Generalization and Data Efficiency in Deep Learning

Arnab Mondal

McGill University and Mila

Fall, 2020

Contents

1 Introduction

This technical report has been divided into three main sections based on the topics covered. This report presents selected topics in data-efficiency and better generalization in deep learning. Generalization in machine learning refers to a model's ability to adapt to new unseen data drawn from the same distribution as the one used to train the model. Having a good generalization strategy helps the model to not overfit the training data and to predict the unseen test data with reasonable accuracy. Regularization in machine learning is a way to prevent overfitting by reducing the complexity of the model. By far, the most common approach is L2 regularization, where the model minimizes the L2 norm of its parameters along with the loss term, which measures how well the model fits the data. The amount of regularization is usually controlled with a regularization coefficient in the regularization term in the loss, to ensure the model does not overfit or underfit. The problem of overfitting is more common in deeper models with higher capacity and has been significantly reduced by modern regularization techniques like dropout [14]. Another way to improve generalization is by augmenting the dataset based on an augmentation scheme, such as adding noise or transformations in the case of images. This can be loosely viewed as a regularization technique. Note that in contrast to standard regularization, which depends on the model parameters, augmentation depends on the available dataset and its distribution. The first section of this report presents generalization techniques based on the manifold of data [11] and shows how they can efficiently boost the model's performance [10]. It is in spirit closely related to the idea of augmentation. These techniques can also prevent a classifier from adversarial attacks, where a small perturbation in input can lead to an error with high confidence [6].

Having discussed generalization in a supervised setting, the next question is if it is possible to use extra data samples without labels to regularize the model. In theory, the more data points we have, the better the estimate of the manifold. This brings us to the idea of semi-supervised learning. Although there are a plethora of traditional methods for semi-supervised learning, the second section focuses on generative modelling and understanding the data manifold. In particular, it reviews the idea of Feature Matching and Triple Generative adversarial networks [1, 5, 12]. Towards the end of the section, the ideas related to generalization using a data manifold are combined with semi-supervised learning [9]. This leads to the idea of data efficiency of labelled samples, by leveraging information from unlabelled samples. The motivation for this is that the acquisition of labelled data can be both time-consuming and expensive depending on the task.

The final section discusses the generalization of a model to new tasks, with use of knowledge from similar tasks when only a few labelled training samples of the new tasks are available. In this report we consider supervised and semi-supervised tasks. This is called "learning to learn" or meta-learning. This can be thought of as an extension of transfer learning which refers to training a model for new task using the knowledge from the previously trained task. The final section focuses on the extension of ideas of supervised meta-learning to the semi-supervised meta-learning [16]. It mainly deals with metric-based meta-learning models. It also introduces the idea of graph neural networks [8,15] and how they can be viewed as a generalization of other metric based meta-learning approaches [4]. In particular, a variation of a graph attention network [15] is used to generalize metric-learning based methods.

2 Generalization in Supervised Learning

2.1 Contractive Autoencoders

An AutoEncoder (AE) is composed of an encoder $f_\theta : \mathbb{R}^s \to \mathbb{R}^d$ and a decoder $g_\phi : \mathbb{R}^d \to \mathbb{R}^s$, where s is the dimension of the input, d is the dimension of the representation space and θ & ϕ are the parameters of the encoder and the decoder respectively. It is a technique for both dimension reduction and representation learning, where the task of the decoder is to reconstruct the input back from the low dimensional encodings. The hypothesis is that the encodings retain enough information about the input to reconstruct it back. In theory, with a complex nonlinear function approximator, $f_\theta : \mathbb{R}^s \rightarrow \mathbb{R}^d$ one can obtain more useful lower-dimensional features compared to techniques like Principle Component Analysis. The loss function of a basic autoencoder can be written as:

$$L_{AE} = \mathbb{E}_{x \sim P_r} R(g_\phi(f_\theta(x)), x)) \tag{1}$$

where $P_r(x)$ denote the density function of the real distribution and R denotes the reconstruction error. Let the training set be $(x^{(1)}, .., x^{(m)})$ and the reconstruction term $R(x, y) = ||x - y||^2$, then the expression reduces to:

$$L_{AE} = \frac{1}{m} \sum_{i=1}^{m} R(g_\phi(f_\theta(x^{(i)})), x^{(i)})) = \frac{1}{m} \sum_{i=1}^{m} ||g_\phi(f_\theta(x^{(i)})) - x^{(i)})||^2. \tag{2}$$

One variation of this is an autoencoder with weight decay, which has an additional L2 norm of its weight in the loss. Another variation is a Denoising AutoEncoder (DAE), where we pass a noisy input through the autoencoder and reconstruct its cleaner version. Both the variations above are ways of regularizing the autoencoder. The loss function of the denoising autoencoder looks like:

$$L_{DAE} = \frac{1}{m} \sum_{i=1}^{m} \mathbb{E}_{\widehat{x} \sim P_n(\widehat{x}|x^{(i)})} ||g_\phi(f_\theta(\widehat{x})) - x^{(i)})||^2 \tag{3}$$

where the corrupted versions $\widehat{x}$ of examples $x^{(i)}$ are obtained from a stochastic corruption process $P_n(\widehat{x}|x^{(i)})$. Typical corruptions are additive isotropic Gaussian noise: $\widehat{x} = x^{(i)} + \epsilon, \quad \epsilon \sim \mathcal{N}(0, \sigma^2 I)$ and a binary masking noise, where a fraction of randomly chosen input components have their value set to 0.

