Model Information and Architectures

Patrick Sullivan

August 25, 2017

1 Variational Autoencoder

The model implemented here is essentially as introduced in (Kingma and Welling 2013). It uses the "reparameterization trick" to construct a Gaussian noise sampling that is differentiable, enabling us to backprop the loss over the entire network. The network is trained end-to-end with a single loss function

$$\mathcal{L}(\mathbf{x}) = \underbrace{\frac{1}{n}\sum_{i}^{n} \mathbf{x}\log\hat{\mathbf{x}} + (1-\mathbf{x})\log(1-\hat{\mathbf{x}})}_{\text{reconstruction error}} + \underbrace{\frac{1}{2n}\sum_{i}^{n} \mu^2 + \sigma^2 - \log\sigma^2 - 1}_{\text{latent loss}} \tag{1}$$

The loss function encourages the network to encode the input into a value that is consistent with being sampled from a Guassian distribution, while encouraging the decoder to translate this encoding to an image that matches the original. By encouraging the encoding to follow a Guassian distribution, we can then sample from it, feed this value to the decoder, and create new generative outputs.

See Fig. 1 and Fig. 2 for the encoder and decoder architecture, respectively.

2 Standard, Wasserstein, and Improved Wasserstein GANs

There are a few different GANs implemented that share the same architecture but differ in how they are trained. They are:

- Standard GAN (Goodfellow et al. 2014)
- Wasserstein GAN (Arjovsky, Chintala, and Bottou 2017)
- Improved Wasserstein GAN (Gulrajani et al. 2017)

See Fig. 4 and Fig. 3 for the generator and discriminator architectures, respectively.

2.1 Standard GAN

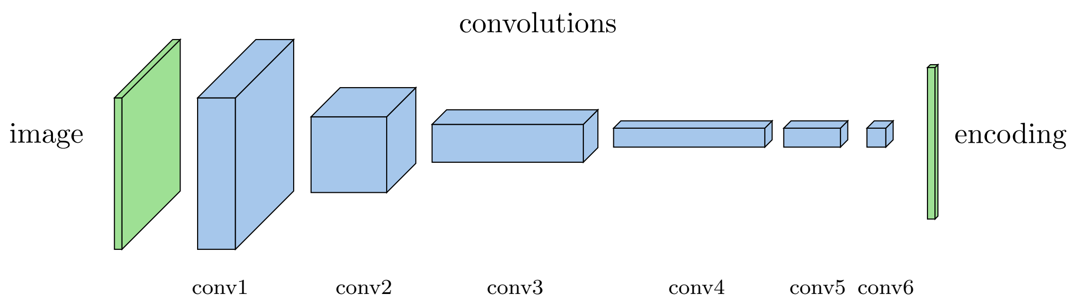
In a standard GAN, the generator is trained against the loss function

$$\mathcal{L}(\mathbf{x}) = \frac{1}{n}\sum_{i}^{n} -\log(\mathrm{disc}(\mathrm{gen}(\mathbf{x}))) \tag{2}$$

while the discriminator is trained against

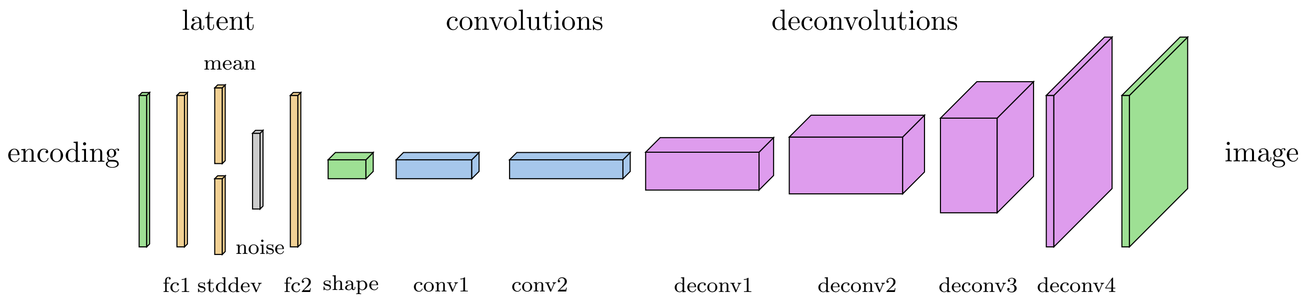
$$\mathcal{L}(\mathbf{x}) = \frac{1}{n}\sum_{i}^{n} -\log(\mathrm{disc}(\mathbf{x})) - \log(1-\mathrm{disc}(\mathrm{gen}(\mathbf{x}))) \tag{3}$$

The discriminator is trained $t > 0$ times in sequence followed by training the generator once.



