10-708: Probabilistic Graphical Models 10-708, Spring 2017

19: Inference and learning in Deep Learning

Lecturer: Zhiting Hu *Scribes: Akash Umakantha, Ryan Williamson*

1 Classes of Deep Generative Models

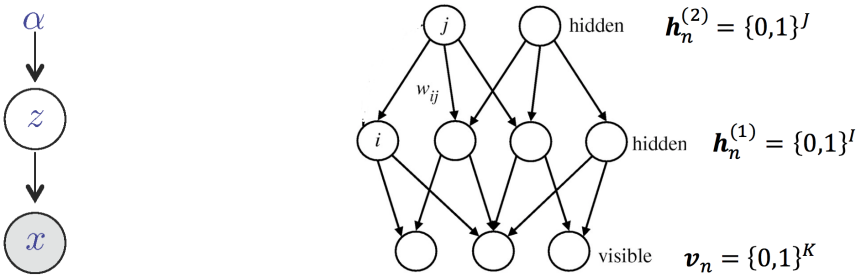
Explicit probabilistic models

Explicit models are one class of probabilistic generative models. They provide an explicit parametric specification of the data distribution and have tractable likelihood functions. Examples of explicit models range from simple models such as those shown in Fig 1a to more complex models such as latent Dirichlet allocation. In Fig 1a, the explicit probability model is

$$p(x, z|\alpha) = p(x|z)p(z|\alpha)$$

Sigmoid belief networks (Fig 1b) and variational auto-encoders (section 4) are examples of explicit deep generative models. The explicit probability model for a sigmoid belief net is:

$$p(v_{kn} = 1|w_k, h_n^{(1)}, c_k) = \sigma(w_k^T h_n^{(1)} + c_k)$$
$$p(h_{in}^{(}1)|w_i, h_n^{(2)}, c_i) = \sigma(w_i^T h_n^{(2)} + c_i)$$



(a) Simple explicit model

(b) Sigmoid belief network

Figure 1: Examples of explicit probabilistic models.

Implicit probabilistic models

Implicit probabilistic models do not specify the distribution of the data itself, but rather define a stochastic process that, after training, aims to draw samples from the underlying data distribution (i.e. simulates the data). Since they do not specify any distribution, implicit models do not require a tractable likelihood function. Generative adversarial networks (section 5), or GANs, are examples of deep generative models that are implicitly defined.

2 Recap of variational inference

Recall that for a probabilistic model $p_\theta(x, z)$ and variational distribution $q_\phi(z|x)$, we can derive the variational lower bound $L(\theta, \phi; x)$ for the data log likelihood:

$$\begin{aligned} p_\theta(x) &= KL(q_\phi(z|x) \parallel p_\theta(z|x)) + \int_z q_\phi(z|x) \log \frac{p_\theta(x, z))}{q_\phi(z|x)} \\ &\geq \int_z q_\phi(z|x) \log \frac{p_\theta(x, z))}{q_\phi(z|x)} \\ &:= L(\theta, \phi; x) \end{aligned}$$

The goal of variational inference is to minimize the KL divergence between the posterior and the variational distribution. This is equivalent to minimizing the free energy F:

$$F(\theta, \phi; x) = -\log p(x) + KL(q_\phi(z|x) \parallel p_\theta(z|x))$$

3 Wake sleep algorithm

Hinton et al. (Science 1995) proposed the wake sleep algorithm (2), a two-phase algorithm for training deep generative networks such as sigmoid belief networks. The general framework is to train an inference network (also called recognition network) for the variational distribution, $q_\phi(z|x)$ that in turn helps train the generative model, $p_\theta(x|z)$. The generative model starts with some prior $p(z)$ that is updated by the recognition network.

