SOPHOS
Cybersecurity made simple.

Adversarial Autoencoders

By Dr. Richard Harang and Madeline Schiappa

So what exactly is an adversarial autoencoder, and why might you want to use one?

Generative modeling usually focuses on three main questions:

1. Inference: How can I take some observation or data point and convert it into a latent or compressed representation?

2. Generation: How can I take a (valid) latent/compressed representation and convert it back into its data point?

3. Sampling: How can I generate new observations that are from the same distribution as my original data?

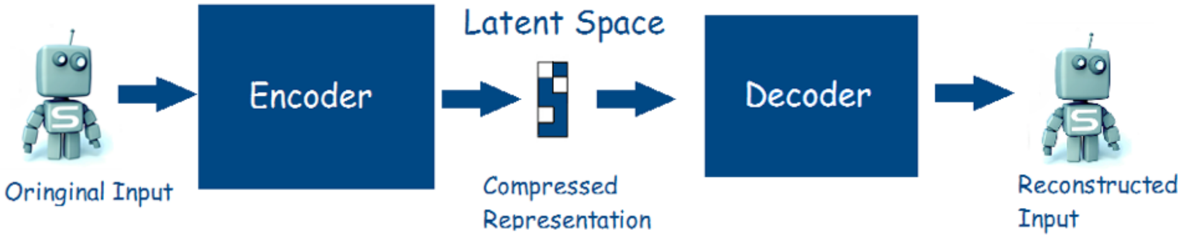

Figure 1. Variational Autoencoder (VAE).

Variational Autoencoders (VAEs), Generative Adversarial Networks (GANs), and Adversarial Autoencoders (AAEs) all take different approaches to this problem. VAEs are the most explicitly 'statistical' of the three. They assume some prior on the latent space p(z) and a decoder p(x|z), at which point the sampling becomes straightforward: draw a sample from p(z), then draw a sample from p(x|z) conditional on that z you just drew. The hard part, of course, is that we don't know p(x|z), and without knowing the z that goes along with some x, it's impossible to learn it. So VAEs try to learn an approximator q(z|x) that learns how to generate a z for a given x, and uses penalization based on the KL divergence, or distance between two distributions, to make sure that that distribution is "close" to p(z).

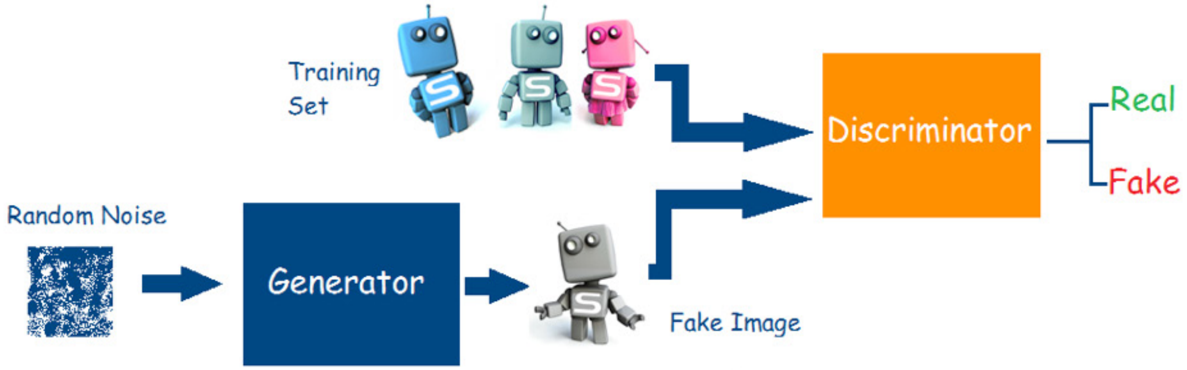

Figure 2. Generative Adversarial Network (GAN).

Adversarial Autoencoders

GANs actually skip the inference bit and focus entirely on the generative step. They just start with a latent space that they can sample from – such as a multivariate Gaussian – as p(z) and never change it. These sampled z values go into a decoder p(x|z) which produces something in the data space, but then – and this is the clever bit – a second network called a discriminator will try to distinguish a sample from p(*x*|*z*) from a sample of p(*x*) – the available training data. The decoder then has to learn to 'trick' the discriminator, and in so doing, learns to produce samples from the actual data space that can be generated by sampling from the fixed latent space.

