Semi Supervised Learning with Deep Generative Models

Ambuj Ojha [1]

Abstract

We review the latest research on the use of Deep Generative Models for Semi Supervised Learning on Open Source Image Datasets. We also provide open source, high quality, modular implementations in Pytorch of the algorithms discussed. While many advances have been made, the performances of Deep Generative Models still is not as good as certain Discriminative techniques such as Temporal Ensembling and Mean Teacher enhancements to that technique. We discuss some theoretical reasons why this is so and provide a path for improvements.

1. Semi Supervised Learning

Semi-supervised learning is a set of techniques used to make use of unlabelled data in supervised learning problems (e.g. classification and regression).

Most of the massive amounts of data on the internet is unlabelled. Rarely do we have something in a nice benchmark format that tells us exactly what we need to do. As an example, there are trillions of unlabelled images all over the internet but only a tiny fraction actually have any sort of label. So our goal here is to get the best performance with a tiny amount of labelled data.

Humans somehow are very good at this because our brains have learnt common features about what we see that allow us to quickly categorize things into buckets like 'Car' or a 'Deer'. Using Deep Generative Models, we want to allow a machine to learn some additional (useful) features in an unsupervised way to help the actual task of which we have very few examples.

[*]Equal contribution [1]New York University, New York, United States of America. Correspondence to: Ambuj Ojha <apo249@nyu.edu>.

Proceedings of the 35^{th} International Conference on Machine Learning, Stockholm, Sweden, PMLR 80, 2018.

2. Variational Auto Encoders

From (Doersch, 2016). Suppose we have some data X, a generative probability model $P(X|\theta)$ that shows us how to randomly sample (e.g. generate) data points that follow the distribution of X, assuming we know the proper values of the θ parameters. Using the Bayes theorem we have:

$$\begin{aligned} p(\theta|X) &= \frac{p(X|\theta)p(\theta)}{p(X)} \\ &= \frac{p(X|\theta)p(\theta)}{\int_{-\infty}^{\infty} p(X|\theta)p(\theta)d\theta} \\ \text{posterior} &= \frac{\text{likelihood} \cdot \text{prior}}{\text{evidence}} \end{aligned} \tag{1}$$

Our goal is to find the posterior, $P(\theta|X)$, that tells us the distribution of the θ parameters so that we can use $P(X|\theta)$ to generate some new data points (e.g. use variational autoencoders to generate a new image). Unfortunately, this problem is intractable (mostly the denominator) for all but the simplest problems.

We approximate $P(\theta|X)$ by another function $Q(\theta|X)$ (it's usually conditioned on X but not necessarily). And solving for Q is (relatively) fast because we can assume a particular shape for $Q(\theta|X)$ and turn the inference problem (i.e. finding $P(\theta|X)$ into an optimization problem (i.e. finding Q). Of course, it can't be just a random function, we want it to be as close as possible to $P(\theta|X)$, which will depend on the structural form of $Q(\theta|X)$ (how much flexibility it has), our technique to find it, and our metric of "closeness".

The standard way of measuring 'closeness' is to use KL Divergence, which is:

$$
\begin{aligned}
D_{KL}(Q||P) &= \int_{-\infty}^{\infty} q(\theta|X) \log \frac{q(\theta|X)}{p(\theta|X)} d\theta \\
&= \int_{-\infty}^{\infty} q(\theta|X) \log \frac{q(\theta|X)}{p(\theta, X)} d\theta \\
&+ \int_{-\infty}^{\infty} q(\theta|X) \log p(X) d\theta \\
&= \int_{-\infty}^{\infty} q(\theta|X) \log \frac{q(\theta|X)}{p(\theta, X)} d\theta + \log p(X) \\
&= E_q\big[\log \frac{q(\theta|X)}{p(\theta, X)}\big] + \log p(X)
\end{aligned} \tag{2}
$$

Rearranging, dropping the KL divergence term and putting it in terms of an expectation of $q(\theta)$, we get what's called the Evidence Lower Bound (ELBO) for a single data point X:

$$
\begin{aligned}
\log p(X) &\geq -E_q\big[\log \frac{q(\theta|X)}{p(\theta, X)}\big] \\
&= E_q\big[\log p(\theta, X) - \log q(\theta|X)\big] \\
&= E_q\big[\log p(X|\theta) + \log p(\theta) - \log q(\theta|X)\big] \\
&= E_q\big[\text{likelihood} + \text{prior} - \text{approx. posterior}\big]
\end{aligned} \tag{3}
$$

For multiple data points, we just sum over them because we're in log space (We assume independence between data points).

ELBO is a lower bound on the evidence, that is, it's a lower bound on the probability of our data occurring given your model. Maximizing the ELBO is equivalent to minimizing the KL divergence. The first two terms in the ELBO try to maximize the MAP estimate (likelihood + prior). The last term tries to ensure Q is diffuse (maximize information entropy).

