Variational Auto-Encoders without (too) much math

Stéphane d’Ascoli
Roadmap

1. A reminder on auto-encoders
   a. Basics
   b. Denoising and sparse encoders
   c. Why do we need VAEs?

2. Understanding variational auto-encoders
   a. Key ingredients
   b. The reparameterization trick
   c. The underlying math

3. Applications and perspectives
   a. Disentanglement
   b. Adding a discrete condition
   c. Applications
   d. Comparison with GANs

4. Do it yourself in PyTorch
   a. Build a basic denoising encoder
   b. Build a conditional VAE
Auto-Encoders
**Basics**

\[ \mathcal{L}(x) = \frac{1}{2} \left( x - \theta(\phi(x)) \right)^2 \]
Denoising and Sparse Auto-Encoders

Denoising:

Sparse: enforces specialization of hidden units

\[ \mathcal{L}(x, \hat{x}) + \lambda \sum_i |a_i^{(h)}| \]

Contractive: enforces that close inputs give close outputs

\[ \mathcal{L}(x, \hat{x}) + \lambda \sum_i \left\| \nabla_x a_i^{(h)}(x) \right\|^2 \]
Why do we need VAE?

VAE’s are used as generative models: sample a latent vector, decode and you have a new sample.

Q: Why can’t we use normal auto-encoders?
A: If we choose an arbitrary latent vector, we get garbage.

Q: Why?
A: Because latent space has no structure!
Variational Auto-Encoders
**Key Ingredients**

**Generative models**: unsupervised learning, aim to learn the distribution underlying the input data

**VAEs**: Map the complicated data distribution to a simpler distribution (encoder) we can sample from (Kingma & Welling 2014) to generate images (decoder)
Q: Why encode into distributions rather than discrete values?
A: To impose that close values of $z$ give close values of $x$: latent space becomes more meaningful.

Now if we sample $z$ anywhere inside the distribution obtained with $x$, we reconstruct $x$. But we want to generate new images!

Problem: if we sample $z$ elsewhere, we get garbage...
SECOND INGREDIENT: IMPOSE STRUCTURE

Q: How can we make the images generated look realistic whatever the sampled $z$?
A: Make sure that $Q(z|x)$ for different $x$’s are close together!

$$z \sim \mathcal{N}(\mu_\phi(x), \sigma_\phi(x))$$
Second Ingredient: impose structure

Q: How do we keep the distributions close together?
A: By enforcing the overall distribution in latent space to follow a standard Gaussian prior.

Q: How?
A: KL divergence! \[ \mathcal{L}_{KL} = \mathbb{E}_{x \sim \text{dataset}} [D_{KL} \{ Q_{\phi}(z|x) \| p(z) \} ], p \sim \mathcal{N}(0, 1) \]
The Reparametrization Trick

Q: How can we backpropagate when one of the nodes is non-deterministic?
A: Use the reparametrization trick!
The Underlying Information Theory

Consider a latent variable model with a data variable $x \in \mathcal{X}$ and a latent variable $z \in \mathcal{Z}$, $p(z, x) = p(z)p_\theta(x|z)$. Given the data $x_1, \ldots, x_n$, we want to train the model by maximizing the marginal log-likelihood:

$$\mathcal{L} = E_{p_d(x)} \left[ \log p_\theta(x) \right] = E_{p_d(x)} \left[ \log \int_z p_\theta(x|z)p(z)dz \right],$$

where $p_d$ denotes the empirical distribution of $X$: $p_d(x) = \frac{1}{n} \sum_{i=1}^n \delta_{x_i}(x)$.

To avoid the (often) difficult computation of the integral above, the idea behind variational methods is to instead maximize a lower bound to the log-likelihood:

$$\mathcal{L} \geq L(p_\theta(x|z), q(z|x)) = E_{p_d(x)} \left[ E_{q(z|x)} \left[ \log p_\theta(x|z) \right] - \text{KL}(q(z|x)||p(z)) \right].$$

Any choice of $q(z|x)$ gives a valid lower bound. Variational autoencoders replace the variational posterior $q(z|x)$ by an inference network $q_\phi(z|x)$ that is trained together with $p_\theta(x|z)$ to jointly maximize $L(p_\theta, q_\phi)$. The variational posterior $q_\phi(z|x)$ is also called the encoder and the generative model $p_\theta(x|z)$, the decoder or generator.

The first term $E_{q(z|x)} \left[ \log p_\theta(x|z) \right]$ is the negative reconstruction error. Indeed under a gaussian assumption i.e. $p_\theta(x|z) = \mathcal{N}(\mu_\theta(z), 1)$ the term $\log p_\theta(x|z)$ reduced to $\alpha \|x - \mu_\theta(z)\|^2$, which is often used in practice. The term $\text{KL}(q(z|x)||p(z))$ can be seen as a regularization term, where the variational posterior $q_\phi(z|x)$ should be matched to the prior $p(z) = \mathcal{N}(0, 1)$. 
Proof of the Lower Bound

Q: Why “variational” auto-encoders?
A: Relies on a variational method

Consider a tractable distribution \( Q \) instead

\[
P(z|x) = \frac{P(x|z)P(z)}{P(x)} = \frac{P(x|z)P(z)}{\sum_z P(x|z)P(z)}
\]

Intractable!

\[
D_{KL}(Q(z|x)||P(z|x)) = \sum_z Q(z|x) \log \frac{Q(z|x)}{P(z|x)} > 0
\]

\[
= \log P(x) + \sum_z Q(z|x) \log \frac{Q(z|x)}{P(z)} - \sum_z Q(z|x) \log P(x|z)
\]

\[
= \log P(x) + D_{KL}(Q(z|x)||P(z)) - \mathbb{E}_{z \sim Q} \log P(x|z)
\]

Regularizer \(-\mathcal{L}\)  Reconstruction loss

\[
\mathbb{E}_{x \sim D} \log P(x) \geq \mathbb{E}_{x \sim D} [D_{KL}(Q(z|x)||P(z|x)) + \mathcal{L}]
\]

ELBO
VAEs in Practice
Disentanglement: Beta-Vae

We saw that the objective function is made of a reconstruction and a regularization part.

\[ \mathcal{L} = \mathbb{E}_{z \sim Q} \log P(x|z) - \beta D_{KL}(Q(z|x) \| P(z)) \]

By adding a tuning parameter we can control the tradeoff.