AAEs are a clever blend of traditional autoencoders and the idea of an adversarial loss that GANs introduced that lead to a framework of surprising flexibility. The key insight is that an autoencoder does the inference and generation steps by design, but sampling is made difficult by the fact that the code space that they produce isn't well-structured. AAEs take the idea of the adversarial loss from GANs to do "adversarial regularization": force that code space to look like a known prior distribution that you can sample from.

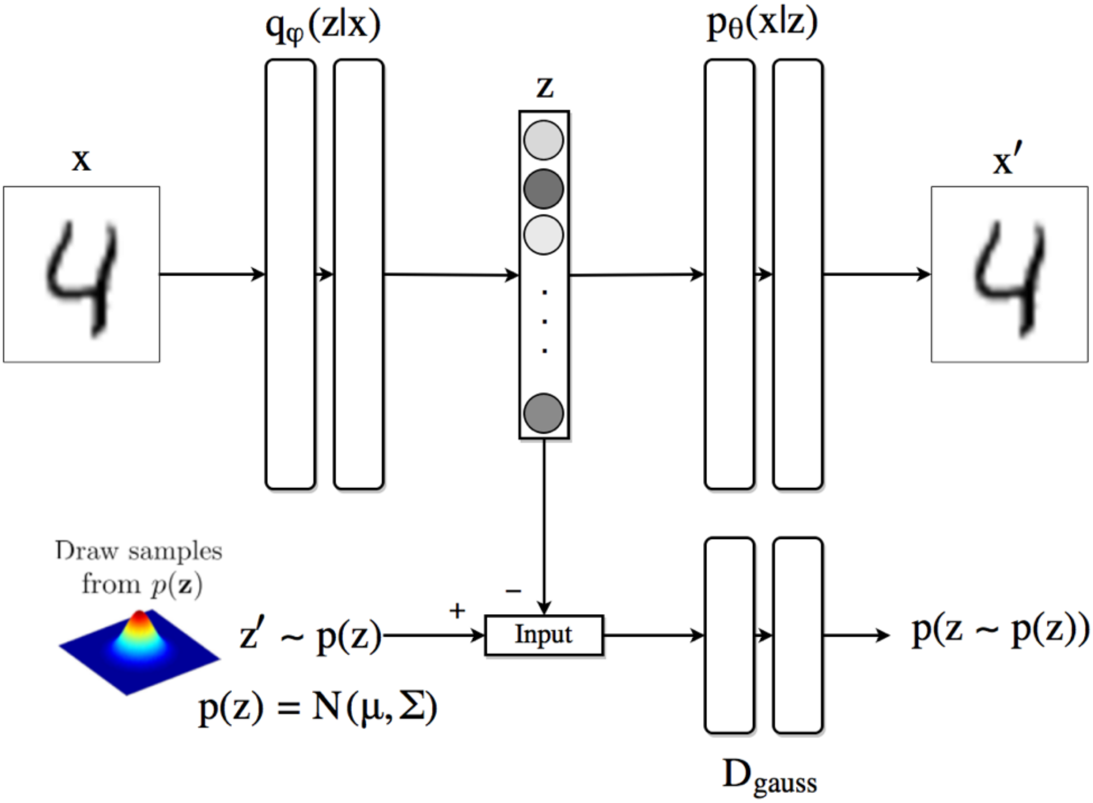

Figure 3. Adversarial Autoencoder (AAE) (from Makhzani et al. 2016).

Once you do that, the encoder is forced to do two things simultaneously:

1. It has to produce a code space that the decoder can work with to reconstruct the original input.
2. It has to produce a code space that can 'trick' the discriminator into thinking that it's just another sample from the prior distribution.

If it succeeds in both tasks, then the sampling step is simple. You can generate new samples just like you would in a GAN: sample from your prior, feed it to your decoder, and let it produce your sample.

A similar idea to the adversarial regularization pops up in VAEs in the KL divergence penalty, but it's much weaker. The KL-based loss is willing to "trade off" between reconstruction error and closeness to the prior, and this leads to a somewhat weaker match between the prior and the latent space code, as you can see in Figure 4c and 4d of the paper.