A Variational Autoencoder defines a generative model for our data. It has two parts an Encoder and a Decoder.

The 'Encoder' goes from observed values to a latent state $(XtoZ)$. This is actually our variational approximation of the posterior $(q(Z|X))$, which is also a neural network defined by $g_{Z|X}$.

The 'Decoder', given an isotropic standard normal distribution (Z), runs it through a deep net (defined by $g_{X|Z}$) to produce the observed data (X).

This is visualized in Figure 1.

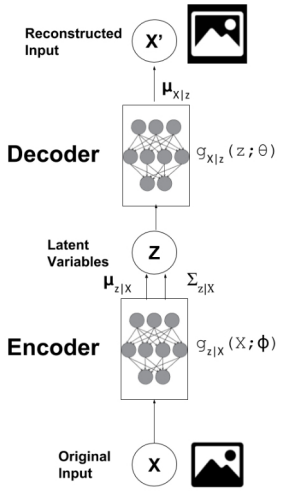


Figure 1. Vanilla Variational Autoencoder

2.1. A Vanilla VAE for Semi-Supervised Learning (M1 Model)

From (Kingma et al., 2014). The latent space defined by Z should capture some useful information about our data such that it's easily separable in our supervised learning problem. Here the training of the Variational Auto Encoder doesn't directly involve any of the y labels.

Steps:

- Train a VAE using all our data points (labelled and unlabelled), and transform our observed data (X) into the latent space defined by the Z variables.
- Solve a standard supervised learning problem on the labelled data using (Z,Y) pairs (where Y is our label).

2.2. Extending the VAE for Semi-Supervised Learning (M2 Model)

From (Kingma et al., 2014). In the M1 model, we ignored our labelled data when training our VAE. The M2 model explicitly takes it into account.

In the 'Decoder':

$$
\begin{aligned}
p(\mathbf{x}|y, \mathbf{z}) &= f(\mathbf{x}; y, \mathbf{z}, \theta) \\
p(\mathbf{z}) &= \mathcal{N}(\mathbf{z}|0, I) \\
p(y|\pi) &= \text{Cat}(y|\pi) \\
p(\pi) &= \text{SymDir}(\alpha)
\end{aligned} \tag{4}
$$

where:

- $\mathbf{x}$ is a vector of our observed variables

- $f(\mathbf{x}; y, \mathbf{z}, \theta)$ is a suitable likelihood function to model our output such as a Gaussian or Bernoulli. We use a deep net to approximate it based on inputs $y, \mathbf{z}$ with network weights defined by θ

- $\mathbf{z}$ is a vector latent variables (same as vanilla VAE)

- y is a one-hot encoded categorical variable representing our class labels, whose relative probabilities are parameterized by π

- SimDir is Symmetric Dirichlet distribution with hyper-parameter α (a conjugate prior for categorical/multinomial variables)

In order to use this for Semi-Supervised Learning, we will define a approximate posterior function $q_\phi(y|\mathbf{x})$ using a deep net that is basically a classifier. We then train this classifier for both labelled and unlabelled data by just training this extended VAE. Figure 2 shows a visualization of the network.

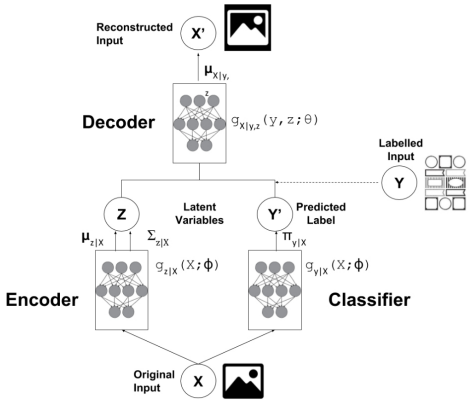


Figure 2. M2 Variational Autoencoder for Semi-Supervised Learning

We have two cases here: one where we observe the y labels and one where we don't. We have to deal with them differently when constructing the approximate posterior q as well as in the Variational Objective.

2.2.1. Variational Objective with Unlabelled Data

In this case, we'll treat $y, \mathbf{z}$ as the unknown latent variables, and perform variational inference (i.e. define approximate posteriors) over them. We exclude π because we don't really care what its posterior is in this case.