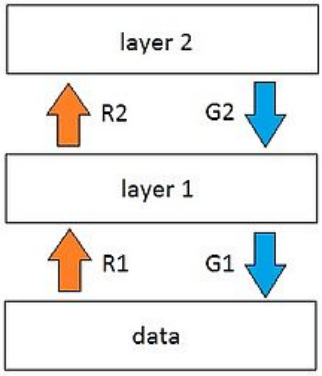


Figure 2: Diagram of the wake sleep algorithm. The R variabiles represent the directionality and weights of the recognition network, while G variables represent the directionality and weights of the generator network.

1. In the wake phase, the goal is to minimize free energy with respect to the generator network, $p_\theta(x|z)$. This is done by generating samples of the latents z by passing training data through the recognition network, and then using these samples to train the generator.

2. In the sleep phase, the goal is to minimize free energy with respect to the recognition network, $q_\phi(z|x)$. However, in practice this optimization is computationally intensive and has high variance. Thus, the original free energy is replaced by a pseudo free energy term, F', that is more suitable.

$$F'(\theta, \phi; x) = -\log p(x) + KL(p_\theta(z|x) \parallel q_\phi(z|x))$$

Now, simulated samples of the data x are generated using the generator network and latent z from the wake phase. These simulated samples are then used to train the recognition network.

There are several limitations to the wake sleep algorithm. First, since KL divergence is not symmetric (i.e., $KL(p\|q) \neq KL(q\|p)$), the pseudo free energy approximation in the sleep phase is imprecise, so optimization during the sleep phase is also imprecise. Additionally, the wake sleep algorithm does not generalize well if we have a well-defined objective function. Finally, wake sleep does not provide any convergence guarantees.

4 Variational Auto-encoders (VAEs)

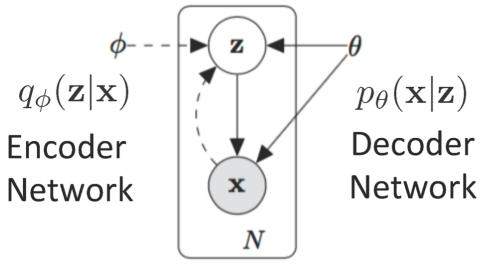


Figure 3: Graph of a variational auto-encoder.

Variational auto-encoders (Kingma and Welling, 2014), or VAEs, can be applied to similar models as the wake sleep algorithm, with the exception of models that have discrete latent variables. In a VAE, the decoder is the generative network for $p_\theta(x|z)$ with prior $p(z)$, and the encoder is the inference/recognition network for $q_\phi(z|x)$ (Fig 3). VAEs, like wake sleep, also optimize a lower bound (L) on the data log-likelihood:

$$\log p(x) \geq E_q[\log p] - KL(q \parallel p) := L(\theta, \phi; x)$$

The first step in training a VAE is to optimize L with respect to the generator network p_θ, which can be done in a similar manner as the wake phase from earlier. However, we again run into a problem of high variance in gradient estimates when trying to train the recognition newtork. Kingma and Welling's key insight is to deal with this issue by using a clever reparameterization trick on the recognition distribution.

The trick is to make the latent variable z a deterministic variable by introducing an auxiliary random noise variable ϵ. Given the recognition network, the latents, which originally were described by the distribution $z \sim q_\phi(z|x)$, can be written as a deterministic function of the input data and the noise variable $z = g_\phi(\epsilon, x)$, where the randomness is now in the noise variable $\epsilon \sim p(\epsilon)$. For example, $z \sim N(\mu, \sigma^2)$ can be rewritten as $z = \mu + \sigma\epsilon$, with $\epsilon \sim N(0, I)$.

Now, the first term of the lower bound $E_q[\log p]$ can be rewritten as an expectation with respect to ϵ rather than the variational distribution itself. This allows gradients to be taken directly with respect to the latent variables, i.e.

$$\nabla_\phi E_{q_\phi(z|x)}[\log p_\theta(x, z)] = E_{\epsilon \sim N(0,I)}[\nabla_\phi \log p_\theta(x, z_\phi(\epsilon))]$$

Indeed, for Gaussian distributions, $KL(q\|p)$ can be computed analytically and these gradients can be computed analytically as well.

