Semi-Supervised Learning with Deep Generative Models
S. Saemundsson, Y. Gao, S. Popescu
Department of Engineering, University of Cambridge

Objective
The motivation behind reproducing the chosen paper was a better understanding of semi-supervised learning methods and stochastic variational inference. At the onset of the project our goals were as follows:
- Implement our own version of the models found in (1) using TensorFlow (2).
- Evaluate their performance on the MNIST dataset and contrast with the original paper.
- Ultimately the goal was to reproduce the results obtained using the stacked deep-generative model of (1) with only 100 labelled examples.

Introduction
Ever-increasing unlabelled data together with prohibitive costs of manually labelling it calls for models that are able to efficiently and accurately generalise from a small number of examples. The framework described in (1) formulates the semi-supervised learning problem as a generative one and uses deep neural networks to parameterise the densities of the data classes as well as intra-class variability. Possible domains with high-impact from semi-supervised learning can be established by combining model 1 and model 2: 

\[ p(x, y, z; \theta) = p(y)p(z)p(y|x)p(z|x) \]  

(2)

where the priors are the same as in the individual models and both \( p(y)p(z; y, z) \) and \( p(x|z) \) are parameterised by neural networks. In practice this is done by first learning a latent representation using M1 which is used as input into the neural network for M2.

Variational Inference
In all our models, computation of the exact posterior distribution is intractable due to the nonlinear, non-conjugate dependencies between the random variables. Therefore using variational inference is an essential approach to allow for tractable and scalable inference and parameter learning. This is done using a variational approximation of the posterior \( q(\theta) \). The objective to minimise in the case of M1 can be shown to be:

\[ \log p(x, y, z; \theta) \geq -U(\theta) + KL(q(\theta || p(\theta))) \]  

(3)

In the case of M2 there are two distinct types of datapoints (i.e. labelled and unlabelled):

\[ \log p(x, y; \theta) \geq -L(x, y) = \text{E}_q[\log p(z|x, y) + \log p(y|x) - q(z|x, y)] \]  

(4)

\[ \log p(x; \theta) \geq -U(x) = \sum_y q(y|x)(-L(x, y) + H(q(y|x))) \]  

(5)

The bound on the marginal likelihood for the entire data is then:

\[ J = \sum_s L(x, y) + \sum_s U(x) \]  

(6)

Additionally, in order for the predictive distribution \( q_0(y|x) \) to contribute also to the term pertaining to the unlabelled data, a hyperparameter \( \alpha \) was introduced whereby:

\[ J^\alpha = J + \alpha \cdot E_{p(\theta)}[-\log q_0(y|x)] \]  

(7)

This hyperparameter can be shown to be equivalent to placing a symmetric Dirichlet prior over the \( \pi \) parameters of the categorical distribution.

Experiments
The dataset was split into 5000/5000/10000 datapoints for training, validation and evaluation respectively. Each result with a set number of labelled examples was run 3 times for 1200 epochs of the data using different initialisations to obtain the average in Table 1. The data was further split into 100 batches containing a fixed number of labelled examples (e.g. 10 in each batch for a 1000 labelled examples). The networks in M1 had two fully connected 500 unit hidden layers. In M2 there was only one hidden layer for each network with 500 hidden units. In both cases the hidden units had softplus activations. Gradient ascent was performed using ADAM (3) with a learning rate of \( 3 \times 10^{-4} \), and decay rate parameters \( \beta_1 = 0.9, \beta_2 = 0.999 \). The weights were initialised at random and in the case of M1, L2 regularisation was used with weight parameter \( 0.001 \). For M2 the weights were conditioned with a standard normal gaussian prior.

Discussion
The results in Table 1 are slightly lower than that of the original paper. The discrepancy is likely caused by one of two factors: i) difference in implementation (e.g. L2 regularisation), ii) due to time constraints the models were run for only 1200 epochs until approximate convergence. TensorFlow was found to be highly useful in terms of balancing out-of-the-box components and flexibility which allowed for quick prototyping.

References