Finally, we can regularize an autoencoder by making it robust to small perturbations around the training data, which is called a Contractive AutoEncoder(CAE) [11]. This can be achieved by minimizing the Jacobian term of the encoder with respect to the input.

The Jacobian term is given as:

$$||J_f(x)||_F^2 = \sum_{i=1}^{s}\sum_{j=1}^{d}(\frac{\partial f_j(x)}{\partial x_i})^2 \tag{4}$$

where $f_j(x)$ is the j-th dimension of the encoder function $f_\theta(x)$ and x_i denotes the i-th dimension of the input. This gives the entire loss function of a CAE as:

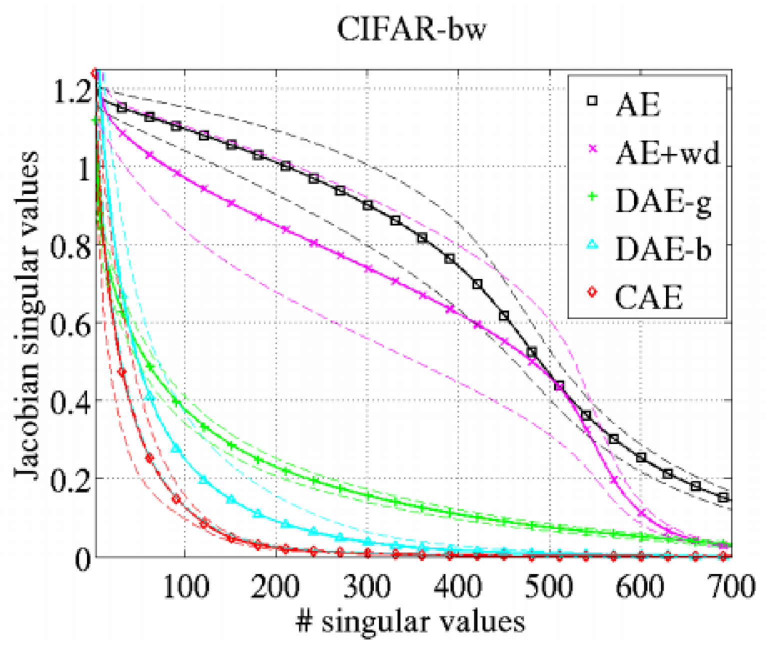
$$L_{CAE} = \frac{1}{m}\sum_{i=1}^{m}(||g_\phi(f_\theta(x^{(i)})) - x^{(i)})||^2 + \lambda||J_f(x^{(i)})||_F^2) \tag{5}$$

where λ controls the emphasis on the Jacobian regularization term. The Jacobian term essentially tries to contract the local neighbourhood of a datapoint in the encoding space. The reconstruction term prevents the neighbourhood's contraction along the direction of the manifold, to preserve enough information to differentiate between two nearby data points. This can be further visualized by looking at the singular values of the Jacobian $J_f(x)$. Fig 1 shows that a trained CAE has fewer dominant directions, corresponding to larger singular values, which suggests that it has successfully contracted the unnecessary normal directions in the encoding space. Note that the Jacobian of the encoder gives the local behaviour in this entire analysis, and a DAE can also be thought of as minimizing the Jacobian $||J_{f\circ g}(x^{(i)})||_F^2$. Rather than contracting the encoding space directly we contract the output in a DAE and observe a similar behaviour of singular values, as shown in Fig 1.

2.2 Manifold Tangent Regularization

Manifold regularization or Jacobian regularization is a way to regularize the classifier $c : \mathbb{R}^s \rightarrow \mathbb{R}^K$, where K is the dimension of output logits, by adding an extra Jacobian term to the standard classifier loss, such as cross-entropy. This term is similar to the one

Figure 1: Average spectrum of the encoder's Jacobian, for the CIFAR-bw dataset. Large singular values correspond to the local directions of "allowed" variation learned from the dataset. Taken from [11]



.

mentioned in the above section:

$$||J_c(x)||_F^2 = \sum_{i=1}^{s} \sum_{j=1}^{d} (\frac{\partial c_j(x)}{\partial x_i})^2. \tag{6}$$

Adding this regularization term makes the classifier robust to local perturbations and has also been shown to enhance its defence against adversarial attacks [6]. Although beneficial, the need to backpropagate for each class output in a classifier makes this step computationally expensive during training. Another way is to minimize the finite-difference by taking a Monte Carlo estimate of it around an ϵ neighbourhood of each data point. Although this kind of regularization makes the classifier robust, it comes at the cost of losing some performance [3]. Intuitively this injects a local invariance to perturbations in an open ball around data-points in the classifier.