We'll assume the approximate posterior $q_\phi(y, \mathbf{z}|\mathbf{x})$ has a fully factorized form as such:

$$\begin{aligned} q_\phi(y, \mathbf{z}|\mathbf{x}) &= q_\phi(\mathbf{z}|\mathbf{x}) q_\phi(y|\mathbf{x}) \\ q_\phi(y|\mathbf{x}) &= \text{Cat}(y|\pi_\phi(\mathbf{x})) \\ q_\phi(\mathbf{z}|\mathbf{x}) &= \mathcal{N}(\mathbf{z}|\mu_\phi(\mathbf{x}), diag(\sigma^2_\phi(\mathbf{x}))) \end{aligned} \tag{5}$$

where $\mu_\phi(\mathbf{x}), \sigma^2_\phi(\mathbf{x}), \pi_\phi(\mathbf{X})$ are all defined by neural networks parameterized by ϕ that we will learn. Note that, $\pi_\phi(\mathbf{X})$ should not be confused with our actual parameter π above, the former is a point-estimate coming out of our network, the latter is a random variable as a symmetric Dirichlet.

Now, we write the ELBO to determine our variational objective for a single data point:

$$\begin{aligned} \log p_\theta(\mathbf{x}) &\geq E_{q_\phi(y,\mathbf{z}|\mathbf{x})}\Big[\log p_\theta(\mathbf{x}|y,\mathbf{z}) + \log p_\theta(y) \\ &\quad + \log p_\theta(\mathbf{z}) - \log q_\phi(y, \mathbf{z}|\mathbf{x})\Big] \\ &= E_{q_\phi(y|\mathbf{x})}\Big[E_{q_\phi(\mathbf{z}|\mathbf{x})}\big[\log p_\theta(\mathbf{x}|y,\mathbf{z}) + K_1 \\ &\quad + \log p_\theta(\mathbf{z}) - \log q_\phi(y|\mathbf{x}) - \log q_\phi(\mathbf{z}|\mathbf{x})\big]\Big] \\ &= E_{q_\phi(y|\mathbf{x})}\Big[E_{q_\phi(\mathbf{z}|\mathbf{x})}\big[\log p_\theta(\mathbf{x}|y,\mathbf{z})\big] + K_1 \\ &\quad - KL[q_\phi(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{z})] - \log q_\phi(y|\mathbf{x})\Big] \\ &= E_{q_\phi(y|\mathbf{x})}\big[-\mathcal{L}(\mathbf{x}, \dagger) - \log q_\phi(y|\mathbf{x})\big] \\ &= \sum_y \big[q_\phi(y|\mathbf{x})(-\mathcal{L}(\mathbf{x}, y)) - q_\phi(y|\mathbf{x}) \log q_\phi(y|\mathbf{x})\big] \\ &= \sum_y q_\phi(y|\mathbf{x})(-\mathcal{L}(\mathbf{x}, y)) + \mathcal{H}(q_\phi(y|\mathbf{x})) \end{aligned} \tag{6}$$

Note that we can factor our q_ϕ function into the separate y and $\mathbf{z}$ parts for both the expectation and the log. Also we absorb $\log p_\theta(y)$ into a constant because p(y) = $p(y|\pi)p(\pi)$, a Dirichlet-multinomial distribution, and simplifies to a constant (Our model assumes that y's are equally likely to happen).

Next, we notice that some terms form a KL distribution between $q_\phi(\mathbf{z}|\mathbf{x})$ and $p_\theta(\mathbf{z})$. Then, we group a few terms together and name it $\mathcal{L}(\mathbf{x}, y)$. This latter term is essentially the same variational objective we use for a vanilla variational autoencoder (without the reference to y). Finally, we explicitly write out the expectation with respect to y.

So Equation 6 defines our objective function for our VAE, which will simultaneously train both the θ parameters of

the "decoder" network as well as the approximate posterior "encoder" ϕ parameters relating to $y, \mathbf{z}$.

2.2.2. Variational Objective with Labelled Data

When training with labelled data, we want to make sure we train both the y and the $\mathbf{z}$ networks at the same time and ensure that the y network makes use of the labelled data from $\mathbf{x}$ to y.

Below is the derivation of the variational objective with labelled data.

First, we'll re-write the factorization of our generative model from Equation 4 to explicitly show π:

$$p(\mathbf{x}, y, \mathbf{z}, \pi) = p(\mathbf{x}|y, \mathbf{z})p(\mathbf{z})p(y|\pi)p(\pi) \tag{7}$$

Notice a couple of things:

- In our generative model, our output ($\mathbf{x}$) only depends directly on $y, \mathbf{z}$, not π
- We are now emphasizing the relationship between y and π, where y depends on π

Next, we'll have to change our posterior approximation a bit:

$$\begin{aligned} q(\pi, \mathbf{z}|\mathbf{x}, y) &= q(\mathbf{z}|\mathbf{x})q(\pi|\mathbf{x}) \\ q(\pi|\mathbf{x}) &= \delta_{\pi_{q(y|\mathbf{x})}}(\pi) \end{aligned} \tag{8}$$

Notice that we're using $q(\pi|\mathbf{x})$ instead of $q(y|\mathbf{x})$. Recall, our approximation network $q(y|\mathbf{x}))$ is outputting the parameters for our categorical variable y, call it $\pi_{q(y|\mathbf{x})}$, this clearly does not define a distribution over π; it's actually just a point estimate of π. In order to get $q(\pi|\mathbf{x})$, we take this point estimate and assume it defines a Dirac delta distribution! In other words, it's density is zero everywhere except at a single point and its integral over the entire support is 1.

