018.

Seonhoon Kim, Jin-Hyuk Hong, Inho Kang, and Nojun Kwak. Semantic sentence matching with densely-connected recurrent and co-attentive information. *CoRR*, abs/1805.11360, 2018. URL http://arxiv.org/abs/1805.11360.

Louis Kirsch, Julius Kunze, and David Barber. Modular networks: Learning to decompose neural computation. In *Advances in Neural Information Processing Systems*, pp. 2408–2418, 2018.

Jan Koutnik, Klaus Greff, Faustino Gomez, and Juergen Schmidhuber. A clockwork rnn. *arXiv preprint arXiv:1402.3511*, 2014.

Alex Krizhevsky. Learning multiple layers of features from tiny images. 2009.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pp. 1097–1105, 2012.

David Krueger, Tegan Maharaj, János Kramár, Mohammad Pezeshki, Nicolas Ballas, Nan Rosemary Ke, Anirudh Goyal, Yoshua Bengio, Aaron Courville, and Chris Pal. Zoneout: Regularizing rnns by randomly preserving hidden activations. *arXiv preprint arXiv:1606.01305*, 2016.

Ngoc-Trung Tran Kwot Sin Lee and Ngai-Man Cheung. Infomax-gan: Mutual information maximization for improved adversarial image generation, 2019.

Samuli Laine and Timo Aila. Temporal ensembling for semi-supervised learning. *CoRR*, abs/1610.02242, 2016. URL http://arxiv.org/abs/1610.02242.

Alex Lamb, Jonathan Binas, Anirudh Goyal, Sandeep Subramanian, Ioannis Mitliagkas, Denis Kazakov, Yoshua Bengio, and Michael C Mozer. State-reification networks: Improving generalization by modeling the distribution of hidden representations. *arXiv preprint*

arXiv:1905.11382, 2019.

Alex Lamb, Anirudh Goyal, Agnieszka Słowik, Michael Mozer, Philippe Beaudoin, and Yoshua Bengio. Neural function modules with sparse arguments: A dynamic approach to integrating information across layers. In Arindam Banerjee and Kenji Fukumizu (eds.), *Proceedings of The 24th International Conference on Artificial Intelligence and Statistics*, volume 130 of *Proceedings of Machine Learning Research*, pp. 919–927. PMLR, 13–15 Apr 2021a. URL https://proceedings.mlr.press/v130/lamb21a.html.

Alex Lamb, Di He, Anirudh Goyal, Guolin Ke, Chien-Feng Liao, Mirco Ravanelli, and Yoshua Bengio. Transformers with competitive ensembles of independent mechanisms. *arXiv preprint arXiv:2103.00336*, 2021b.

Alex M Lamb, Devon Hjelm, Yaroslav Ganin, Joseph Paul Cohen, Aaron C Courville, and Yoshua Bengio. Gibbsnet: Iterative adversarial inference for deep graphical models. In *Advances in Neural Information Processing Systems*, pp. 5089–5098, 2017.

Bruno Lecouat, Chuan-Sheng Foo, Houssam Zenati, and Vijay Chandrasekhar. Manifold regularization with gans for semi-supervised learning. *arXiv preprint arXiv:1807.04307*, 2018.

Yann LeCun and Corinna Cortes. The mnist database of handwritten digits. 2005.

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436, 2015.

Dong-Hyun Lee, Saizheng Zhang, Asja Fischer, and Yoshua Bengio. Difference target propagation. In *Machine Learning and Knowledge Discovery in Databases (ECML/PKDD)*. 2015.

Kwot Sin Lee and Christopher Town. Mimicry: Towards the reproducibility of gan research, 2020.

W. S. Lee, P. L. Bartlett, and R. C. Williamson. Lower bounds on the vc dimension of smoothly parameterized function classes. *Neural Computation*, 7(5):1040–1053, Sep. 1995. ISSN 0899-7667. doi: 10.1162/neco.1995.7.5.1040.

Shuai Li, Wanqing Li, Chris Cook, Ce Zhu, and Yanbo Gao. Independently recurrent neural network (indrnn): Building a longer and deeper rnn. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 5457–5466, 2018a.