In practice, VAEs obtain much higher data log-likelihood than the wake sleep algorithm. However, the images generated by VAEs tend to be somewhat blurry relative to realistic images.

Limitations and variants

For images, VAEs have classically used pixel-wise reconstruction error, so the network is sensitive to irrelevant variations (such as translations, rotations, obstructions) in the training data. One possible solution is to use feature-wise (perceptual-level) reconstruction error (for example by pre-training a nerual network to extract features).

Another limitation of VAEs is that the reparameterization trick only works for continuous latent variables z. One solution is to marginalize out the discrete latents, but this can become expensive if the latent space is large. Another solution is to use a continuous approximation to discrete latents (e.g. Gumbel-softmax for multinomial variables).

Finally, the prior $p(z)$ in VAEs is typically just a Gaussian because this allows for easier inference and learning. However, this is limiting because this simple, single-mode latent distribution must account for all the complexities of the original data distribution. Goyal et al. (2017) propose to instead use hierarchical nonparametric priors over the latents.

5 Generative adversarial networks (GANs)

GANs were first introduced in Goodfellow 2014 as an implicit generative model in which the cost function can be interpreted as a minimax game between a generator and a discriminator (See Figure 4). One notable advantage of these models over other generative models is their ability to generate qualitatively sharp, high-fidelity samples (e.g., images). The model is defined as follows:

Generator: maps noise variable to data space

$$x_n = G(\mathbf{z}_n, \theta_g);\ \mathbf{z}_n \sim N(\mathbf{0}, \mathbf{I})$$

Discriminator: outputs the probability that $\mathbf{x}$ came from the data rather than the generator

$$D(\mathbf{x}, \theta_d) \in [0, 1]$$

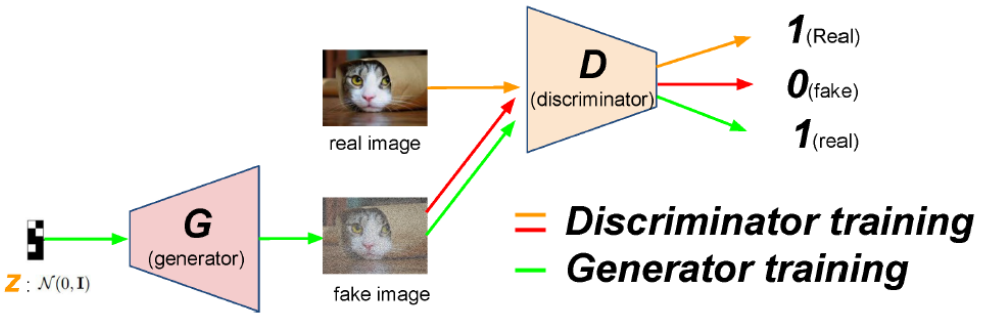


Figure 4: Diagram of the Generative Adversarial Network.

5.1 Learning

Conceptually, the goal of learning is to maximize the expectation that the discriminator, D, correctly categorizes the data as either real or fake. The goal of learning for the generator, G, is to fool the discriminator.

Mathematically, the goal of learning is to minimize the following objective function:

$$\min_g \max_D V(D,G) = \mathbb{E}_{x\sim p_{data}(x)}[\log D(x)] + \mathbb{E}_{z\sim p_z(z)}[\log(1 - D(G(z)))]$$

where $D(x)$ assigns a label of 1 for real data and a label of 0 for fake. In practice, training is alternated between training D and training G.

