Graphical Models For Complex Health Data (P8124)

Daniel Malinsky

Columbia University
dsm2128@cumc.columbia.edu

Deep Learning and Graphical Models

What is deep learning?

Methods that are "deep" typically involve sequentially stacking or nesting many simple models together ("multiple layers"), such that the overall stacked model can be very complex, highly flexible, and good for nonlinear problems.

Ex: single-hidden-layer neural network vs deep neural network

Neural networks

Inputs: $X = (X_1, ..., X_p)$
Hidden units: $Z = (Z_1, ..., Z_M)$
Outputs: $Y = (Y_1, ..., Y_K)$

$$Z_m = \sigma(\alpha_{0m} + \alpha_m^T X) \quad \forall m = 1, ..., M$$

$$T_k = \beta_{0k} + \beta_k^T Z \quad \forall k = 1, ..., K$$

$$\mathbb{E}[Y_k] = f_k(X) = g_k(T) \quad \forall k = 1, ..., K$$

where $\sigma(\cdot)$ is some *activation function* which might be $\sigma(v) = \frac{1}{1+e^v}$ (sigmoid/logistic), or $\sigma(v) = \tanh(v)$, or $\sigma(v) = \max(0, v)$ (ReLU)

and g_k is final transformation function, e.g. identity T_k or softmax $\frac{e^{T_k}}{\sum_{l=1}^K e^{T_l}}$

Neural networks

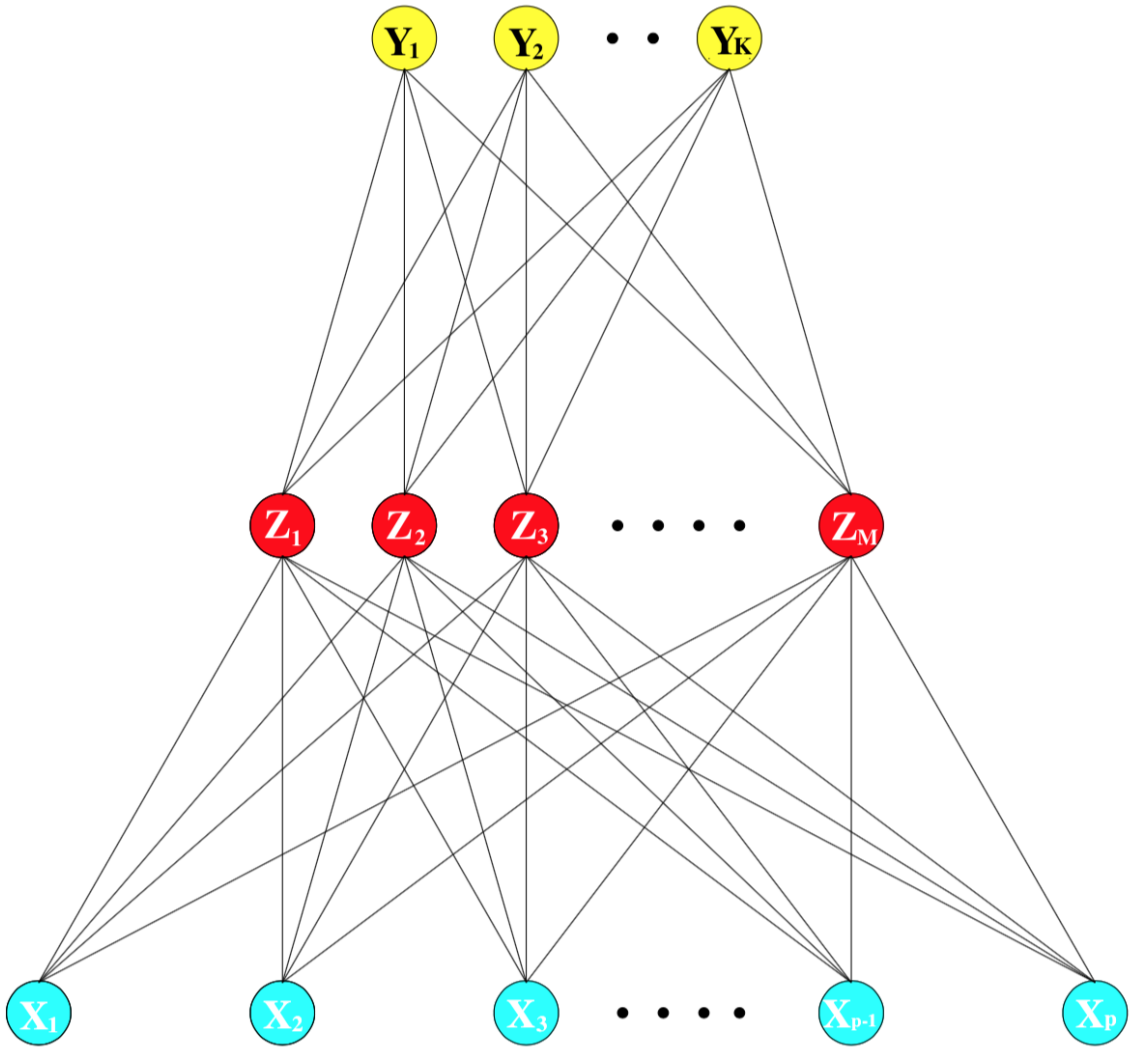


FIGURE 11.2. *Schematic of a single hidden layer, feed-forward neural network.*

a.k.a. multilayer perceptron. (Often drawn undirected, but really a directed acyclic graph... why?)