Shuai Li, Wanqing Li, Chris Cook, Ce Zhu, and Yanbo Gao. Independently recurrent neural network (indrnn): Building A longer and deeper RNN. *CoRR*, abs/1803.04831, 2018b. URL http://arxiv.org/abs/1803.04831.

Opher Lieber. Rltime: A reinforcement learning library for state-of-the-art q-learning. https://github.com/opherlieber/rltime, 2019.

Ilya Loshchilov and Frank Hutter. SGDR: stochastic gradient descent with restarts. *CoRR*, abs/1608.03983, 2016. URL http://arxiv.org/abs/1608.03983.

Ping Luo, Xinjiang Wang, Wenqi Shao, and Zhanglin Peng. Towards understanding regularization in batch normalization. *arXiv preprint arXiv:1809.00846*, 2018a.

Yucen Luo, Jun Zhu, Mengxi Li, Yong Ren, and Bo Zhang. Smooth neighbors on teacher graphs for semi-supervised learning. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*, pp. 8896–8905, 2018b.

Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=rJzIBfZAb.

Puneet Mangla, Mayank Singh, Abhishek Sinha, Nupur Kumari, Vineeth N Balasubramanian, and Balaji Krishnamurthy. Charting the right manifold: Manifold mixup for few-shot learning. *arXiv preprint arXiv:1907.12087*, 2019.

Charles C Margossian. A review of automatic differentiation and its efficient implementation. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 9(4):e1305, 2019.

Joseph Marino, Yisong Yue, and Stephan Mandt. Iterative amortized inference. *arXiv preprint arXiv:1807.09356*, 2018.

Mason McGill and Pietro Perona. Deciding how to decide: Dynamic routing in artificial neural networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 2363–2372. JMLR. org, 2017.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. In *International Conference on Learning Representations*, 2013.

Tom M Mitchell et al. Machine learning. 1997. *Burr Ridge, IL: McGraw Hill*, 45(37):870–877, 1997.

Sarthak Mittal, Alex Lamb, Anirudh Goyal, Vikram Voleti, Murray Shanahan, Guillaume Lajoie, Michael Mozer, and Yoshua Bengio. Learning to combine top-down and bottom-up signals in recurrent neural networks with attention over modules. In Hal Daumé III and Aarti Singh (eds.), *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 6972–6986. PMLR, 13–18 Jul 2020. URL https://proceedings.mlr.press/v119/mittal20a.html.

Takeru Miyato, Shin ichi Maeda, Masanori Koyama, and Shin Ishii. Virtual adversarial training: a regularization method for supervised and semi-supervised learning. *IEEE transactions on pattern analysis and machine intelligence*, 2018a.

Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral normalization for generative adversarial networks. *arXiv preprint arXiv:1802.05957*, 2018b.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602, 2013. URL http://arxiv.org/abs/1312.5602.

Bastien Moysset and Ronaldo Messina. Manifold mixup improves text recognition with ctc loss. *arXiv preprint arXiv:1903.04246*, 2019.

Preetum Nakkiran. Adversarial robustness may be at odds with simplicity. *arXiv preprint arXiv:1901.00532*, 2019.

Arvind Neelakantan, Luke Vilnis, Quoc V Le, Ilya Sutskever, Lukasz Kaiser, Karol Kurach, and James Martens. Adding gradient noise improves learning for very deep networks. *arXiv preprint arXiv:1511.06807*, 2015.

Daniel Neil, Michael Pfeiffer, and Shih-Chii Liu. Phased lstm: Accelerating recurrent network training for long or event-based sequences. In *Advances in neural information processing systems*, pp. 3882–3890, 2016.

A. Oliver, A. Odena, C. Raffel, E. D. Cubuk, and I. J. Goodfellow. Realistic Evaluation of Deep Semi-Supervised Learning Algorithms. In *Neural Information Processing Systems (NIPS)*, 2018.