Another way of using this idea is to only regularize and inject invariance along the tangent direction of the data manifold [10]. This can be elegantly done by using ideas from a CAE. The training of such classifiers can be divided into three major stages:

- Remove the final layer of the classifier (c), treat it as an encoder (f_θ), design a decoder and train using the CAE loss.

- Use the pretrained encoder f_θ and obtain its Jacobian matrix $J_f(x)$. Compute the Singular Value Decomposition of $J_f(x) = U(x)\Sigma(x)V(x)^T$ and define two sets as a function of x:

$$\mathcal{B}_x = \{V_k(x)|\Sigma_{kk}(x) > \epsilon\} \quad \& \quad \mathcal{H}_x = \{x + v|v \in span(\mathcal{B}_x)\} \tag{7}$$

  where $V_k(x)$ is the k-th column of $V(x)$, $\Sigma_{kk}(x)$ denotes the k-th diagonal element of $\Sigma(x)$ and $span(z_k) = \{x|x = \sum_k w_k z_k, w_k \in \mathbb{R}\}$. Essentially we just need to store $\mathcal{B}_x$ for our calculations. Intuitively, we obtain those directions of the input's manifold, which gives an eigenvalue larger than a threshold. This obtains a basis of tangent directions going by the CAE contraction argument.

- Once $\mathcal{B}_x$ is obtained, add the final layer and finetune the classifier with the added datapoint dependent regularization term along with the standard cross-entropy term:

  $$R(x) = \sum_{u \in \mathcal{B}_x} ||J_c(x)u||^2. \tag{8}$$

  Note that, we are using the tangent basis vectors of each datapoint x to compress the space spanned by them, and the term inside the norm is a linear transformation of u to space of the gradients of encoding using the jacobian $J_c(x)$.

This kind of regularization along the tangent direction of the manifold has shown promise in boosting the performance of the classifier [10]. In particular, this can be helpful if the labelled data is limited and unlabelled data is in abundance. At the end of the next section, this idea is presented.

3 Semi supervised learning

Although there are several traditional ways to carry out semi-supervised learning, we focus on using generative model-based ideas. Further, this section combines the idea of semi-supervised learning with manifold tangent invariance. Towards the end of this section, the current state-of-the-art method in semi-supervised learning is also discussed. According to the semi-supervised learning hypothesis, learning aspects of the input distribution $P(x)$ can improve models of the conditional distribution of the supervised target $P(y|x)$. This hypothesis suggests the possible benefits of using a generative model for a semi-supervised setting. One naive approach is to pre-train the classifier as the encoding network of an autoencoder using the entire dataset, and then the classifier (with a final layer added) is trained with labelled data points. There are also methods to use a Variational Autoencoder (VAE) [7] instead and carry out semi-supervised learning using it. However, this section reviews the methods that involve Generative Adversarial Networks (GANs) [5].

3.1 Generative Adversarial Networks (GANs)

A GAN pairs a parameterized generator g_ϕ, which learns to produce the output samples from a target real distribution, with a parameterized discriminator d_ψ, which learns to distinguish true data from the output of the generator. It is a two-player adversarial optimization problem between a generator and a discriminator. The generator tries to fool the discriminator, and the discriminator tries to keep from being fooled. The objective of a GAN looks like:

$$\min_{g_\phi} \max_{d_\psi} L(g_\phi, d_\psi) = \mathbb{E}_{x \sim P_r(x)}[\log d_\psi(x)] + \mathbb{E}_{z \sim P(z)}[\log(1 - d_\psi(g_\phi(z)))] \tag{9}$$