Neural networks

Deep networks have many layers of hidden units:

$$Z_{1m} = \sigma(\alpha_{01m} + \alpha_{1m}^T X) \quad \forall m = 1, ..., M_1$$

$$Z_{dm} = \sigma(\alpha_{0dm} + \alpha_{dm}^T Z_{d-1}) \quad \forall d = 2, ..., D, m = 1, ..., M_d$$

$$T_k = \beta_{0k} + \beta_k^T Z_D \quad \forall k = 1, ..., K$$

$$\mathbb{E}[Y_k] = f_k(X) = g_k(T) \quad \forall k = 1, ..., K$$

Neural networks

- Are called "universal approximators" b/c can approximate any suitably smooth function w/ sufficient # of hidden units
- Can come in different architectures (recurrent NN, convolutional NN) for different purposes
- Often have millions of parameters (highly overparameterized, not identifiable)
- Often require lots of data and computational resources to train $\Rightarrow$ so much work on this!

A note on nonidentifiability...

Multiple different settings of the NN parameter values may imply the same likelihood (or same prediction risk). This makes traditional statistical inference for those parameters impossible $\Rightarrow$ no guarantee that $\hat{\theta} \rightarrow^{P} \theta^*$ for some "true value" θ^*.

- In NN models, the various weight parameters are typically not of scientific interest in themselves. They don't correspond to quantities with real scientific meanings, but are rather instrumentally useful in parameterizing a highly nonlinear function.
- Multiple minimizers of the prediction risk might be ok ("as long as prediction risk is low, that is sufficient for prediction")...
- but may also cause problems (e.g., specific predictions for data points or subgroups may be quite different for distinct minimizers, even if average risk is the same. Might lead to instability and unreliable performance in deployment. See this recent paper: [link].)

Training NNs: backpropagation

The parameters of a NN are typically chosen to minimize risk (expected loss) for some choice of loss function, e.g.:

$$R(\theta) = \sum_{j=1}^{n} \sum_{k=1}^{K} (y_k^j - f_k(x^j))^2$$

or

$$R(\theta) = -\sum_{j=1}^{n} \sum_{k=1}^{K} y_k^j \log f_k(x^j)$$

depending on if the outcomes are continuous or binary (regression vs. classification).

To prevent overfitting, some regularization is needed, e.g., minimize $R(\theta) - \lambda J(\theta)$ for some $\lambda > 0$ and choice of penalty function $J(\theta)$.

Training NNs: backpropagation

Typically risk is minimized by calculating the gradient of R^j ($R(\theta) = \sum_{j=1}^{n} R^j$) wrt params, and iteratively updating with gradient descent.

For one-hidden-layer network w/ squared error loss (ignoring penalty):

$$\frac{\partial R^j}{\partial \beta_{km}} = -2(y_k^j - f_k(x^j))g_k'(\beta_k^T z^j)z_m^j$$

$$\frac{\partial R^j}{\partial \alpha_{ml}} = -\sum_{k=1}^{K} -2(y_k^j - f_k(x^j))g_k'(\beta_k^T z^j)\beta_{km}\sigma'(\alpha_m^T x^j)x_l^j$$

This is an application of the chain rule for derivatives and is called *backpropagation*.

Training NNs: backpropagation

Update equations:

$$\beta_{km}^{t+1} = \beta_{km}^{t} - \gamma^t \sum_{j=1}^{n} \frac{\partial R^j}{\partial \beta_{km}^{t}}$$