Giambattista Parascandolo, Niki Kilbertus, Mateo Rojas-Carulla, and Bernhard Schölkopf. Learning independent causal mechanisms. In *Proceedings of the 35th International Conference on Machine Learning (ICML)*, pp. 4033–4041, 2018a. URL http://proceedings.mlr.press/v80/parascandolo18a.html.

Giambattista Parascandolo, Niki Kilbertus, Mateo Rojas-Carulla, and Bernhard Schölkopf. Learning independent causal mechanisms. In *International Conference on Machine Learning*, pp. 4036–4044. PMLR, 2018b.

Sungrae Park, JunKeon Park, Su-Jin Shin, and Il-Chul Moon. Adversarial dropout for supervised and semi-supervised learning. *AAAI*, 2018.

Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. Understanding the exploding gradient problem. *CoRR, abs/1211.5063*, 2(417):1, 2012.

Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. pp. 1310–1318, 2013.

Judea Pearl. *Causality: Models, Reasoning, and Inference*. Cambridge University Press, New York, NY, 2nd edition, 2009.

Jonas Peters, Dominik Janzing, and Bernhard Schölkopf. *Elements of Causal Inference - Foundations and Learning Algorithms*. MIT Press, Cambridge, MA, USA, 2017. ISBN 978-0-262-03731-0.

Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

Scott Reed and Nando De Freitas. Neural programmer-interpreters. *arXiv preprint arXiv:1511.06279*, 2015.

Eric Ronco, Henrik Gollee, and Peter J Gawthrop. Modular neural networks and self-decomposition. *Technical Report CSC-96012*, 1997.

Clemens Rosenbaum, Tim Klinger, and Matthew Riemer. Routing networks: Adaptive selection of non-linear functions for multi-task learning. *arXiv preprint arXiv:1711.01239*, 2017.

Clemens Rosenbaum, Ignacio Cases, Matthew Riemer, and Tim Klinger. Routing networks and the challenges of modular and compositional computation. *arXiv preprint arXiv:1904.12774*, 2019.

Karsten Roth, Timo Milbich, Samarth Sinha, Prateek Gupta, Bjorn Ommer, and Joseph Paul Cohen. Revisiting training strategies and generalization performance in deep metric learning. In Hal Daumé III and Aarti Singh (eds.), *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 8242–8252. PMLR, 13–18 Jul 2020. URL https://proceedings.mlr.press/v119/roth20a.html.

David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

Mehdi Sajjadi, Mehran Javanmardi, and Tolga Tasdizen. Regularization with stochastic transformations and perturbations for deep semi-supervised learning. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, pp. 1171–1179, USA, 2016. Curran Associates Inc. ISBN 978-1-5108-3881-9. URL http://dl.acm.org/citation.cfm?id=3157096.3157227.

Ruslan Salakhutdinov and Geoffrey Hinton. Deep boltzmann machines. In *Artificial intelligence and statistics*, pp. 448–455, 2009.

Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. arxiv 2016. *arXiv preprint arXiv:1606.03498*.

Tim Salimans, Ian J. Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. *CoRR*, abs/1606.03498, 2016. URL http://arxiv.org/abs/1606.03498.

Adam Santoro, David Raposo, David G Barrett, Mateusz Malinowski, Razvan Pascanu, Peter Battaglia, and Timothy Lillicrap. A simple neural network module for relational reasoning.

In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (eds.), *Advances in Neural Information Processing Systems 30*, pp. 4967–4976. Curran Associates, Inc., 2017a.

Adam Santoro, David Raposo, David G Barrett, Mateusz Malinowski, Razvan Pascanu, Peter Battaglia, and Timothy Lillicrap. A simple neural network module for relational reasoning. In *Advances in neural information processing systems*, pp. 4967–4976, 2017b.

Adam Santoro, Ryan Faulkner, David Raposo, Jack W. Rae, Mike Chrzanowski, Theophane Weber, Daan Wierstra, Oriol Vinyals, Razvan Pascanu, and Timothy P. Lillicrap. Relational recurrent neural networks. *CoRR*, abs/1806.01822, 2018. URL http://arxiv.org/abs/1806.01822.