So now that we have re-defined our posterior approximation, we go through our ELBO equation as before:

$$\begin{aligned} \log p_\theta(\mathbf{x}, y) &\geq E_{q(\mathbf{z},\pi|\mathbf{x},y)}\Big[\log p_\theta(\mathbf{x}, y, \mathbf{z}, \pi) - \log q(\mathbf{z}, \pi|\mathbf{x}, y)\Big] \\ &= E_{q(\mathbf{z},\pi|\mathbf{x},y)}\Big[\log p_\theta(\mathbf{x}|y, \mathbf{z}) + \log p_\theta(y|\pi) \\ &\quad + \log p_\theta(\pi) + \log p_\theta(\mathbf{z}) - \log q(\mathbf{z}|\mathbf{x}) - \log q(\pi|\mathbf{x})\Big] \\ &= E_{q(\mathbf{z}|\mathbf{x})}\Big[\log p_\theta(\mathbf{x}|y, \mathbf{z}) + \log p_\theta(\mathbf{z}) - \log q(\mathbf{z}|\mathbf{x})\Big] \\ &\quad + E_{q(\pi|\mathbf{x})}\Big[\log p_\theta(y|\pi) + \log p_\theta(\pi) - \log q(\pi|\mathbf{x})\Big] \\ &= -\mathcal{L}(\mathbf{x}, y) + E_{q(\pi|\mathbf{x})}\Big[\log p_\theta(y|\pi)\Big] \\ &\quad - KL[q(\pi|\mathbf{x})||p_\theta(\pi)] + K_1 \end{aligned} \tag{9}$$

We expand our generative model and posterior approximation according to the factorizations in Equations 7 and 8, and then group them together into their respective expectations. Finally, we see that the $-\mathcal{L}(\mathbf{x}, y)$ terms appear as before along with a KL divergence term.

Note here, the KL divergence term is actually ∞ (by method of taking the limit implicit in the Dirac delta distribution) because the divergence between a symmetric Dirichlet distribution ($p(\pi)$) and a point estimate using a Dirac delta distribution ($q(\pi|\mathbf{x})$) is infinite. However, looking at it from another angle because we chose a Dirac delta distribution for the posterior approximation, the divergence will always be infinite. So since it is always infinite, we can ignore this term in our loss function.

$$\begin{aligned} &-\mathcal{L}(\mathbf{x}, y) + E_{q(\pi|\mathbf{x})}\Big[\log p_\theta(y|\pi)\Big] + K_1 - KL[q(\pi|\mathbf{x})||p_\theta(\pi)] \\ &\approx -\mathcal{L}(\mathbf{x}, y) + E_{q(\pi|\mathbf{x})}\Big[\log p_\theta(y|\pi)\Big] + K_1 \end{aligned} \tag{10}$$

Continuing on (after getting rid of the KL divergence term), we utilize our selection of $q(\pi|\mathbf{x})$ as a Dirac delta distribution:

$$
\begin{aligned}
&- \mathcal{L}(\mathbf{x}, y) + E_{q(\pi|\mathbf{x})}\Big[\log p_\theta(y|\pi)\Big] + K_1 \\
&= -\mathcal{L}(\mathbf{x}, y) + \int_{-\infty}^{\infty} q(\pi|\mathbf{x}) \log p(y|\pi) d\pi + K_1 \\
&= -\mathcal{L}(\mathbf{x}, y) + \int_{-\infty}^{\infty} \delta_{\pi_{q(y|\mathbf{x})}}(\pi) \log p(y|\pi) d\pi + K_1 \\
&= -\mathcal{L}(\mathbf{x}, y) + \log p(y|\pi_{q(y|\mathbf{x})}) + K_1 \\
&= -\mathcal{L}(\mathbf{x}, y) + \log[\prod_{i=1}^{K} (\pi_{q(y=i|\mathbf{x})})^{I(y=i)}] + K_1 \\
&= -\mathcal{L}(\mathbf{x}, y) + \log q(y = i|\mathbf{x}) + K_1 \\
&\approx -\mathcal{L}(\mathbf{x}, y) + \alpha \log q(y = i|\mathbf{x}) + K_1
\end{aligned} \tag{11}
$$

We can see the Dirac delta simplifies the expectation significantly, which just "filters" out the logarithm from the integral. Next, we expand out $p(y|\pi_{q(y|\mathbf{x})})$ with the PDF of a categorical variable at a given value of y (I is the indicator function). The indicator function essentially filters out the proportion for the observed y value, which is just the PDF of $q(y|\mathbf{x})$, our approximate posterior as required.