where $P_r(x)$ denotes the density function of the real distribution and $P(z)$ denotes the density function of the input noise. It can be noted that the first term has no impact on g_ϕ during the optimizer update. The ideal maxima for the discriminator of a GAN is $d^*_\psi(x) = \frac{P_r(x)}{P_r(x)+P_g(x)} \in [0,1]$ where $P_g(x)$ is the density function of generated samples. Substituting that in $L(g_\phi, d^*_\psi) = \int_x (P_r(x) \log \frac{P_r(x)}{P_r(x)+P_g(x)} + P_g(x) \log \frac{P_g(x)}{P_r(x)+P_g(x)}) dx$, we get: $L(g_\phi, d^*_\psi) = JS(\mathbb{P}_r, \mathbb{P}_g) - 2\log 2$ where $JS()$ is the Jensen-Shannon Divergence, $\mathbb{P}_r$ is real and $\mathbb{P}_g$ is the generator distribution. This means that the GAN generator essentially optimizes the JS divergence when the discriminator achieves optimality. Training GANs can be challenging as the discriminator converges quickly and this can lead to vanishing gradients of the generator. One way to alleviate this, as presented in the original paper [5], is to use a non-saturating loss for the generator, i.e, $\max_{g_\phi} \mathbb{E}_{z \sim P(z)}[\log(d_\psi(g_\phi(x)))]$. Although this fixes the vanishing gradients of the generator, the training updates are still unstable. This leads to a Feature Matching (FM) GAN [12], which makes the GAN training stable. The standard generator loss of the GAN is replaced by a feature matching loss in this case, which is given by: $\min_{g_\phi} ||\mathbb{E}_{x \sim P_r(x)} f(x) - \mathbb{E}_{z \sim P(z)} f(g_\phi(z))||_2^2$ where $f()$ is the feature extracted from the pre-final layer of the discriminator d_ψ. However this stability comes at the cost of reduction in generated sample quality as the generator now simply matches the average statistics of the extracted features [12].

3.2 Semi-supervised Learning using an FM-GAN

A byproduct of using an FM-GAN [12] is its improved semi-supervised learning ability. There is an elegant way to show one can learn from unlabelled data using this GAN based generative modelling framework. A standard classifier $c_\psi : \mathbb{R}^s \to \mathbb{R}^K$ classifies a data point x into one of K possible classes and outputs a K-dimensional vector of logits $[l_1, \ldots, l_K]$, that can be turned into class probabilities by applying the softmax: $p_{c_\psi}(y = j|x) = \frac{\exp(l_j(x))}{\sum_{k=1}^{K} \exp(l_k(x))}$. In supervised learning, such models are trained by minimizing the cross-entropy between the observed labels and the model predictive distribution $p_{c_\psi}(y = j|x)$. For semi-supervised learning, one simply adds samples from the generator g_ϕ to the data set, labeling them with a new class $y = K + 1$, while correspondingly increasing the dimension of the classifier output from K to $K + 1$. Then $p_{c_\psi}(y = K + 1|x)$ can be used as the probability that x is fake, corresponding to $1 - d_\psi(x)$ in the original GAN framework. The objective that the classifier which is also used as a discriminator maximizes is:

$$\begin{aligned} \max_{c_\psi} L(c_\psi, g_\phi) = \mathbb{E}_{x,y\sim P_r(x,y)} \log p_{c_\psi}(y|x, y < K + 1) + \mathbb{E}_{x\sim P_r(x)} \log[1 - p_{c_\psi}(y = K + 1|x)] \\ + \mathbb{E}_{z\sim P(z)} \log[p_{c_\psi}(y = K + 1|g_\phi(z))] \end{aligned} \tag{10}$$

The first term is the supervised loss, the second term is the unlabelled loss, and the third is the loss coming from the fake samples. Intuitively the classifier tries to label the labelled samples to its class, the unlabelled samples as one from the real classes and the fake sample as the $K + 1$-th class. As the classifier with $K + 1$ outputs is over-parameterized, we can set $l_{K+1}(x) = 0 \quad \forall x$ which reduces the first term in Equation 10 to a standard cross-entropy loss, and any classifier can be used in this setting. It has been observed that the feature matching loss of the generator suits semi-supervised learning as less realistic samples give a better decision boundary for a discriminator, which is the classifier in this case [12].

3.3 Triple GAN

A Triple GAN [1] solves the problem of poor generated sample quality faced by an FM GAN by decoupling the classifier and the discriminator. It also gives comparable semi-supervised learning performance. Unlike an FM-GAN, now we have three networks: a class conditional generator g_ϕ, a joint discriminator d_θ and a classifier c_ψ (which outputs a K-dimensional vector of logits $[l_1, \dots, l_K]$ like in Section 3.2). Two major differences from standard GANs are that it uses a generator that is conditioned on the class and a discriminator, which labels the joint distribution as true or fake. There are three players, and the overall adversarial objective becomes:

$$\begin{aligned}\min_{g_\phi, c_\psi} \max_{d_\theta} L(g_\phi, c_\psi, d_\theta) = &\mathbb{E}_{x,y \sim P_r(x,y)}[\log d_\theta(x, y)] + \alpha \mathbb{E}_{x \sim P_r(x)}[\log(1 - d_\theta(x, y_c))] \\ &+ (1-\alpha)\mathbb{E}_{z \sim P(z), y \sim \mathcal{U}(K)}[\log(1 - d_\theta(g_\phi(z, y), y))]\end{aligned} \tag{11}$$