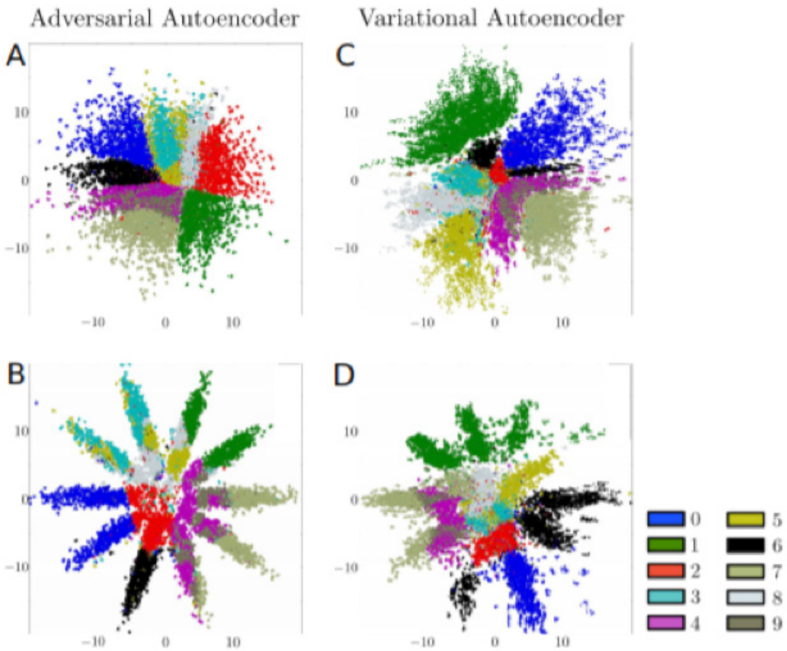

Figure 4. The images represent the hidden code z of the hold-out images for an Autoencoder to fit to (a and c) a 2D Gaussian (b and d) a mixture of 10 2D Gaussians. Each color represents the associated label (from Makhzani et al. 2016).

But wait, there's more!

The neat thing about this paper is how they take this notion of adversarial regularization, making the latent space look like a specific distribution, and extend it to a wide range of other uses. Since the only requirement that is placed on the prior distribution is that you have to be able to sample from it, this means that you can make it as wild as you want, and this lets you introduce things like semi-supervised learning and style disentanglement.

For example, if there's labeled data available, it might be nice to try to make the encoder jump through yet another hoop: in addition to producing codes that are indistinguishable from the prior distribution, we also might want the prior distribution to have a class-based structure. For instance, we might want each mode in a mixture distribution to correspond to a particular digit, as below:

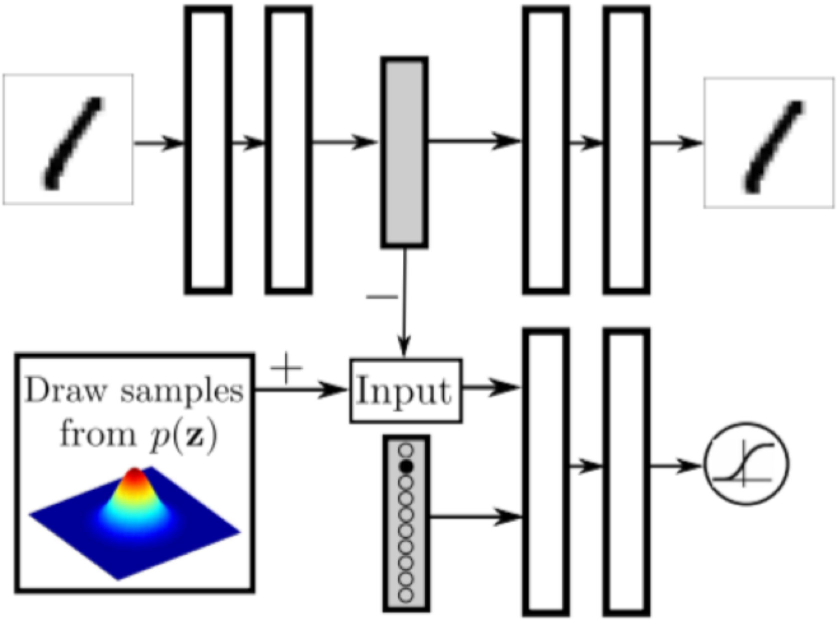

Figure 5. Adds a one-hot vector containing labeling information with an extra label for a sample with an unknown classes (from Makhzani et al. 2016).

By giving the discriminator a 'hint' in the form of the class label, and then only sampling from the mode of the distribution associated with that digit when providing prior samples, it forces the encoder to map a particular digit to a particular mode. If the generator produces a code that maps to the wrong mode, the discriminator can tell it came from the generator and not the prior, and so provide a very strong signal to the generator about how to change its output.