2.2.3. Training the M2 Model

Using Equations 6 and 9, we can derive the following loss function which is the negative of ELBO:

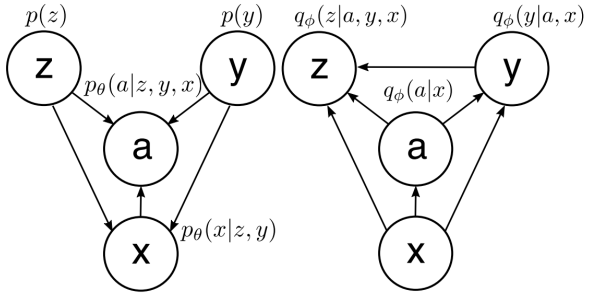
$$
\begin{aligned}
\mathcal{J} = &\sum_{\mathbf{x} \in \mathcal{D}_{unlabelled}} \Big[\sum_y q_\phi(y|\mathbf{x})(\mathcal{L}(\mathbf{x}, y)) - \mathcal{H}(q_\phi(y|\mathbf{x})) \Big] \\
&+ \sum_{(\mathbf{x}, y) \in \mathcal{D}_{labelled}} \Big[\mathcal{L}(\mathbf{x}, y) - \alpha \log q_\phi(y|\mathbf{x}) \Big]
\end{aligned} \tag{12}
$$

We train the network with this loss function by simply grab a mini-batch, computing the needed values in the network (i.e. $q(y|\mathbf{x}), q(z|\mathbf{x}), p(\mathbf{x}|y, z)$), computing the loss function above using the appropriate summation depending on if we have labelled or unlabelled data, and finally take the gradients to update our network parameters θ, ϕ. The network is similar to a vanilla VAE with the addition of the posterior on y, and the additional terms to the loss function with the difference ofcourse being the different treatments of the two types of data (labelled and unlabelled).

2.3. Auxiliary Deep Generative Models

From (Maale et al., 2016) The M2 architecture tries to solve the semisupervised learning problem by modeling the joint distribution over data and labels. The Kingma paper tries to extend this by having an hierarchichal model with a layer each of M1 and M2. This model, with more than one layer of stochastic latent variables is difficult to train end-to-end and required training layer by layer. We basically train M1 as a feature extractor first.

The Auxiliary Deep Generative Models (ADGM) utilize an extra set of auxiliary latent variables to increase the flexibility of the variational distribution. The auxiliary variable models can fit complex latent distributions and thereby improve the variational lower bound.



(a) Generative model P. (b) Inference model Q.

Figure 3. Probabilistic graphical model of the ADGM for semisupervised learning. The incoming joint connections to each variable are deep neural networks with parameters θ and ϕ.

The generative model is extended with variables a to $p(x, z, a)$ such that the original model is invariant to marginalization over a: $p(x, z, a) = p(a|x, z)p(x, z)$. In the variational distribution, on the other hand, a is used such that marginal $q(z|x) = \int q(z|a, x)p(a|x)\, da$ is a general non-Gaussian distribution. This hierarchical specification allows the latent variables to be correlated through a, while maintaining the computational efficiency of fully factorized models (Fig: 3).

Using the auxiliary variables a we extend the variational distribution: $q(a, z|x) = q(z|a, x)q(a|x)$ such that the marginal distribution $q(z|x)$ can fit more complicated posteriors $p(z|x)$. In order to have an unchanged generative model, $p(x|z)$, it is required that the joint mode $p(x, z, a)$ gives back the original $p(x, z)$ under marginalization over a, thus $p(x, z, a) = p(a|x, z)p(x, z)$. The auxiliary VAE lower bound becomes:

$$
\begin{aligned}
\log p(x) &= log \int_a \int_z p(x, a, z)\, da\, dz \\
&\geq E_{q_\phi}(a, z|x)\Big[\frac{log p_\theta(a|z, x) p_\theta(x|z) p(z)}{q_\phi(a|x) q_\phi(z|a, x)}\Big] \\
&\equiv -\mathcal{U}_{AVAE}(x)
\end{aligned} \tag{13}
$$

with $p_\theta(a|z,x)$ and $q_\phi(a|x)$ being diagonal Gaussian distributions parameterized by deep neural networks.