where $\mathcal{U}(K)$ refers to a uniform distribution over K classes, $y_c = arg \max_i l_i(x)$ and $\alpha \in (0, 1)$ denotes the relative importance of classification and generation. Notice that the classifier is treated as a pseudo-label generator in the above framework and y_c can be sampled instead of taking $arg \max$. The expression of joint distributions of the generator and the classifier is $P_{c_\psi}(x, y) = P_r(x)P_{c_\psi}(y|x)$ and $P_{g_\phi}(x, y) = P(y)P_{g_\phi}(x|y)$. While the first expression is straightforward, the second equation is obtained assuming $y \sim P(y) = \mathcal{U}(K)$ and distribution $P(z)$ is transformed to $P_{g_\phi}(x|y)$ by g_ϕ given the label y. This game achieves its equilibrium if and only if $P_r(x, y) = (1-\alpha)P_{g_\phi}(x, y) + \alpha P_{c_\psi}(x, y)$. As this doesn't guarantee $P_r(x, y) = P_{g_\phi}(x, y) = P_{c_\psi}(x, y)$, the standard crossentropy term is added in the classifier loss [1]. In practice once the generator is well trained, the generated samples are also passed through the classifier and the cross entropy loss is computed which further regularizes the network. The final classifier loss is:

$$\min_{c_\psi}\{L(g_\phi, c_\psi, d_\theta) - \mathbb{E}_{x,y \sim P_r(x,y)} \log p_{c_\psi}(y|x) - \mathbb{E}_{z \sim P(z), y \sim \mathcal{U}(K)} \log p_{c_\psi}(y|g_\phi(z, y))\}. \tag{12}$$

A Triple GAN gives performances comparable to FM GAN and successfully generates realistic looking samples.

3.4 Manifold Tangent Invariance for Semi-supervised Learning

The GAN based semi-supervised learning methods can be improved further using the regularization ideas discussed in the previous section. In theory, we can estimate the manifold tangent directions from the entire dataset as long as we use an encoding-decoding architecture with Jacobian regularization of the encoder. In [9], a BiGAN [3] has been used for this task, and this has boosted FMGAN based semi-supervised learning. A BiGAN has an encoder f_θ, a generator or decoder g_ϕ and a joint discriminator d_η. The objective of a BiGAN is given by:

$$\min_{f_\theta, g_\phi} \max_{d_\eta} L(f_\theta, g_\phi, d_\eta) = \mathbb{E}_{x\sim P_r(x)}[\log d_\eta(x, f_\theta(x)] + \mathbb{E}_{z\sim P(z)}[\log(1 - d_\eta(g_\phi(z), z))]. \tag{13}$$

Training a biGAN can lead to convergence issues; hence [9] suggests using feature matching for training its encoder and generator instead, along with a modified adversarial loss for the discriminator. Hence, the objective of the encoder and generator becomes:

$$\min_{f_\theta, g_\phi} ||\mathbb{E}_{x\sim P_r(x)} \hat{d}(x, f_\theta(x)) - \mathbb{E}_{z\sim P(z)} \hat{d}(g_\phi(z), z)||_2^2 + \mathbb{E}_{x\sim P_r(x)} ||J_f(x)||_F^2 \tag{14}$$

and that of the discriminator:

$$\begin{aligned} \max_{d_\eta} \mathbb{E}_{x\sim P_r(x)}[\log d_\eta(x, f_\theta(x)] + \mathbb{E}_{z\sim P(z)}[\log(1 - d_\eta(g_\phi(z), z))] + \\ \mathbb{E}_{x\sim P_r(x)}[\log(1 - d_\eta(g_\phi(f_\theta(x)), f_\theta(x)))] \end{aligned} \tag{15}$$

where $\hat{d}$ represents the extracted features from the pre-final layer of the discriminator. After training the BiGAN, the manifold tangent basis set $\mathcal{B}_x$ can be obtained as described in Subsection 2.2. Notice that this entire process could have also been carried out just with a simple autoencoder but using a BiGAN leads to the reuse of the trained generator g_ϕ.

Once this is obtained, one can train a semi-supervised FM GAN as described in Subsection 3.2 and add the additional regularization terms in the classifier loss given in Equation 10. The final classifier loss becomes:

$$\min_{c_\psi}\{-L(c_\psi, g_\phi) + \lambda_1 \mathbb{E}_{x\sim P_r(x)} \sum_{u\in\mathcal{B}_x} ||J_c(x)u||^2 + \lambda_2 \mathbb{E}_{x\sim P_r(x)} ||J_c(x)||_F^2\} \tag{16}$$

The last term tries to contract the network in every direction from a point on data manifold, which results in robustness to perturbations. λ_1 and λ_2 controls the amount of regularization we need in an FMGAN. In a low data setting, with several unlabelled examples that result in a reasonable estimate of the manifold tangents, regularizing along the tangent direction can be quite useful [9].