Benjamin Scellier and Yoshua Bengio. Equilibrium propagation: Bridging the gap between energy-based models and backpropagation. *Frontiers in computational neuroscience*, 11, 2017.

Jürgen Schmidhuber. One big net for everything. *arXiv preprint arXiv:1802.08864*, 2018.

Bernhard Schölkopf, Dominik Janzing, Jonas Peters, Eleni Sgouritsa, Kun Zhang, and Joris Mooij. On causal and anticausal learning. In J. Langford and J. Pineau (eds.), *Proceedings of the 29th International Conference on Machine Learning (ICML)*, pp. 1255–1262, New York, NY, USA, 2012. Omnipress.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

John Shawe-Taylor, Peter Bartlett, Robert C. Williamson, and Martin Anthony. A framework for structural risk minimisation. pp. 68–76, 01 1996. doi: 10.1145/238061.238070.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538*, 2017.

Ravid Shwartz-Ziv and Naftali Tishby. Opening the black box of deep neural networks via information. *CoRR*, abs/1703.00810, 2017. URL http://arxiv.org/abs/1703.00810.

Herbert A Simon. The architecture of complexity. In *Facets of systems science*, pp. 457–476. Springer, 1991.

Courtney J Spoerer, Patrick McClure, and Nikolaus Kriegeskorte. Recurrent convolutional neural networks: a better model of biological object recognition. *Frontiers in psychology*, 8: 1551, 2017.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

Mark Steijvers and Peter Grünwald. A recurrent network that performs a context-sensitive prediction task. In *IN PROCEEDINGS OF THE 18TH ANNUAL CONFERENCE OF THE COGNITIVE SCIENCE*, pp. 335–339. Morgan Kauffman, 1996.

Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. In *International Conference on Learning Representations*, 2014.

Antti Tarvainen and Harri Valpola. Mean teachers are better role models: Weight-averaged consistency targets improve semi-supervised deep learning results. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (eds.), *Advances in Neural Information Processing Systems 30*, pp. 1195–1204. Curran Associates, Inc., 2017a.

Antti Tarvainen and Harri Valpola. Mean teachers are better role models: Weight-averaged consistency targets improve semi-supervised deep learning results. In *Advances in Neural Information Processing Systems 30*, pp. 1195–1204, 2017b.

Naftali Tishby and Noga Zaslavsky. Deep learning and the information bottleneck principle. *CoRR*, abs/1503.02406, 2015. URL http://arxiv.org/abs/1503.02406.

Yuji Tokozume, Yoshitaka Ushiku, and Tatsuya Harada. Between-class learning for image classification. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018a.

Yuji Tokozume, Yoshitaka Ushiku, and Tatsuya Harada. Between-class learning for image classification. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018b.

Simon Tong. Restricted bayes optimal classifiers. In *In Proceedings of the 17th National Conference on Artificial Intelligence (AAAI*, pp. 658–664, 2000.

Dimitris Tsipras, Shibani Santurkar, Logan Engstrom, Alexander Turner, and Aleksander Madry. Robustness may be at odds with accuracy. *stat*, 1050:11, 2018.

Sjoerd Van Steenkiste, Michael Chang, Klaus Greff, and Jürgen Schmidhuber. Relational neural expectation maximization: Unsupervised discovery of objects and their interactions. *arXiv preprint arXiv:1802.10353*, 2018.

Vladimir Vapnik. Pattern recognition using generalized portrait method. *Automation and remote control*, 24:774–780, 1963.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pp. 5998–6008, 2017.

Andreas Veit and Serge Belongie. Convolutional networks with adaptive inference graphs. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 3–18, 2018.

Vikas Verma, Alex Lamb, Christopher Beckham, Amir Najafi, Ioannis Mitliagkas, Aaron Courville, David Lopez-Paz, and Yoshua Bengio. Manifold Mixup: Better Representations by Interpolating Hidden States. *arXiv e-prints*, art. arXiv:1806.05236, Jun 2018.