If we increase beta:
- The dimensions of the latent representation are more disentangled
- But the reconstruction loss is less good
Generating Conditionally: CVAEs

Add a one-hot encoded vector to the latent space and use it as categorical variable, hoping that it will encode discrete features in data (number in MNIST)

Q: The usual reparameterization trick doesn’t work here, because we need to sample discrete values from the distribution! What can we do?
A: Gumbel-Max trick

Q: How do I balance the regularization terms for the continuous and discrete parts?
A: Control the KL divergences independently

\[ \mathcal{L}(\theta, \phi) = \mathbb{E}_{q_{\phi}(z, c|x)}[\log p_{\theta}(x|z, c)] \]

\[-\gamma |D_{KL}(q_{\phi}(z|x) \| p(z)) - C_z| - \gamma |D_{KL}(q_{\phi}(c|x) \| p(c)) - C_c|\]
Applications

Image generation: Dupont et al. 2018


“i want to talk to you.”
“i want to be with you.”
“i do n’t want to be with you.”
i do n’t want to be with you.
she did n’t want to be with him.

he was silent for a long moment.
he was silent for a moment.
it was quiet for a moment.
it was dark and cold.
there was a pause.
it was my turn.
## Comparison with GANS

<table>
<thead>
<tr>
<th>VAE</th>
<th>GAN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Easy metric : reconstruction loss</td>
<td>Cleaner images</td>
</tr>
<tr>
<td>Interpretable and disentangled latent space</td>
<td>Low interpretability</td>
</tr>
<tr>
<td>Easy to train</td>
<td>Tedious hyperparameter searching</td>
</tr>
<tr>
<td>Noisy generation</td>
<td>Clean generation</td>
</tr>
</tbody>
</table>
Towards a Mix of the Two?

Prominent attributes: White, Male, Curly Hair, Frowning, Eyes Open, Pointy Nose, Flash, Posed Photo, Eyeglasses, Narrow Eyes, Teeth Not Visible, Senior, Receding Hairline.
Do It Yourself
In Pytorch
**Auto-Encoder**

1. Example: a **simple fully-connected** auto-encoder

```python
loss_fn = torch.nn.MSELoss()

def train_model(model, loss_fn, data_loader=None, epochs=1, optimizer=None):
    model.train()
    for epoch in range(epochs):
        for batch_idx, (data, _) in enumerate(train_loader):
            data = data.view([-1, 784])
            output = model(data)
            loss = loss_fn(output, data)
            loss.backward()
            optimizer.step()
            if batch_idx % 50 == 0:
                print(f'Train Epoch: {epoch} [{batch_idx * len(data_loader.dataset) / len(data_loader.dataset):.0f}%]
                      Loss: {loss.item():.6f}'.format(epoch, batch_idx * len(data_loader.dataset), len(data_loader.dataset), loss.data.item()))
```

```python
class AutoEncoder(nn.Module):
    def __init__(self, input_dim, encoding_dim):
        super(AutoEncoder, self).__init__()
        self.encoder = nn.Linear(input_dim, encoding_dim)
        self.decoder = nn.Linear(encoding_dim, input_dim)

    def forward(self, x):
        encoded = F.relu(self.encoder(x))
        decoded = self.decoder(encoded)
        return decoded
```

2. DIY: implement a **denoising convolutional** auto-encoder for MNIST
1. Example: a **simple** VAE

```python
def train(model, data_loader=data_loader, num_epochs=num_epochs):
    for epoch in range(num_epochs):
        for i, (x, _) in enumerate(data_loader):

            # Forward pass
            x = x.to(device).view(-1, image_size)
            x_reconst, mu, log_var = model(x)

            # Compute reconstruction loss and kl divergence
            reconstr_loss = F.binary_cross_entropy(x_reconst, x, reduction='sum')
            kl_div = - 0.5 * torch.sum(1 + log_var - mu.pow(2) - log_var.exp())

            # Backprop and optimize
            loss = reconstr_loss + kl_div
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()
```

2. DIY: implement a **conditional** VAE for MNIST

```python
class VAE(nn.Module):
    def __init__(self, image_size=784, h_dim=400, z_dim=20):
        super(VAE, self).__init__()
        self.fc1 = nn.Linear(image_size, h_dim)
        self.fc2 = nn.Linear(h_dim, z_dim)
        self.fc3 = nn.Linear(h_dim, z_dim)
        self.fc4 = nn.Linear(z_dim, h_dim)
        self.fc5 = nn.Linear(h_dim, image_size)

    def encode(self, x):
        h = F.relu(self.fc1(x))
        return self.fc2(h), self.fc3(h)

    def reparameterize(self, mu, log_var):
        std = torch.exp(log_var/2)
        eps = torch.randn_like(std)
        return mu + eps * std

    def decode(self, z):
        h = F.relu(self.fc4(z))
        return torch.sigmoid(self.fc5(h))

    def forward(self, x):
        mu, log_var = self.encode(x)
        z = self.reparameterize(mu, log_var)
        x_reconst = self.decode(z)
        return x_reconst, mu, log_var
```
Questions