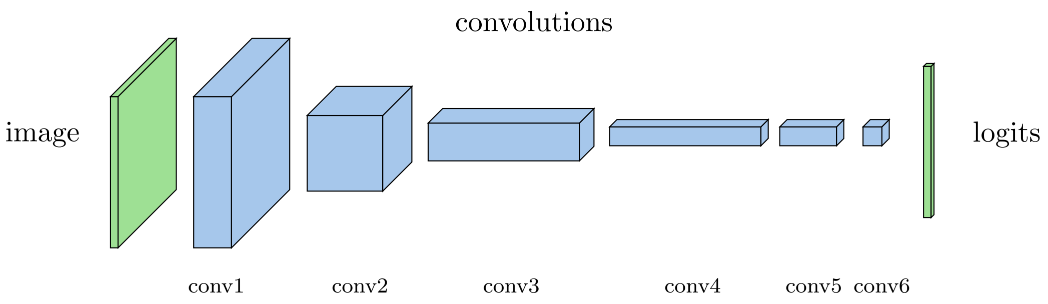
layer	filter size	stride	# filters
conv1	5	2	64
conv2	5	2	128
conv3	5	2	256
conv4	5	2	256
conv5	32	1	96
conv6	32	1	32

Figure 1: **VAE encoder** architecture. All layers use Leaky ReLU activations. The final convolution output is flattened and represents our encoded image.



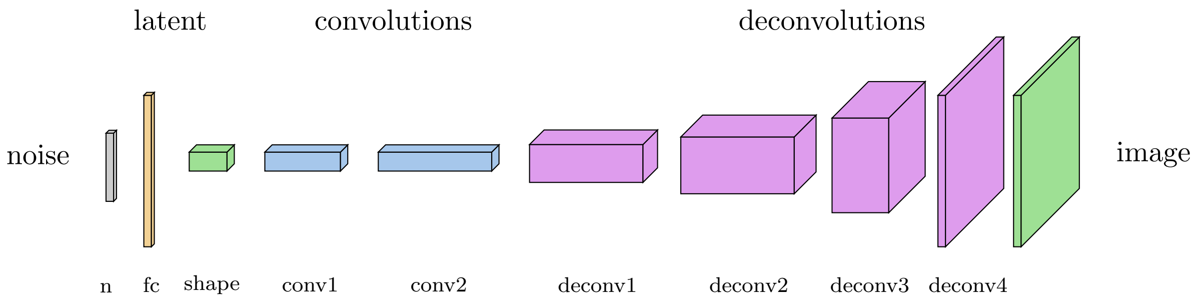
layer	filter size	stride	# filters (or size)
fc1			512
stddev			200
mean			200
noise			200
fc2			512
shape			4x4x32
conv1	1	1	96
conv2	1	1	256
deconv1	5	2	256
deconv2	5	2	128
deconv3	5	2	64
deconv4	5	2	3

Figure 2: **VAE decoder** architecture. The latent is generated by remapping the encoding into two vectors representing the *mean* and the *standard deviation* of a Gaussian, which is then (via the reparameterization trick) used to generate a Gaussian. This is then mapped back out through a dense layer before reshaping into a small, but deep, image. This is then run through two convolution filters to expand the number of filters before upscaling with deconvolution operations.



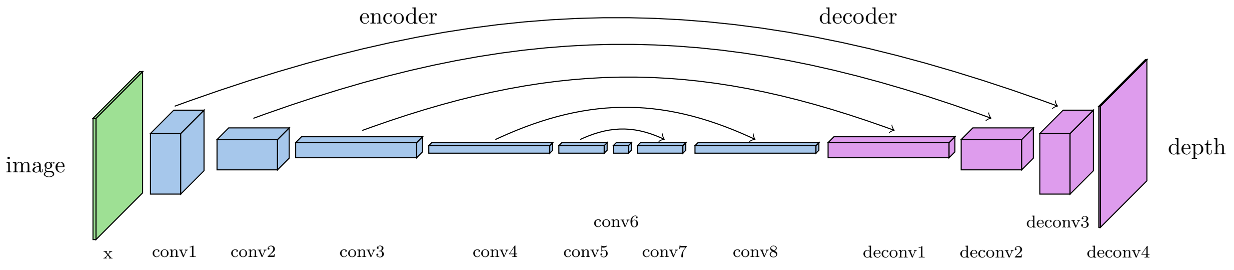
layer	filter size	stride	# filters
conv1	5	2	64
conv2	5	2	128
conv3	5	2	256
conv4	5	2	256
conv5	32	1	96
conv6	32	1	32

Figure 3: **GAN/WGAN discriminator** architecture. In the original GAN/WGAN architecture, all convolution layers use batch normalization and a leaky ReLU activation function. In the improved WGAN architecture, batch norm is omitted. All three models use *sigmoid* activation in the logits layer.



layer	filter size	stride	# filters (or size)
n			200
fc			512
shape			4x4x32
conv1	1	1	96
conv2	1	1	256
deconv1	5	2	256
deconv2	5	2	128
deconv3	5	2	64
deconv4	5	2	3