2.3.1. Semi-Supervised Learning Model

We now discuss how to use the auxiliary approach to build semi-supervised models that learn classifiers from labeled and unlabeled data. Similarly as in M2, to encompass the class information we introduce an extra latent variable y. The generative model P is defined as $p(y)p(z)p_\theta(a|z,y,x)p_\theta(x|y,z)$ (Fig: 3 a):

$$p(z) = N(z|0,I) \tag{14}$$
$$p(y) = Cat(y|\pi) \tag{15}$$
$$p(a|z,y,x) = f(a,z,y,x,\theta) \tag{16}$$
$$p(x|z,y) = f(z,z,y,\theta) \tag{17}$$

where a, y, z are the auxiliary variable, class label, and latent features, respectively. Cat(.) is a multinomial distribution, where y is treated as a latent variable for the unlabeled data points. $f(x;z,y,\theta)$ is iid categorical or Gaussian for discrete and continuous observations x. $p_\theta(.)$ are deep neural networks with parameters θ. The inference model is defined as $q_\phi(a|x)q_\phi(z|a,y,x)q_\phi(y|a,x)$ (Fig. 3 b):

$$q_\phi(a|x) = N\Big(a|\mu_\phi(x), diag(\sigma^2_\phi(x))\Big) \tag{18}$$
$$q_\phi(y|a,x) = Cat(y|\pi_\phi(a,x)) \tag{19}$$
$$q_\phi(z|a,y,x) = N\Big(z|\mu_\phi(a,y,x), diag(\sigma^2_\phi(a,y,x))\Big) \tag{20}$$

In order to model Gaussian distributions $p_\theta(a|z,y,x), p_\theta(x|z,y), q_\phi(a|x)$ and $q_\phi(z|a,y,x)$ we define two separate outputs from the top deterministic layer in each deep neural network, $\mu_{\phi\vee\theta}(.)$ and $\log\sigma^2_{\phi\vee\theta}(.)$. From these outputs we are able to approximate the expectations E by applying the reparameterization trick.

The key point of the ADGM is that the auxiliary unit a introduces a latent feature extractor to the inference model giving a richer mapping between x and y. We can use the classifier (20) to compute probabilities for unlabeled data x_u being part of each class and to retrieve a cross-entropy error estimate on the labeled data x_l. This can be used in cohesion with the variational lower bound to define a good objective function in order to train the model end-to-end.

2.3.2. Variational Lower Bound

We optimize the model by maximizing the lower bound on the likelihood. The variational lower bound on the marginal likelihood for a single labeled data point is

$$\begin{aligned}\log p(x,y) &= log\int_a\int_z p(x,y,a,z)\,dz\,da \\ &\geq E_{q_\phi}(a,z|x,y)\left[log\frac{p_\theta(x,y,a,z)}{q_\phi(a,z|x,y)}\right] \\ &\equiv -\mathcal{L}(x,y)\end{aligned} \tag{21}$$

with $q\_phi(a,z|x,y) = q_\phi(a|x)q_\phi(z|a,y,x)$. For unlabeled data we further introduce the variational distribution for y, $q_\phi(y|a,x)$:

$$\begin{aligned}\log p(x) &= log\int_a\int_y\int_z p(x,y,a,z)\,dz\,dy\,da \\ &\geq E_{q_\phi}(a,y,z|x)\left[log\frac{p_\theta(x,y,a,z)}{q_\theta(a,y,z|x)}\right] \\ &\equiv -\mathcal{U}(x)\end{aligned} \tag{22}$$

with $q_\phi(a,y,z|x) = q_\phi(z|a,y,x)q_\phi(y|a,x)q_\phi(a|x)$.

The classifier (20) appears in $-\mathcal{U}(x_u)$, but not in $\mathcal{L}(x_l,yl)$. The classification accuracy can be improved by introducing an explicit classification loss for labeled data:

$$\mathcal{L}_l(x_l,y_l) = \mathcal{L}(x_l,y_l) + \alpha * E_{q_\phi(a|x_l)}\Big[-\log q_\phi(y_l|a,x_l)\Big] \tag{23}$$

where α is a weight between generative and discriminative learning. The α parameter is set to $\beta * \frac{N_l+N_u}{N_l}$, where β is a scaling constant, N_l is the number of labeled data points and N_u is the number of unlabeled data points. The objective function for labeled and unlabeled data is

$$\mathcal{J} = \sum_{(x_l,y_l)}\mathcal{L}_l(x_l,y_l) + \sum_{(x_u)}\mathcal{U}(x_u) \tag{24}$$

3. Implementation Details

3.1. M1 Model

- $q(z|x)$ is estimated by a Deep Network with 3 conv layers, and 2 fully connected layers with batch normalization, dropout and ReLU activation.
- $p(x|z)$ is estimated with a fully connected layer, followed by 4 transposed conv layers (the first 3 with ReLU activation the last with sigmoid for the output).
- The classifier is a Support Vector Machine going from Z to Class Label

3.2. M2 Model

- $q(y|x)$ is estimated with a CNN which has 3 conv layers, 2 max pool layers, a softmax layer, with dropout and ReLU activation.
- $q(z|x)$ is estimated with 3 conv layers, and 2 fully connected layers with batch normalization, dropout and ReLU activation.
- $p(x|y,z)$ is estimated with a fully connected layer, followed by 4 transposed conv layers (the first 3 with ReLU activation the last with sigmoid for the output).