3.5 Fixmatch

Apart from generative modelling based methods, one of the most straightforward ideas in semi-supervised learning is to use consistency regularization and pseudo-labelling. Consistency regularization utilizes unlabeled data by assuming that the model should output similar predictions when fed perturbed versions of the same input. In large scale image based models, a stronger form of augmentation such as RandAugment [2] is used. Pseudo-labeling leverages the idea of the use of the model itself to obtain artificial labels for unlabeled data. It gives a hard label, the $arg\max$ of the model's output, to the data samples, where the largest class probability is above a certain threshold. Note that as the pseudo labels of the samples are used to compute their cross-entropy loss, this idea is closely related to entropy minimization. Now Fixmatch combines both these ideas and uses an objective given by (using notations from Subsection 3.2):

$$\min_{c_\psi} \mathbb{E}_{x,y\sim P_r(x,y)} \log p_{c_\psi}(y|\alpha(x)) + \mathbb{E}_{x\sim P_r(x)} \mathbb{I}(\max l_i(x) > \tau) \log p_{c_\psi}(y_c|\mathcal{A}(x)) \tag{17}$$

where $\alpha()$ & $\mathcal{A}()$ refer to weak & strong augmentation, $y_c = arg\max_i l_i(x)$ and $\mathbb{I}$ is the indicator function which decides which unlabelled samples to consider. Note that weak augmentation refers to simple cropping, reflection and rotation for image input. A simple architecture like this, combined with a ResNet backbone, can give state-of-the-art performance [13].

4 Semi-supervised Meta-Learning

Meta-learning, also known as "learning to learn", refers to a class of learning algorithms that can learn new skills or adapt to new environments rapidly with a few training examples, inspired by how humans learn. They can further be metric-based, like learning an efficient distance metric, or model-based, such as recurrent networks with external and internal memory or optimization-based where the model parameters are optimized for faster learning. Metric based methods and how they can be extended for semi-supervised tasks are explained in this section. Before looking at the algorithms, it is important to define the problem of meta-learning and semi-supervised meta-learning. In a typical meta-learning setting, the dataset consisting of N classes is divided in N_{train} and N_{test} classes. To form n_{train} training episodes or tasks $\{\mathcal{T}_i\}_{i=0}^{n_{train}}$ of Z-shot K-way one need to sample K classes from N_{train} classes. From these K class datasets, Z labelled instances of each class are chosen for the labelled support set: $S_l^i = \{(x_1, y_1), ..., (x_{Z\times K}, y_{Z\times K}\}$, M labelled instances are chosen for the query set: $Q_l^i = \{(x_1, y_1), ..., (x_M, y_M\}$ and R unlabelled instances are chosen for the unlabelled support set: $S_u^i = \{x_1, ..., x_R\}$. Note that the set S_l^i, Q_l^i and S_u^i are mutually exclusive. We can similarly form n_{test} testing episodes or tasks $\{\mathcal{T}_i\}_{i=0}^{n_{test}}$ from N_{test} classes for testing our meta-learner. In supervised meta-learning, $R = 0$ for both training and testing episodes. Before getting into the details of training these models,we review Graph Neural Networks (GNNs) which are required for generalizing the metric based meta-learning ideas.

4.1 Graph Neural Networks

This section presents GNNs in their most simple message passing form. A GNN framework takes an input graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, along with a set of node features $X \in \mathbb{R}^{d\times|\mathcal{V}|}$, and uses this information to generate node embeddings $z_u, \forall u \in \mathcal{V}$. The message passing update in most general form can be expressed as:

$$h_u^{(k+1)} = \text{update}^{(k)}(h_u^{(k)}, \text{aggregate}^{(k)}(h_v^{(k)}, \forall v \in \mathcal{N}(u)) \tag{18}$$

where the update$^{(k)}()$ and aggregate$^{(k)}()$ can be a whole range of functions and $\mathcal{N}(u)$ denotes the neighbours of u. Since the aggregate$^{(k)}()$ function takes a set as input, GNNs defined in this way are permutation equivariant by design. The most basic choice aggregate$^{(k)}()$ function is addition scaled by some normalization factor like symmetric degree normalization and that of update$^{(k)}()$ is weighted addition. The vector expression for this looks like:

$$H^{(k+1)} = \sigma(H^{(k)}W_{self}^{(k)} + D^{-1/2}AD^{-1/2}H^{(k)}W_{neigh}^{(k)} + b^{(k)}) \tag{19}$$

where $h_u^{(k)}$ is placed row-wise to form $H^{(k)}$, A is the adjacency matrix (without self connection), D is the degree matrix ($D_{ii} = \sum_j A_{ij}$), $W_{self}^{(k)}$ & $W_{neigh}^{(k)}$ are the weight matrices, $b^{(k)}$ is the bias and $\sigma()$ is a nonlinearity. When $W_{self}^{(k)}$ & $W_{neigh}^{(k)}$ are shared this reduces to the standard Graph Convolution Networks (GCNs) [8]. The aggregate() function can be further generalized by using an attention mechanism [15] given by:

$$\text{aggregate}(h_v, \forall v \in \mathcal{N}(u) = \sum_{v\in\mathcal{N}(u)} \alpha_{u,v} W_{neigh} h_v \tag{20}$$

and the attention can be computed as:

$$\alpha_{u,v} = \frac{\exp(\sigma(a^T[Wh_u \bigoplus Wh_v])))}{\sum_{\bar{v}\in\mathcal{N}(u)} \exp(\sigma(a^T[Wh_u \bigoplus Wh_{\bar{v}}])))} \tag{21}$$

where a is a trainable attention vector, W is a trainable matrix, $\sigma()$ is a non-linearity and $\bigoplus$ denotes the concatenation operation. Note that the superscript denoting the iteration is omitted in the above equations for notational brevity. Having defined the aggregate() like this, one can replace the weighted addition in the update() with concatenation followed by a linear layer or a linear layer for the self features and then concatenate with aggregated features. If the adjacency matrix of GNN contains self-connection then self attention is computed, and the update() function only passes the aggregated features making it a Graph Attention Network (GAT). [15]

4.2 Generalizing metric based meta-learning with GNNs

For using a GNN in meta-learning which typically uses image datasets, the images have to be brought to a feature space using a shared encoder $f_\theta()$. During training in an episode for supervised meta-learning, once the encoded features are obtained, they are appended with corresponding one hot labels for labelled support images and uniform confidence for query images. For support images, final feature representations are obtained by $\mathbf{x} = [f_\theta(x) \bigoplus h(y)]\ \forall (x, y) \in S_l^i$ where $\bigoplus$ denotes the concatenation and $h()$ denotes the one-hot function. While for query images they are obtained by $\mathbf{x} = [f_\theta(x) \bigoplus K^{-1}\mathbf{1}_K]$ $\forall (x) \in Q_l^i$ where $\mathbf{1}_K$ is the K-dimensional one vector. These are further passed through a densely connected GNN, and loss is computed at the output of the query images. The optimization objective for an episode becomes:

$$L(\mathcal{T}_i) = \sum_{k=1}^{M} -logP(y_k|x_k, \mathcal{T}_i) \qquad \forall (x_k, y_k) \in Q_l^i \tag{22}$$

where this negative log-likelihood is computed based on the output logits at the query nodes. The GNN used is essentially similar to a graph attention network with attention interpreted as the metric. The distance metric between two feature vectors is computed as $\delta_{\hat{\theta}}(|\mathbf{x}_u - \mathbf{x}_v|)$, where $\delta_{\hat{\theta}}()$ is a MLP, which gives the attention weights:

$$\alpha_{u,v} = \frac{\exp(-\delta_{\hat{\theta}}(|\mathbf{x}_u - \mathbf{x}_v|))}{\sum_{\bar{v}\in\mathcal{N}(u)} \exp(-\delta_{\hat{\theta}}(|\mathbf{x}_u - \mathbf{x}_{\bar{v}}|))} \tag{23}$$

Now the vector form of the final update equation used in [4] is:

$$H^{(k+1)} = \sigma(H^{(k)}W^{(k)}_{self} + \mathcal{A}H^{(k)}W^{(k)}_{neigh} + b^{(k)}) \tag{24}$$

where $\mathcal{A}_{u,v} = \alpha_{u,v}$ is the attention matrix. This encoder, followed by an attention based GNN, generalizes all the existing metric-based meta-learning as it has a non-linear metric that is learned during the training. This idea also naively exploits the unlabelled samples from the support set by obtaining their feature vector as $\mathbf{x} = [f_\theta(x) \bigoplus K^{-1}\mathbf{1}_K] \; \forall(x) \in S^i_u$ and passing them through the GNN. The loss for this setting is still the negative log-likelihood computed at the query nodes, but having connections to unlabelled samples allows information flow from them to generalize the model further.

4.3 MetaGANs

A MetaGAN [16] uses the same idea of feature matching based semi-supervised learning from Section 3.2. It uses all the unlabelled images in each task in the query set to backpropagate the unsupervised loss and further regularize the model. This technique is meta-learner agnostic and only provides a way to incorporate losses from unlabelled samples in training. For this section, consider unsupervised query sets $Q^i_u = S^i_u$ for each task $\mathcal{T}_i$. Every task has an $K + 1$-th class for the fake samples. To generate fake data that is close to the real data manifold in one specific task $\mathcal{T}_i$, there is an instance encoder $e_\theta()$ which takes $x \in S^i_l$ and outputs an encoded vector $e_\theta(x)$. The encoded vectors are aggregated using an average pooling operator to give a task representation $h_{\mathcal{T}_i}$. Now R random $z(\sim P(z))$ are sampled and a conditional generator $g_\phi(z, h_{\mathcal{T}_i})$ gives R generated samples which are stored in a set Q^i_g. As the meta-learner model is used as a discriminator in this setup, the loss functions become:

$$L_{lab}(\mathcal{T}_i) = \sum_{k=1}^{M} -logP(y = y_k|x_k, y_k \leq K, \mathcal{T}_i) \qquad \forall(x_k, y_k) \in Q_l^i \tag{25}$$

$$L_{unlab}(\mathcal{T}_i) = \sum_{k=1}^{R} -logP(y \leq K|x_k, \mathcal{T}_i) \qquad \forall(x_k) \in Q_u^i \tag{26}$$

$$L_{fake}(\mathcal{T}_i) = \sum_{k=1}^{R} -logP(y = K+1|x_k, \mathcal{T}_i) \qquad \forall(x_k) \in Q_g^i \tag{27}$$