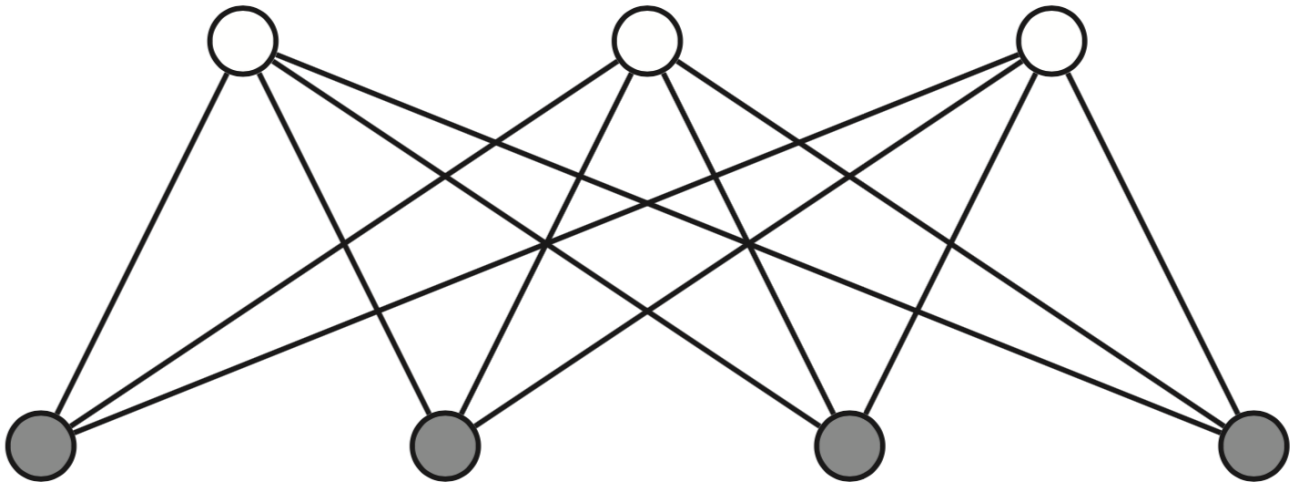
$$\alpha_{ml}^{t+1} = \alpha_{ml}^{t} - \gamma^t \sum_{j=1}^{n} \frac{\partial R^j}{\partial \beta_{ml}^{t}}$$

w/ learning rate γ^t.

Most often people use stochastic gradient decent (SGD), which is an optimization approach that is scaleable to very large data sets, in addition to all sorts of optimization tricks to tackle this optimization problem. (Won't get into it here...)

Restricted Boltzmann Machine (RBM)

A latent variable model inspired by the "layered" structure of a NN:



$$p(h, x) = \frac{1}{Z} \prod_{i=1}^{p} \prod_{m=1}^{M} \phi_{im}(x_i, h_m)$$

Here the hidden variables are denoted by $H = (H_1, ..., H_M)$ and observed are $X = (X_1, ..., X_p)$. An RBM is restricted to form an undirected bipartite graph. Special case of pairwise MRF.

RBMs

Typically latent variables in an RBM are binary. One reason RBMs are attractive:

$$p(h|x) = \prod_{m=1}^{M} p(h_m|x)$$

so it is easy to do "inference" for the latent states, e.g., sampling.

Binary RBMs

$$p(x, h; \theta) = \frac{1}{Z(\theta)} \exp(-E(x, h; \theta))$$

$$-E(x, h; \theta) = -\sum_{i=1}^{p} \sum_{m=1}^{M} x_i h_m W_{im} - \sum_{i=1}^{M} x_i b_i - \sum_{m=1}^{M} h_m c_m$$

$$= -(x^T W h + x^T b + h^T c)$$

$$Z(\theta) = \sum_{x} \sum_{h} \exp(-E(x, h; \theta))$$

for notational simplicity, can also collect the bias terms b, c in an expanded W matrix. The model implies:

$$p(h|x; \theta) = \prod_{m=1}^{M} p(h_m|x; \theta) = \prod_{m=1}^{M} \text{Bern}(h_k|\text{expit}(w_{:,k}^T x))$$

$$p(x|h; \theta) = \prod_{i=1}^{p} p(x_i|h; \theta) = \prod_{i=1}^{p} \text{Bern}(x_i|\text{expit}(w_{i,:}^T h))$$

Gaussian RBMs

For $X \in \mathbb{R}^p$ it is common to use a Gaussian RBM (σ^2 fixed to 1, data assumed standardized):

$$-E(x, h; \theta) = -\sum_{i=1}^{p}\sum_{m=1}^{M} x_i h_m W_{im} - \frac{1}{2}\sum_{i=1}^{M}(x_i - b_i)^2 - \sum_{m=1}^{M} h_m c_m$$

leads to

$$\text{logit } p(h_m = 1|x; \theta) = c_m + \sum_i w_{im} x_i$$

$$p(x_i|h; \theta) \sim N(b_i + \sum_m w_{im} h_m, 1)$$

Learning a RBM

The gradient of log-likelihood $\ell(\theta)$ for learning RBM params W looks like:

$$\frac{\partial \ell(\theta)}{\partial w_{im}} = \frac{1}{n} \sum_{j=1}^{n} \mathbb{E}[x_i h_m | x^j; \theta] - \mathbb{E}[x_i h_m; \theta]$$

If we can calculate this gradient, can initialize params randomly and update param estimates in the usual iterative fashion. But how to calculate these expectations?