5.2 Theory

Many of the theoretical results for GANs assume that the discriminator is optimal. For fixed G, the optimal discriminator, D, is defined by:

$$D^*_G(x) = \frac{p_{data}(x)}{p_{data}(x) + p_g(x)}$$

With this assumption, it is possible to identify the optimal objection function value for all GANs as follows:

$$\begin{aligned}
C(G) &= \min_g \max_D V(D,G) \\
&= \mathbb{E}_{x\sim p_{data}(x)}[\log D(x)] + \mathbb{E}_{z\sim p_z(z)}[\log(1 - D(G(z)))] \\
&= \mathbb{E}_{x\sim p_{data}(x)}[\log D(x)] + \mathbb{E}_{z\sim p_z(z)}[\log(1 - D(z))] \\
&= \mathbb{E}_{x\sim p_{data}(x)}[\log \frac{p_{data}(x)}{p_{data}(x) + p_g(x)}] + \mathbb{E}_{z\sim p_z(z)}[\log(\frac{p_g(x)}{p_{data}(x) + p_g(x))}] \\
&= -\log(4) + KL\left(p_{data}\middle|\middle|\frac{p_{data} + p_g}{2}\right) + KL\left(p_g\middle|\middle|\frac{p_{data} + p_g}{2}\right) \\
&= -\log(4) + 2 \cdot JSD(p_{data}||p_g)
\end{aligned}$$

Therefore $C^* = -\log 4$ is the global minimum and the only solution is $p_g = p_{data}$

5.3 GANs in Practice

In practice, GANs require some adjustments to make optimization feasible. Here are a few examples:

Optimizing D: Optimizing D to completion is computationally prohibitive. Therefore, typical training for D occurs for k steps per step of G optimization.

Vanishing gradient for G: the objective for G suffers from a vanishing gradient when D is too close to the optimal value. Therefore the objective is typically altered as follows:

$$\min_G E_z[\log 1 - D(G(Z))] \rightarrow \max_G E_z[\log D(G(z))]$$

Instability of training: Optimization requires a careful balance between D and G. It has been shown that $E_z[\log D(G(z))]$, the steep gradient objective for G, can be centered on a Cauchy distribution with infinite expectation and variance.

Mode collapsing: Generated samples often represent only a few modes of the data distribution (e.g., only a few letters may be represented from the MNIST data set)

5.4 Applications

Examples of GAN applications include generating images [Goodfellow, 2014], translating images [Isola et al., 2016], domain adaptation [e.g., Purushotham et al., 2017], imitation learning [Ho & Ermon 2016]

5.5 Limitations and variants

There are several limitations of GANs. First, they do not support inference, so there is no mechanism for inferring z from x. However, a few variants exist to overcome this problem.

A larger issue with GANs is their inability to use discrete inputs. This stems from the fact that the gradient must backpropagate through the discriminator to the generator. This cannot be done if the input layer of the discriminator is discrete. One GAN variant works around this issue by treating the generator training as a form of policy learning [Yu et al., 2017], although this method suffers from high variance and slow convergence. Another variant utilizes continuous approximations of discrete variables [Kusner & Hemndez-Lobato, 2016]; however this method has only demonstrated qualitative results for small discrete inputs. Finally, another approach to incorporating discrete variables is to combine VAEs (which can handle discrete inputs) and GANs, using a wake-sleep style learning approach.

Another issue with GANs is the lack of intuition for controlling attributes in generated images. This is part of a larger problem of poor interpretability of latent variables. This is a common issue among deep generative models, including VAEs. One approach to overcoming this has been to enforce disentangled hidden codes by adding a mutual information regularizer [Chen et al., 2016]. This approach is limited in that it is unsupervised and therefore the semantics of each latent dimension are only observed after training and cannot be specified beforehand by the user. Other approaches have recently been proposed that allow for supervised enforcement of semantic constraints in certain latent dimensions [Odena et al., 2017; Hu et al., 2017].

6 Harnessing DNNs with Logic Rules

Typical neural networks rely heavily on large data-sets and are not well-suited to incorporating heuristics like human intention, common sense, or domain knowledge. In contrast, humans learn from both concrete examples (similar to neural networks) and general knowledge and experiences. For example, humans do not learn how to conjugate the past tense of verbs by memorizing examples, but by learning rules that can be applied to new examples.