3.3. Auxiliary Deep Generative Model

The ADGM is parameterized by 5 neural networks (NN):

- Auxiliary Inference Model $q_\phi(a|x)$,
- Latent Inference Model $q_\phi(z|a,y,x)$,
- Classification Model $q_\phi(y|a,x)$
- Generative Model $p_\theta(a|.)$
- Generative Model $p_\theta(x|.)$

The neural networks consists of 2 fully connected hidden layers. All hidden layers use rectified linear activation functions. To compute the approximations of the stochastic variables we place two independent output layers after the final hidden layer, and $\log \sigma^2$.

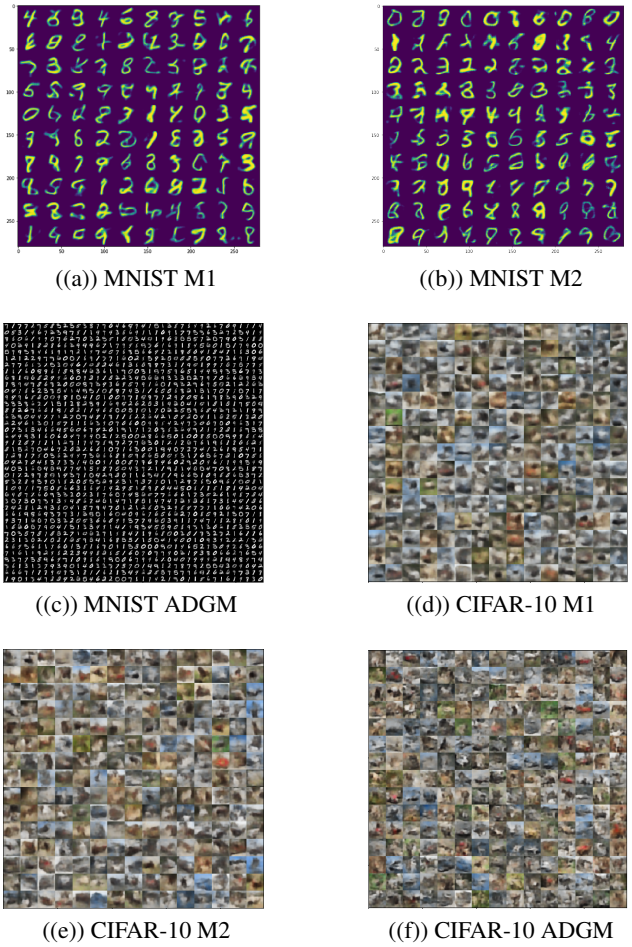
3.4. Inception

In order to compare performances on the Cifar-10 dataset, I used a pre-trained Inception network available in Pytorch. We add an extra global average pooling layer, a dense layer, followed by a softmax layer which are trained only on the labelled data while freezing all the original pre-trained Inception layers.

3.5. Datasets

We use the following datasets

- **MNIST**
  Training Set: 60000 images, with 10 classes. We use 1000 labelled images distributed equally amongst all classes.
  Test Set: 10000 images
- **CIFAR 10**
  Training Set: 50000 images, with 10 classes. We use 1000 labelled images distributed equally amongst all classes.
  Test Set: 10000 images



((a)) MNIST M1 ((b)) MNIST M2 ((c)) MNIST ADGM ((d)) CIFAR-10 M1 ((e)) CIFAR-10 M2 ((f)) CIFAR-10 ADGM

Figure 4. Randomly Generated Images

- **CIFAR 100**
  Training Set: 50000 images, with 100 classes. We use 1000 labelled images distributed equally amongst all classes.
  Test Set: 10000 images

3.6. Code Repository

One of the objectives of my project is to release an open source code repository with highly efficient and modular Pytorch implementations of the various VAE architectures. The code repository is located here https://github.com/ambujojha/SemiSupervisedLearning.git

4. Semi Supervised Learning Results

As the Table 1 shows, on the MNIST dataset, ADGM performs exceptionally well and the performance of M2 is pretty great as well. The very naive model M1 isn't too too bad either.

Table 1. Classification Accuracies for M1, M2 and ADGM on various data sets.

DATA SET	MODEL	ACCURACY
MNIST	M1	0.873
MNIST	M2	0.957
MNIST	ADGM	0.993
CIFAR 10	M1	0.347
CIFAR 10	M2	0.452
CIFAR 10	ADGM	0.605
CIFAR 10	INCEPTION	0.693
CIFAR 100	M1	0.161
CIFAR 100	M2	0.267
CIFAR 100	ADGM	0.418
CIFAR 100	INCEPTION	0.473

A good explorative estimate of the models ability to comprehend the data manifold, or in other words be as close to the posterior distribution as possible, is to evaluate the generative model. As Figures 4(a), 4(b) and 4(c) show, for M1, M2 and ADGM if we sample some randomly generated images, we get well defined images where the classification is clear.