There are a few different ways, using mean-field inference or Gibbs sampling. The most common method is using the *contrastive divergence* algorithm. Effectively, this approximates $\mathbb{E}[x_i h_m; \theta]$ by a single Gibbs sample from the model.

Learning a RBM

Algorithm 0.1: CONTRASTIVE DIVERGENCE$(\cdot)$

Input: Samples of the vector X^j $(j = 1, ..., n)$, learning rate γ
Output: Estimates of W, b, c

1. Initialize $\theta^0 = (W^0, b^0, c^0)$ randomly.
2. **for all** $j = 1, ..., n$
3. $\quad x^0 \leftarrow x^j$
4. $\quad h^0 \sim p(h|x^0; \theta^{j-1})$ Gibbs sampling
5. $\quad x^1 \sim p(x|h^0; \theta^{j-1})$ Gibbs sampling
6. $\quad \Delta W \leftarrow x^0[p(h = 1|x^0)]^T - x^1[p(h = 1|x^1)]^T$
7. $\quad \Delta b \leftarrow x^0 - x^1$
8. $\quad \Delta c \leftarrow p(h = 1|x^0) - p(h = 1|x^1)$
9. $\quad W^j \leftarrow W^{j-1} + \gamma \Delta W$
10. $\quad b^j \leftarrow b^{j-1} + \gamma \Delta b$
11. $\quad c^j \leftarrow c^{j-1} + \gamma \Delta c$

Note: since distributions factorize $p(h|x) = \prod_{m=1}^{M} p(h_m|x)$, can sample vector h from each conditional separately. Same for $p(x|h) = \prod_{i=1}^{p} p(x_i|h)$

How are RBMs used?

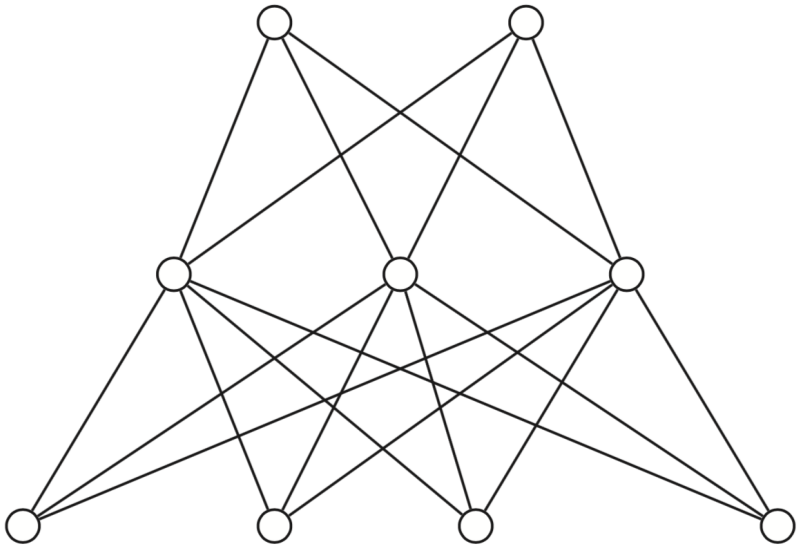
- Sometimes used as classifiers: compute $p(y|x)$ based on the LV model ("ClassRBM"). Here Y is just another "visible" node along with X. In some settings achieves similar or better performance than SVMs, RFs, NNs.
- Sometimes used as a "feature extractor," i.e., estimate H and use these as input into another task (prediction or clustering). This is a kind of dimension reduction.
- Sometimes used to generate samples $\sim p(h|x)$.

Deep Boltzmann Machines

RBMs can be stacked together to make Deep Boltzmann Machines with many layers of hidden nodes

$$p(h_1, h_2, \ldots, h_d, x) =$$
$$\frac{1}{Z} \exp\left(\sum_{im} x_i h_{1m} W_{1im} + \sum_{jk} h_{1j} h_{2k} W_{2jk} + \cdots + \sum_{rs} h_{(d-1)r} h_{ds} W_{drs}\right)$$

ignoring single-unit "bias" terms here.



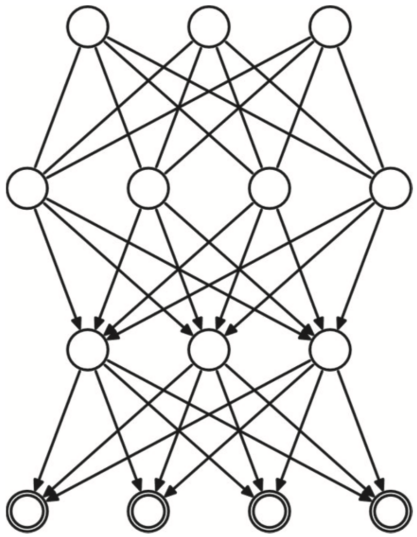
Deep Belief Networks

Closely related to DBMs are Deep Belief Networks, which are mixed directed + undirected models.