6.1 Rule knowledge distillation

One approach to incorporating knowledge into neural networks is to provide the network with access to logic rules [Hu, 2016]. This approach has been successfully applied to a sentiment classification task, showing superior performance compared to other methods. Logic rules provide a flexible declarative language that facilitates expression of structured knowledge. More formally, our goal is the incorporate soft first-order logic (FOL) rules that consists of a (1) rule, $r(X, Y) \in [0, 1]$ that maps an input target space (X, Y) to 0 or 1 and (2) a confidence level, λ. This is incorporated into a neural network $p_\theta(y|x)$ which is designed to imitate the outputs of a rule-regularized teacher network. This approach has been referred to as distillation.

Training for the neural network occurs as follows:

$$\theta^{t+1} = \arg\min_{\theta\in\Theta} \frac{1}{N}\sum_{n=1}^{N}(1-\pi)\ell(y_n, \sigma_\theta(x_n)) + \pi\ell(s_n^{(t)}, \sigma_\theta(x_n))$$

where the parameters are defined as:

- y_n: true hard label
- $\sigma_\theta(x_n)$: soft prediction of p_θ
- π: balancing parameter
- $s_n^{(t)}$: soft prediction of the teacher network

The teacher network $q(Y|X)$ fit the logic rules $E_q[r(X,Y)] = 1$ with confidence λ. Optimization occurs in parallel with training of the neural network as follows:

$$\min_{q,\xi\geq 0} KL(q||p_\theta(Y|X)) + C\sum_t \xi_l$$
$$s.t.\ \lambda_l(1 - \mathbb{E}_q[r_l(X,Y)]) \leq \xi_l$$
$$l = 1, ..., L$$

Where

- $r_l(X,Y)$: are rule constraints
- ξ_l: are slack variables

This minimization has a closed form solution of:

$$q^*(Y|X) \propto p_\theta(Y|X)\exp\left\{-\sum_l C\lambda_l(1 - r_l(X,Y))\right\}$$

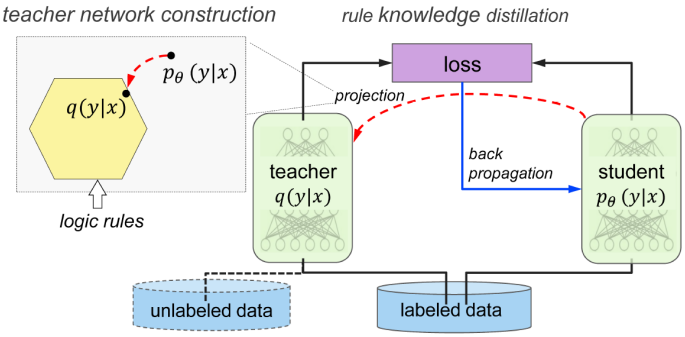


Figure 5: Diagram of training the teacher and student networks

Training at each iteration occurs by constructing a teacher network through posterior constraints and training the neural network to emulate the predictions of the teacher (see Figure 5).

7 Summary

- Explicit generative models specify the form of the data distribution and have tractable likelihoods. Implicit generative models specify a stochastic process to simulate data.
- The wake sleep algorithm is useful for training deep generative models.
  - In the wake phase, samples obtained from the recognition network are used to train the generator network.
  - In the sleep phase, samples simulated from the generator network are used to train the recognition network.
- Variational auto-encoders (VAEs) can be applied to similar models as wake sleep algorithm (except models with discrete latent variables).
  - The first step for training VAEs is similar to the wake phase.
  - The second step uses a re-parameterization trick to make latent variables deterministic so that expectations are with respect to a noise distribution. This allows gradients to be taken directly with respect to latent variables.
- GANs are useful for generating sharp, high-fidelity images.
  - GAN training can be interpreted as a minimax game between a generator network and a discriminator network.
  - GANs suffer from major limitations in that training can be unstable, they tend to express only a limited number of modes, and they cannot use discrete inputs.
- A general limitation of both GANs and VAEs is that it is difficult to control attributes in the generated images
- Logic rules can be incorporated until DNN using a student-teacher distillation training method.