Figure 4: **GAN/WGAN generator** architecture. Starting with a latent vector of Gaussian noise, we expand it using a fully connected layer and then reshape it into an image with depth. The number of filters is then expanded further using convolutions before upscaling it several times using deconvolution operations. Each (de)convolution layer uses batch normalization and a ReLU activation function, and the final output values are rescaled with *tanh*.



layer	filter size	stride	# filters (or size)
x			64x646x3
conv1	5	2	64
conv2	5	2	128
conv3	5	2	256
conv4	5	2	256
conv5	1	1	96
conv6	1	1	32
conv7	1	1	96 + skip
conv8	1	1	256 + skip
deconv1	5	2	256 + skip
deconv2	5	2	128 + skip
deconv3	5	2	64 + skip
deconv4	5	2	1

Figure 5: **cGAN generator** architecture. A 64x64x3 RGB image is passed into the network and is then encoded into a 512-sized latent vector (conv6). This is then decoded back out to a 64x64x1 image representing an estimation of the depth. Skip layers are added from each layer in the encoder to its match in the decoder via concatenation. Activations in the generator are LeakyReLU while the decoder uses ReLU, with the final layer using tanh. If enabled, batchnorm and dropout occurs at all layers except the first and last layers. The loss for this network is

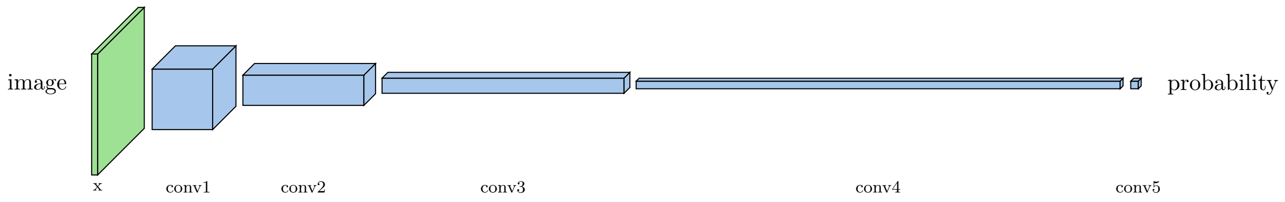
$$\mathcal{L}(\mathbf{x}) = \frac{1}{n}\sum_{i}^{n} \text{disc}(\text{gen}(\mathbf{x})) + \frac{1}{n}\sum_{i}^{n} \sqrt{(\mathbf{x} - \hat{\mathbf{x}})^2} \tag{5}$$

aka the Wasserstein loss from the discriminator plus the RMSE loss of the reconstructed depth.

3 Conditional WGAN

This model is actually neither conditional nor a GAN, however, it bears several similaries. It is a combination of the “pix2pix” model described in (Isola et al. 2016) and the Improved Wasserstein training techniques found in (Gulrajani et al. 2017).

See Fig. 5 and Fig. 6 for the generator and discriminator architecture, respectively.



layer	filter size	stride	# filters (or size)
x			64x64x4
conv1	5	2	64
conv2	5	2	128
conv3	5	2	256
conv4	1	1	512
conv5	1	1	1

Figure 6: **cGAN discriminator** architecture. This is a standard DCGAN architecture that takes a 64x64x4 image (RGB concatenated with depth) and returns a single probability as to whether the input is fake or real. Activations are LeakyReLU, with the exception of the final layer, which uses sigmoid. The loss for this network is

$$\mathcal{L}(\mathbf{x}) = \frac{1}{n}\sum_{i}^{n} \text{disc}(\text{gen}(\mathbf{x})) - \frac{1}{n}\sum_{i}^{n} \text{disc}(\mathbf{x}) + \lambda P \tag{7}$$

where P is the gradient penalty from Gulrajani et al. 2017.

4 Bibliography

References

Arjovsky, M., S. Chintala, and L. Bottou (2017). "Wasserstein GAN". In: *ArXiv e-prints.* arXiv: `1701.07875 [stat.ML]`.

Goodfellow, Ian J. et al. (2014). "Generative Adversarial Networks". In: *CoRR* abs/1406.2661. URL: `http://arxiv.org/abs/1406.2661`.

Gulrajani, Ishaan et al. (2017). "Improved Training of Wasserstein GANs". In: *CoRR* abs/1704.00028. URL: `http://arxiv.org/abs/1704.00028`.

Isola, Phillip et al. (2016). "Image-to-Image Translation with Conditional Adversarial Networks". In: *CoRR* abs/1611.07004. URL: `http://arxiv.org/abs/1611.07004`.

Kingma, D. P and M. Welling (2013). "Auto-Encoding Variational Bayes". In: *ArXiv e-prints.* arXiv: `1312.6114 [stat.ML]`.