$$p(h_1, h_2, \ldots, h_d, x) =$$

$$\prod_i \text{Bern}(x_i|\text{expit}(h_1^T w_{1i})) \prod_j \text{Bern}(h_{1j}|\text{expit}(h_2^T w_{2j})) \times \cdots \times$$

$$\times \prod_k \text{Bern}(h_{(d-2)k}|\text{expit}(h_{(d-1)}^T w_{(d-1)k})) \frac{1}{Z} \exp\left(\sum_{rs} h_{(d-1)r} h_{ds} W_{drs}\right)$$



Deep Belief Networks

Learning DBNs is like learning a sequence of RBMs. Called "Greedy layer-wise learning":

- Fit an RBM to learn W_1 (first layer weights) using contrastive divergence algorithm or similar.
- Fit a second RBM with input data sampled from $p(h_1|x; W_1)$ to learn W_2.
- Iterate this procedure for each layer until some stopping criterion is met.
- Additionally, sometimes a "fine-tuning" step will subsequently improve all the weights depending on the objective.

Remarkably effective in image analysis applications, especially as feature extraction/"pre-training" for supervised learning classifiers.

Combining deep NNs with graphical models

Deep Boltzmann Machines and Deep Belief Networks are deep learning models that are also graphical models. The architecture of these models encode conditional independence constraints that are exploited by learning procedures (e.g., involving Gibbs sampling) for fast learning.

Another way that deep learning and graphical models intersect is by using neural networks in place of the simpler parameteric models that together comprise a factorized joint distribution. Consider:

$$p(x, z; \theta) = p(z; \theta)p(x|z; \theta)$$

where Z is a latent vector, X is an observed vector, and $p(x|z; \theta)$ is parameterized by a neural network.

Deep latent variable models

$$
\begin{aligned}
p(z) &\sim N(0, \mathbf{I}) \\
\eta &= \text{NeuralNet}_\theta(z) \\
\log p(x|z) &= \sum_{i=1}^{p} \log p(x_i|z) = \sum_{i=1}^{p} \log \text{Bern}(\eta_i) \\
&= \sum_{i=1}^{p} x_i \log \eta_i + (1 - x_i) \log(1 - \eta_i)
\end{aligned}
$$

The Bernoulli parameters $\eta = (\eta_1, ..., \eta_p)$ are parameterized by a neural network (called "decoder" network).

$p(x; \theta) = \int p(x, z; \theta) dz$ and $p(z|x; \theta)$ are intractable to learn exactly. Turn to approximation methods.

Encoder or approximate posterior

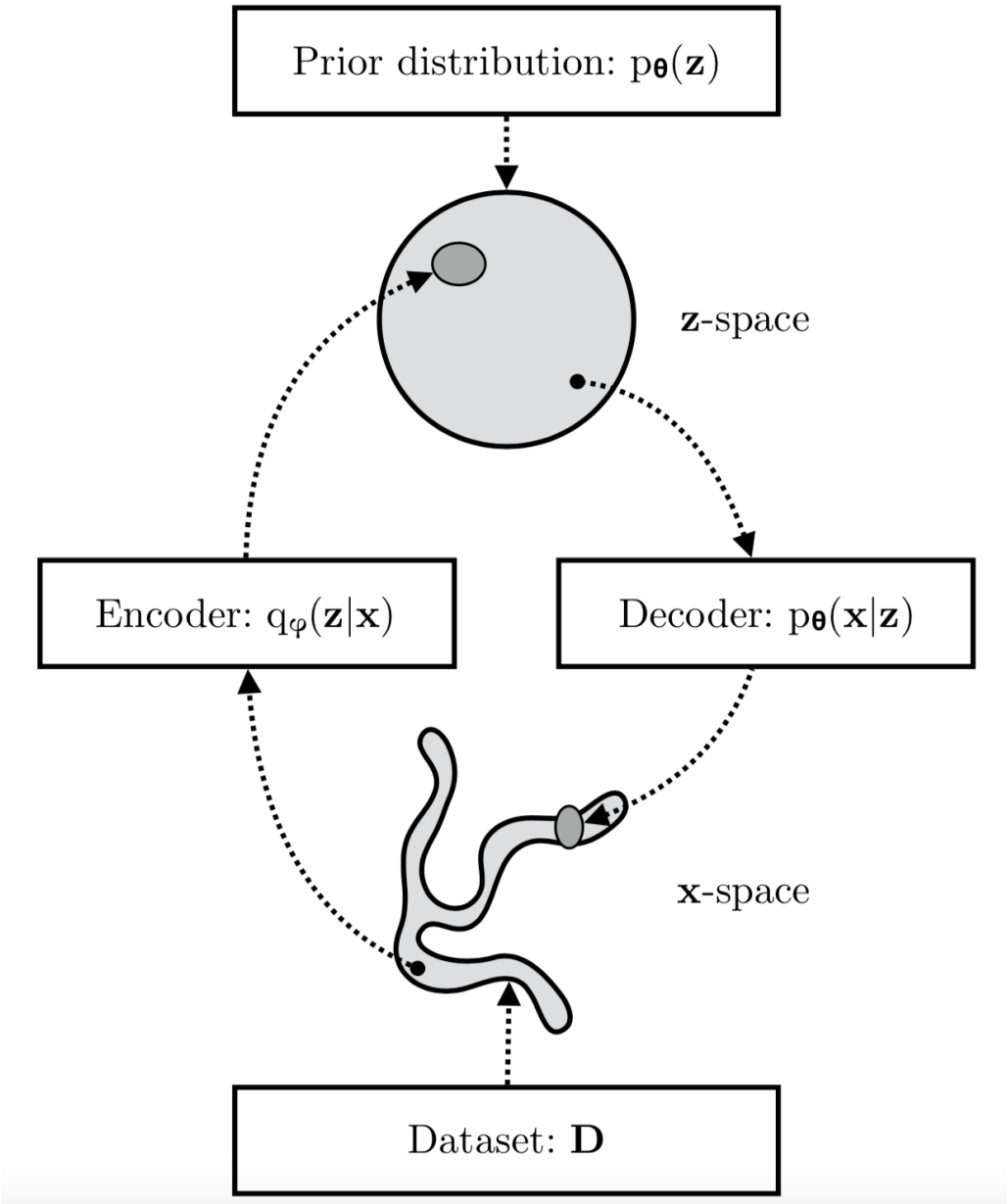
Optimize variational parameters ϕ such that $q(z|x;\phi) \approx p(z|x;\theta)$.