Adding the above three losses gives the total loss of the model. Apart from this, the generator and the encoder are trained by either minimizing the non saturating loss function:

$$L_{gen}(\mathcal{T}_i) = \sum_{k=1}^{R} -logP(y \leq K|x_k, \mathcal{T}_i) \qquad \forall(x_k) \in Q_g^i \tag{28}$$

or minimizing the Feature Matching loss, as mentioned in Section 3.2. In contrast to GNN based semi-supervised meta-learning, a MetaGAN provides a way to compute loss for the unlabelled samples, which is then used to backpropagate gradients. Often this can be more beneficial than naively pulling information from unlabelled data using a graph. As both ideas are complementary, their combination can potentially lead to better models. Whether simple ideas like Fixmatch can be combined with GNN based meta-learner is an open question. Finally, all this makes semi-supervised meta-learning both an exciting and active area of research.

References

[1] CHONGXUAN, L., XU, T., ZHU, J., AND ZHANG, B. Triple generative adversarial nets. In *Advances in neural information processing systems* (2017), pp. 4088–4098.

[2] CUBUK, E. D., ZOPH, B., SHLENS, J., AND LE, Q. V. Randaugment: Practical automated data augmentation with a reduced search space. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops* (2020), pp. 702–703.

[3] DONAHUE, J., KRÄHENBÜHL, P., AND DARRELL, T. Adversarial feature learning. *arXiv preprint arXiv:1605.09782* (2016).

[4] GARCIA, V., AND BRUNA, J. Few-shot learning with graph neural networks. *arXiv preprint arXiv:1711.04043* (2017).

[5] GOODFELLOW, I., POUGET-ABADIE, J., MIRZA, M., XU, B., WARDE-FARLEY, D., OZAIR, S., COURVILLE, A., AND BENGIO, Y. Generative adversarial nets. In *Advances in neural information processing systems* (2014), pp. 2672–2680.

[6] JAKUBOVITZ, D., AND GIRYES, R. Improving dnn robustness to adversarial attacks using jacobian regularization. In *Proceedings of the European Conference on Computer Vision (ECCV)* (2018), pp. 514–529.

[7] KINGMA, D. P., AND WELLING, M. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114* (2013).

[8] KIPF, T. N., AND WELLING, M. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907* (2016).

[9] KUMAR, A., SATTIGERI, P., AND FLETCHER, T. Semi-supervised learning with gans: Manifold invariance with improved inference. In *Advances in Neural Information Processing Systems* (2017), pp. 5534–5544.

[10] RIFAI, S., DAUPHIN, Y. N., VINCENT, P., BENGIO, Y., AND MULLER, X. The manifold tangent classifier. In *Advances in neural information processing systems* (2011), pp. 2294–2302.

[11] RIFAI, S., VINCENT, P., MULLER, X., GLOROT, X., AND BENGIO, Y. Contractive auto-encoders: Explicit invariance during feature extraction. In *Icml* (2011).

[12] SALIMANS, T., GOODFELLOW, I., ZAREMBA, W., CHEUNG, V., RADFORD, A., AND CHEN, X. Improved techniques for training gans. In *Advances in neural information processing systems* (2016), pp. 2234–2242.

[13] SOHN, K., BERTHELOT, D., LI, C.-L., ZHANG, Z., CARLINI, N., CUBUK, E. D., KURAKIN, A., ZHANG, H., AND RAFFEL, C. Fixmatch: Simplifying semi-supervised learning with consistency and confidence. *arXiv preprint arXiv:2001.07685* (2020).

[14] SRIVASTAVA, N., HINTON, G., KRIZHEVSKY, A., SUTSKEVER, I., AND SALAKHUTDINOV, R. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research 15*, 1 (2014), 1929–1958.

[15] VELIČKOVIĆ, P., CUCURULL, G., CASANOVA, A., ROMERO, A., LIO, P., AND BENGIO, Y. Graph attention networks. *arXiv preprint arXiv:1710.10903* (2017).

[16] ZHANG, R., CHE, T., GHAHRAMANI, Z., BENGIO, Y., AND SONG, Y. Metagan: An adversarial approach to few-shot learning. In *Advances in Neural Information Processing Systems* (2018), pp. 2365–2374.