Midterm Project: β-VAE

Armaan Kohli - ECE471 Computational Graphs for Machine Learning
Autumn 2019

Remarks

We attempted to replicate results from *β-VAE: Learning Basic Visual Concepts with a Constrained Variational Framework*[1]. Published in 2017 at ICLR, the team at Google DeepMind demonstrated that variational autoencoders, first described in [2], had the ability to produce 'disentangled' representations by augmenting the loss function described in [2] with an additional hyper-parameter, β. In a follow-up publication, [3], DeepMind explains this effect further. Effectively, the parameter β finds latent components which make different contributions to the log-likelihood term of the objective function in [2], and that latent components correspond to features that are qualitatively different.

In their publication, DeepMind illustrates the effectiveness of their network on the 3DChairs and CelebA datasets. We have attempted to replicate Fig. 1 and Fig. 2 of [1] for the β-VAE in order to demonstrate that our implementation based on their paper faithfully replicates their results.

There were several parameters needed for implementation that the paper neglected to mention. The number of gradient steps nor were their compute resources stated in the paper. So, we opted to use a gpu and train for as long as possible, saving checkpoints and outputs along the way. They also didn't mention the dimensionality of the latent space, nor how they sampled or traversed the latent space to generate Fig. 1 and Fig. 2 in [1]. As such, we tried two different methods to replicate their results: We took random samples from a 7 dimensional latent space and we took the first n samples from the same latent space. We found that these unknown parameters had a significant impact on our results.

To see the full codebase, please visit github.com/armaank/bVAE. See the *Code* section for selected code snippets. We elected to implement our version of β-VAE in pytorch, for the learning experience and to cut time spent making a dataloader (since we were working with multiple datasets) in tensorflow, which by constrast, is easily done in pytorch. The CelebA dataset was trained using Ali's computer (GTX2070), and the 3DChairs dataset was trained on a P100 on a Google Cloud VM instance.

Results & Discussion

Code

Here is a code snippet showing the model architecture we replicated from the appendix of [1].

```
"""
model.py

contains network architecture for beta vae, as described in the appendix of [2]

"""
import torch
import torch.nn as nn
from torch.autograd import Variable
import torch.nn.init as init


def reparam(mu, logvar):
    """reparametization 'trick'

    allows optimization through sampling process.
    inputs: mean and variance
    outputs: random var with perscribed mean and noisy variance terms

    """

    std = logvar.div(2).exp()
    eps = Variable(std.data.new(std.size()).normal_())

    return mu + std * eps


class View(nn.Module):
    """View

    acts like tf/np reshape

    """

    def __init__(self, size):
        super(View, self).__init__()
        self.size = size

    def forward(self, tensor):
        return tensor.view(self.size)

```

```
42
43 class betaVAE(nn.Module):
44     """betaVAE
45
46     class used to setup the betaVAE architecture
47
48     """
49
50     def __init__(self, z_dim=10, nchan=1):
51         super(betaVAE, self).__init__()
52         self.z_dim = z_dim
53
54         self.encoder = nn.Sequential(
55             nn.Conv2d(nchan, 32, 4, 2, 1),
56             nn.ReLU(True),
57             nn.Conv2d(32, 32, 4, 2, 1),
58             nn.ReLU(True),
59             nn.Conv2d(32, 32, 4, 2, 1),
60             nn.ReLU(True),
61             nn.Conv2d(32, 32, 4, 2, 1),
62             nn.ReLU(True),
63             View((-1, 32 * 4 * 4)),
64             nn.Linear(32 * 4 * 4, 256),
65             nn.ReLU(True),
66             nn.Linear(256, 256),
67             nn.ReLU(True),
68             nn.Linear(256, z_dim * 2),
69         )
70
71         self.decoder = nn.Sequential(
72             nn.Linear(z_dim, 256),
73             View((-1, 256, 1, 1)),
74             nn.ReLU(True),
75             nn.ConvTranspose2d(256, 64, 4),
76             nn.ReLU(True),
77             nn.ConvTranspose2d(64, 64, 4, 2, 1),
78             nn.ReLU(True),
79             nn.ConvTranspose2d(64, 32, 4, 2, 1),
80             nn.ReLU(True),
81             nn.ConvTranspose2d(32, 32, 4, 2, 1),
82             nn.ReLU(True),
83             nn.ConvTranspose2d(32, nchan, 4, 2, 1),
84         )
85
```

```
86      def forward(self, x):
87          """forward
88
89          propgates input through the network
90          inputs: sample input
91          output: reconstructed input, mu and var from the latent space
92
93          """
94          dist = self.encode(x)
95          mu = dist[:, :self.z_dim]
96          logvar = dist[:, self.z_dim:]
97          z = reparam(mu, logvar)
98          x_recon = self.decode(z)
99
100         return x_recon, mu, logvar
101
102     def encode(self, x):
103         return self.encoder(x)
104
105     def decode(self, z):
106         return self.decoder(z)
107
108
109 if __name__ == "__main__":
110     pass
```

References

[1] I. Higgins, L. Matthey, A. Pal, C. Burgess, X. Glorot, M. M. Botvinick, S. Mohamed, and A. Lerchner, "beta-vae: Learning basic visual concepts with a constrained variational framework," in *ICLR*, 2017.

[2] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *CoRR*, vol. abs/1312.6114, 2013.

[3] C. P. Burgess, I. Higgins, A. Pal, L. Matthey, N. Watters, G. Desjardins, and A. Lerchner, "Understanding disentangling in beta-vae," *ArXiv*, vol. abs/1804.03599, 2018.

Credits

Ali, for his friendship and gpu :)