Style disentanglement works a similar way, except in this case the reconstruction network gets the 'hint' about the class label; this effectively relieves the latent space of carrying any label information and allows it to model other factors about the data that are important to reconstruction.

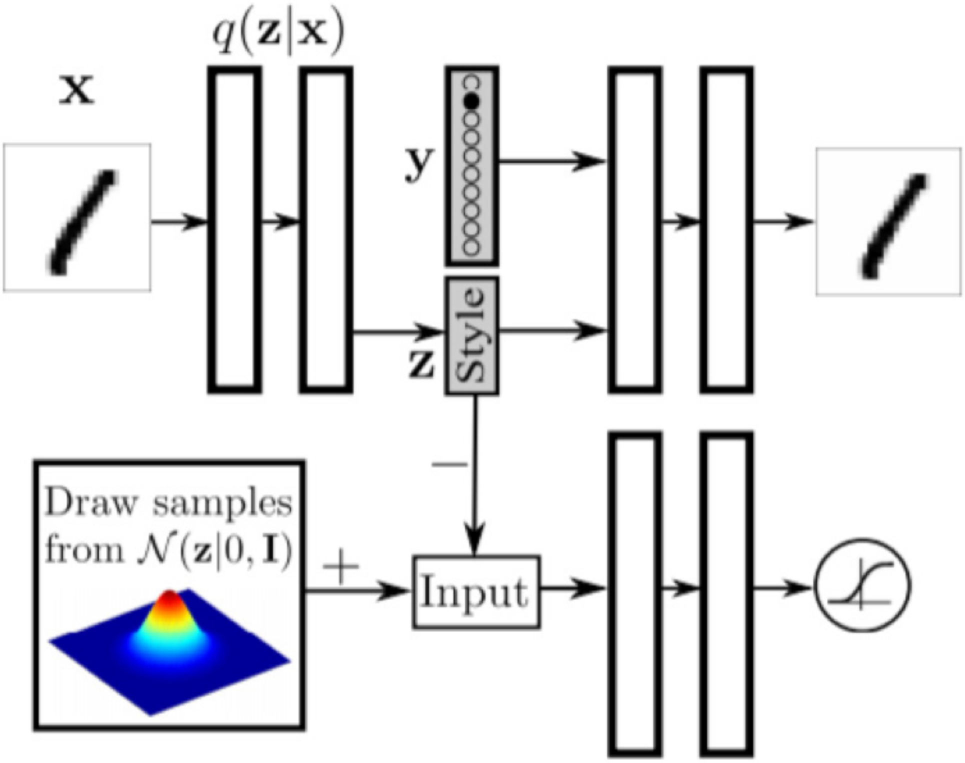
$q(\mathbf{z}|\mathbf{x})$
$\mathbf{x}$
$\mathbf{y}$
$\mathbf{z}$
Style
−
Draw samples from $\mathcal{N}(\mathbf{z}|0,\mathbf{I})$
+
Input

Figure 6. One-hot vector provided to generative model and the hidden code in this case learns to represent the 'style' of the image (from Makhzani et al. 2016).

Finally, by forcing the encoder to predict both label and 'style' information, and making sure that the label information is also adversarially regularized to produce something that looks like a softmax vector, we can do both semi-supervised learning and unsupervised clustering. For semi-supervised learning, we apply a classification loss on the encoder's label output whenever we have it, and trust the adversarial regularization to make it look like a softmax output when there's no label information. For unsupervised clustering, you add another 'cluster' layer after the softmax and, instead of a supervised signal, apply a penalty whenever the clusters get too close to each other.