$q(z|x)$ can correspond to any chosen directed graphical model $q(z|x) = \prod_{m=1}^{M} q(z_m|\,\text{Pa}(Z_m, \mathcal{G}))$, but common choice is:

$$(\mu, \log \sigma) = \text{NeuralNet}_\phi(x)$$
$$q(z|x;\phi) \sim N(\mu, \text{diag}(\sigma^2))$$

This neural net is called the "encoder" network.

The variational approach to estimating this DLVM is known as the variational autoencoder (VAE).



from Kingma and Welling (2019)

Maximizing the ELBO

Recall in variational inference we choose parameters ϕ to maximize the ELBO:

$$\mathcal{L}_{\theta,\phi}(x) = \mathbb{E}_q[\log p(x, z; \theta) - \log q(z|x; \phi)]$$

A lower bound on the observed data (marginal) likelihood

$$\mathcal{L}_{\theta,\phi}(x) \leq \log p(x; \theta)$$

Derivation:

$$\begin{aligned}
\log p(x;\theta) &= \mathbb{E}_q[\log p(x;\theta)] \\
&= \mathbb{E}_q[\log \frac{p(x,z;\theta)}{p(z|x;\theta)}] \\
&= \mathbb{E}_q[\log \frac{p(x,z;\theta)}{q(z|x;\phi)} \frac{q(z|x;\phi)}{p(z|x;\theta)}] \\
&= \mathbb{E}_q[\log \frac{p(x,z;\theta)}{q(z|x;\phi)}] + \mathbb{E}_q[\log \frac{q(z|x;\phi)}{p(z|x;\theta)}] \\
&= \mathcal{L}_{\theta,\phi}(x) + D_{KL}(q||p)
\end{aligned}$$

Estimating the ELBO

We want gradients of the ELBO $\mathcal{L}_{\theta,\phi}(x)$ wrt θ, ϕ so we can use SGD to estimate params.

$\nabla_\theta \mathcal{L}_{\theta,\phi}(x)$ is not so problematic:

$$\begin{aligned}\nabla_\theta \mathcal{L}_{\theta,\phi}(x) &= \nabla_\theta \mathbb{E}_{q}[\log p(x,z;\theta) - \log q(z|x;\phi)] \\ &= \mathbb{E}_{q}[\nabla_\theta \log p(x,z;\theta)]\end{aligned}$$

which can be approximated with a Monte Carlo sample from $q(z|x;\phi)$. But

$$\begin{aligned}\nabla_\phi \mathcal{L}_{\theta,\phi}(x) &= \nabla_\phi \mathbb{E}_{q}[\log p(x,z;\theta) - \log q(z|x;\phi)] \\ &\neq \mathbb{E}_{q}[\nabla_\phi(\log p(x,z;\theta) - \log q(z|x;\phi))]\end{aligned}$$

Reparameterization trick

Re-express the r.v. $z \sim q(z|x;\phi)$ as a differentiable and invertible transformation of r.v. $\epsilon \sim p(\epsilon)$ such that

$$z = g(\epsilon, \phi, x)$$

Then $\mathbb{E}_q[\cdot] = \mathbb{E}_{p(\epsilon)}[\cdot]$.