Vikas Verma, Alex Lamb, Juho Kannala, Yoshua Bengio, and David Lopez-Paz. Interpolation consistency training for semi-supervised learning. *arXiv preprint arXiv:1903.03825*, 2019a.

Vikas Verma, Meng Qu, Alex Lamb, Yoshua Bengio, Juho Kannala, and Jian Tang. Graphmix: Regularized training of graph neural networks for semi-supervised learning. *arXiv preprint arXiv:1909.11715*, 2019b.

Alexander Vezhnevets, Volodymyr Mnih, Simon Osindero, Alex Graves, Oriol Vinyals, John Agapiou, et al. Strategic attentive writer for learning macro-actions. In *Advances in neural information processing systems*, pp. 3486–3494, 2016.

Bao Wang, Xiyang Luo, Zhen Li, Wei Zhu, Zuoqiang Shi, and Stanley J. Osher. Deep learning with data dependent implicit activation function. *CoRR*, abs/1802.00168, 2018a. URL http://arxiv.org/abs/1802.00168.

Xin Wang, Fisher Yu, Zi-Yi Dou, Trevor Darrell, and Joseph E Gonzalez. Skipnet: Learning dynamic routing in convolutional networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 409–424, 2018b.

Paul Werbos. Beyond regression:" new tools for prediction and analysis in the behavioral sciences. *Ph. D. dissertation, Harvard University*, 1974.

Paul J Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.

James CR Whittington and Rafal Bogacz. An approximation of the error backpropagation algorithm in a predictive coding network with local hebbian synaptic plasticity. *Neural computation*, 2017.

Ronald J Williams and David Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural computation*, 1(2):270–280, 1989a.

Ronald J. Williams and David Zipser. A Learning Algorithm for Continually Running Fully Recurrent Neural Networks. *Neural Computation*, 1(2):270–280, 06 1989b. ISSN 0899-7667. doi: 10.1162/neco.1989.1.2.270. URL https://doi.org/10.1162/neco.1989.1.2.270.

Sanghyun Woo, Jongchan Park, Joon-Young Lee, and In So Kweon. Cbam: Convolutional block attention module. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 3–19, 2018.

Yan Wu, Jeff Donahue, David Balduzzi, Karen Simonyan, and Timothy Lillicrap. Logan: Latent optimisation for generative adversarial networks, 2019.

Zuxuan Wu, Tushar Nagarajan, Abhishek Kumar, Steven Rennie, Larry S Davis, Kristen Grauman, and Rogerio Feris. Blockdrop: Dynamic inference paths in residual networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 8817–8826, 2018.

Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. *CoRR*, abs/1605.07146, 2016a. URL http://arxiv.org/abs/1605.07146.

Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. In Edwin R. Hancock Richard C. Wilson and William A. P. Smith (eds.), *Proceedings of the British Machine Vision Conference (BMVC)*, pp. 87.1–87.12. BMVA Press, September 2016b. ISBN 1-901725-59-6. doi: 10.5244/C.30.87. URL https://dx.doi.org/10.5244/C.30.87.

Amir R Zamir, Te-Lin Wu, Lin Sun, William B Shen, Bertram E Shi, Jitendra Malik, and Silvio Savarese. Feedback networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1308–1317, 2017.

Matthew D. Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. *CoRR*, abs/1311.2901, 2013. URL http://arxiv.org/abs/1311.2901.

Han Zhang, Ian Goodfellow, Dimitris Metaxas, and Augustus Odena. Self-attention generative adversarial networks. *arXiv preprint arXiv:1805.08318*, 2018a.

Hongyi Zhang, Moustapha Cisse, Yann N. Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. *International Conference on Learning Representations*, 2018b. URL https://openreview.net/forum?id=r1Ddp1-Rb.

Hongyi Zhang, Moustapha Cisse, Yann N. Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. In *International Conference on Learning Representations*, 2018c. URL https://openreview.net/forum?id=r1Ddp1-Rb.

Jake Zhao and Kyunghyun Cho. Retrieval-augmented convolutional neural networks for improved robustness against adversarial examples. *CoRR*, abs/1802.09502, 2018. URL http://arxiv.org/abs/1802.09502.