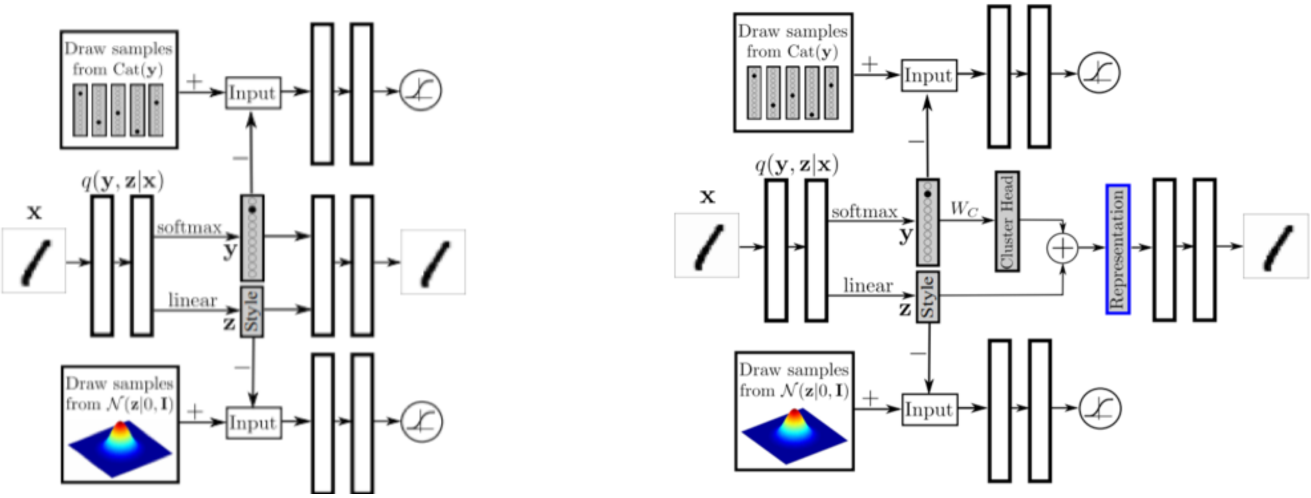
Draw samples from Cat($\mathbf{y}$)
+
Input
−
$q(\mathbf{y},\mathbf{z}|\mathbf{x})$
$\mathbf{x}$
softmax
$\mathbf{y}$
linear
$\mathbf{z}$
Style
Draw samples from $\mathcal{N}(\mathbf{z}|0,\mathbf{I})$
Draw samples from Cat($\mathbf{y}$)
+
Input
−
$q(\mathbf{y},\mathbf{z}|\mathbf{x})$
$\mathbf{x}$
softmax
$\mathbf{y}$
W_C
Cluster Head
Representation
linear
$\mathbf{z}$
Style
Draw samples from $\mathcal{N}(\mathbf{z}|0,\mathbf{I})$

Figure 7. On the right: Semi-Supervised AAE. On the left: Dimensionality Reduction with AAE: Clustering (from Makhzani et al. 2016).

Some issues:

While the AAE framework is extremely flexible, training AAEs can be tricky, and there's not a lot of discussion in the paper about how to get it to work well.

First, it's important to "balance" the training process for the discriminators and the generator/decoder. If either one gets too far ahead too early, they can dominate the training process from that point forward. Either the adversarial regularization grabs the wheel and refuses to let the encoder learn anything that the decoder can use, or the encoder/decoder dominates training and never adapts the latent code to match the prior.

Second, the prior has to match your data if you're adding a supervised component. There's a note in the paper indicating that they balanced the MNIST classes so that they were uniform, and this turns out to be critical to performance. If you have, for instance, a uniform prior across classes but an imbalance in your training data when you're doing semi-supervised training, then the generator has two competing objectives: it has to accurately predict the (imbalanced) class, but it also has to produce a balanced class distribution to trick the discriminator. This makes the training unstable, you're often off to NaNville pretty quickly after that. This is even more pronounced if you want to use supervision on a multivariate component; you need to accurately model any covariance between components in your prior, otherwise you once again make it very easy for the discriminator to distinguish between the generated samples (which are constrained by supervised loss to have some covariance structure conditional on your inputs) and the prior (uncorrelated).

Finally, the competition between the generator and discriminator networks means that there can be whipsaw changes in losses that can cause gradients to spike unpredictably. Simple SGD often recovers from this without too much trouble, but using optimizers with some sort of memory (SGD+Momentum, pseudo-second-order methods like ADAM or Adagrad) can often be thrown by this, and once again carry you off to the Land of NaNs. Using SGD with a low learning rate can largely mitigate this, but also leads to slow training. Finding a good balance is tricky, and seems to require a lot of parameter twiddling on small test sets up front.

Conclusion

AAEs are a clever idea leading to a flexible and general framework for a lot of interesting tasks. Training them can be tricky, but the core idea of using adversarial components to regularize latent states to conform to a particular distribution is a neat tool to have in your toolbox when you're looking at modeling problems.

United Kingdom and Worldwide Sales
Tel: +44 (0)8447 671131
Email: sales@sophos.com

North American Sales
Toll Free: 1-866-866-2802
Email: nasales@sophos.com

Australia and New Zealand Sales
Tel: +61 2 9409 9100
Email: sales@sophos.com.au

Asia Sales
Tel: +65 62244168
Email: salesasia@sophos.com