Now $\mathcal{L}_{\theta,\phi}(x^j)$ (for single data point x^j) is estimated using

$$\begin{aligned}
\epsilon^s &\sim p(\epsilon) \\
z^{j,s} &= g(\epsilon^s, \phi, x^j) \\
\widetilde{\mathcal{L}}_{\theta,\phi}(x^j) &= \frac{1}{S}\sum_{s=1}^{S} \log p(x^j, z^{j,s};\theta) - \log q(z^{j,s}|x^j;\phi)
\end{aligned}$$

This is something which we can estimate and differentiate to get $\nabla_{\theta,\phi}\widetilde{\mathcal{L}}_{\theta,\phi}(x)$ and perform gradient descent.

Reparameterization trick

Note $\log q(z|x;\phi)$ can be computed knowing from choice of $p(\epsilon)$ and $g(\cdot)$.

$$\log q(z|x;\phi) = \log p(\epsilon) - \log |\det \frac{\partial z}{\partial \epsilon}|$$

this uses the determinant of the Jacobian matrix for the transformation $z = g(\epsilon, \phi, x)$.

For specific cases this actually turns out much simpler than it may look. For $q(z|x) = \prod_m N(z_m; \mu_m, \sigma_m^2)$ choose $\epsilon \sim N(0, \mathbf{I})$ and $z = \mu + \sigma \odot \epsilon$. Then $\log |\det \frac{\partial z}{\partial \epsilon}| = \sum_m \log \sigma_m$ so

$$\log q(z|x;\phi) = \sum_m \log N(\epsilon_m; 0, 1) - \log \sigma_m$$

ELBO

We wrote the ELBO as:

$$\mathcal{L}_{\theta,\phi}(x) = \mathbb{E}_q[\log p(x, z; \theta) - \log q(z|x; \phi)]$$

Sometimes you will instead see it rewritten equivalently as:

$$\mathcal{L}_{\theta,\phi}(x) = -D_{KL}(q(z|x; \phi)||p(z; \theta)) + \mathbb{E}_q[\log p(x|z; \theta)]$$

which follows from some algebra. The first term can be written down exactly for some common choices of prior $p(z; \theta)$ and proposal $q(z|x; \phi)$.

Gaussian example

For the factorized Gaussian proposal distribution, ELBO estimator (for one observation x^j) can be written:

$$\widetilde{\mathcal{L}}_{\theta,\phi}(x^j) = \frac{1}{2}\sum_{m=1}^{M}(1 + \log(\sigma_m^j)^2 - \mu_m^j - (\sigma_m^j)^2) + \frac{1}{S}\sum_{s=1}^{S}\log p(x^j|z^{j,s};\theta)$$

with $\epsilon^s \sim N(0, \mathbf{I})$ and $z^{j,s} = \mu^j + \sigma^j \odot \epsilon^s$, and $\log p(x^j|z^{j,s};\theta)$ parameterized by a NN. Recall X depends on Z via $\eta = \text{NeuralNet}_\theta(z)$ and Z depends on X via $(\mu, \log\sigma) = \text{NeuralNet}_\phi(x)$ in our model.

Can take derivatives of this and update params with SGD.

Sampling from latent space

(a) 2-D latent space (b) 5-D latent space (c) 10-D latent space (d) 20-D latent space

Figure 5: Random samples from learned generative models of MNIST for different dimensionalities of latent space.

from Kingma and Welling (2014)