However on Cifar10 and Cifar100 datasets its another story. The M1 and M2 accuracy is much poorer than the pre-trained Inception network. As shown in Figures 4(d), 4(e) its hard for us to figure out what the label should be.

However the classification accuracy of ADGM while still quite bad, starts approaching the pre-trained Inception Model.

Also as we can see in 4(f) ADGM's randomly generated images are much sharper than the ones generated by M1 or M2.

5. Comparison with Other Methods

Using a Discriminative rather than a generative approach, (Tarvainen & Valpola, 2017) have claimed an error percentage rate of 21.55 ± 1.48 on Cifar-10 with number of labelled images=1000. They propose an enhancement to the Temporal Ensembling (Laine & Aila, 2016) a pseudo labelling method which forms a consensus prediction of the unknown labels using the outputs of the network-in-training on different epochs and under different regularization and input augmentation conditions. This ensemble prediction can be expected to be a better predictor for the unknown labels than the output of the network at the most recent training epoch, and can thus be used as a target for training.

From a Generative Model perspective, (Wei et al., 2018) have claimed the best results so far with an error percentage rate of 9.98 ± 0.21(Number of labelled images=4000) and an accuracy of over 90%. They propose a novel approach to enforcing the Lipschitz continuity in the training procedure of Wasserstein GANs. Wasserstein GANs avoid the caveats in the minmax two-player training of GANs by using the 1-Lipschitz continuity of the discriminator.

5.1. Consistency Regularization

Presently the most successful approaches to semi-supervised learning are based on consistency regularization, whereby a model is trained to be robust to small perturbations of its inputs and parameters (Li et al., 2019) (Athiwaratkun et al., 2019). In the SSL literature, one of the key assumptions is the smoothness assumption where similar data are more likely to share the same label. Both the state of the art approaches mentioned add a consistency cost to the loss function. These methods use unlabeled data to stabilize their predictions under input or weight perturbations.

Consistency-enforcing methods can also be used at scale with state-of-the-art architectures. For example, the recent Mean Teacher model has been used with the Shake-Shake (Gastaldi, 2017) regularization and has achieved the best semi-supervised performance on the consequential CIFAR benchmarks (Athiwaratkun et al., 2019).

Image classification is a particularly suitable area for the consistency-based methods. This is because the convolutional structure regularizes the lower layers of the network well even with standard supervised learning on little data. Consequently the most useful role for unlabeled data is to train the upper layers of the network, and that's where the proxy target for the consistency cost helps the most. It is expected that in other domain areas, where the lower-level representations have a larger role, other semi-supervised approaches, such as GANs, VAEs, denoising auto-encoders, etc. become more important and might perform better. As indeed has been shown by the successful application of Generative Models to Natural Language Processing.

6. Conclusion

In this paper we have studied the application of Deep Generative Models of the Variational Auto Encoder type to the problem of Semi Supervised Learning on Image Classification on various open source datasets. We show with image classification because of locality, stationarity and compositionality of the image-data, use of consistency kind of loss functions leads to better results. These consistency based loss functions can be easily applied to Discriminative models or to GANs leading to better performances than VAEs.

References

Athiwaratkun, B., Finzi, M., Izmailov, P., and Wilson, A. G. 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.

Doersch, C. Tutorial on variational autoencoders, 2016. URL http://arxiv.org/abs/1606.05908. cite arxiv:1606.05908.

Kingma, D. P., Rezende, D. J., Mohamed, S., and Welling, M. Semi-supervised learning with deep generative models. *CoRR*, abs/1406.5298, 2014. URL http://arxiv.org/abs/1406.5298.

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

Li, Y., Liu, L., and Tan, R. T. Certainty-driven consistency loss for semi-supervised learning. *CoRR*, abs/1901.05657, 2019. URL http://arxiv.org/abs/1901.05657.

Maale, L., Snderby, C. K., Snderby, S. K., and Winther, O. Auxiliary deep generative models. In Balcan, M. F. and Weinberger, K. Q. (eds.), *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pp. 1445–1453, New York, New York, USA, 20–22 Jun 2016. PMLR. URL http://proceedings.mlr.press/v48/maaloe16.html.

Tarvainen, A. and Valpola, H. Weight-averaged consistency targets improve semi-supervised deep learning results. *CoRR*, abs/1703.01780, 2017. URL http://arxiv.org/abs/1703.01780.

Wei, X., Liu, Z., Wang, L., and Gong, B. Improving the improved training of wasserstein GANs. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=SJx9GQb0-.