Sampling from latent space *in some preferred direction*

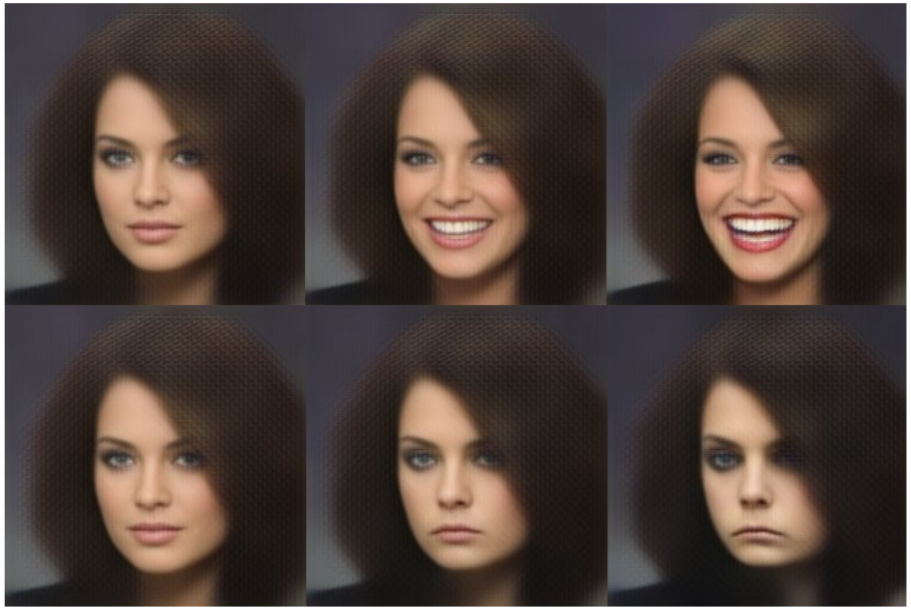


Figure 4.4: VAEs can be used for image resynthesis. In this example by White, 2016, an original image (left) is modified in a latent space in the direction of a *smile vector*, producing a range of versions of the original, from smiling to sadness.

from Kingma and Welling (2019)

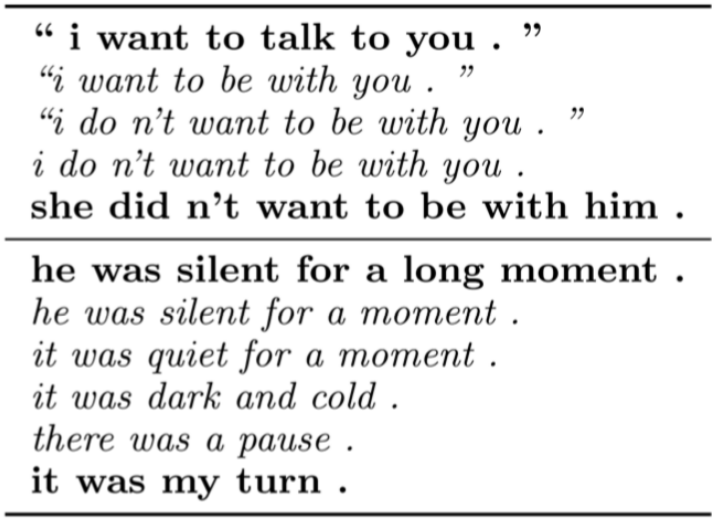


Figure 4.3: An application of VAEs to interpolation between pairs of sentences, from (Bowman *et al.*, 2015). The intermediate sentences are grammatically correct, and the topic and syntactic structure are typically locally consistent.

from Kingma and Welling (2019)

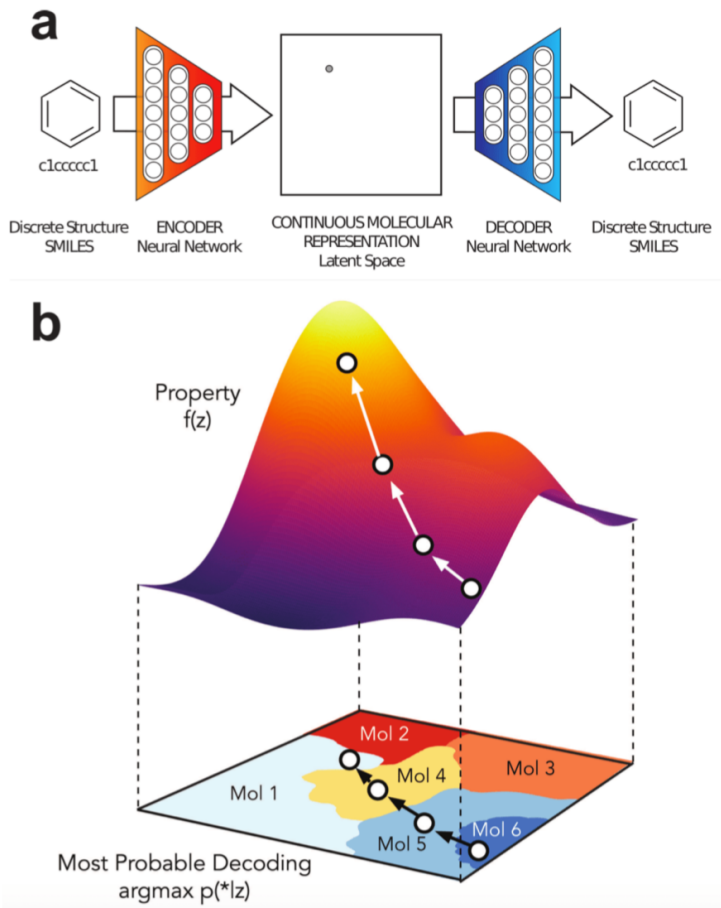


Figure 4.2: (a) Application of a VAE to chemical design in (Gómez-Bombarelli *et al.*, 2018). A latent continuous representation $\mathbf{z}$ of molecules is learned on a large dataset of molecules. (b) This continuous representation enables gradient-based search of new molecules that maximizes $f(\mathbf{z})$, a certain desired property.

from Kingma and Welling (2019)