MENU

Variational Autoencoders Explained

06 AUGUST 2016

In my previous post about generative adversarial networks, I went over a simple method to training a network that could generate realistic-looking images.

However, there were a couple of downsides to using a plain GAN.

First, the images are generated off some arbitrary noise. If you wanted to generate a picture with specific features, there's no way of determining which initial noise values would produce that picture, other than searching over the entire distribution.

Second, a generative adversarial model only discriminates between "real" and "fake" images. There's no constraints that an image of a cat has to look like a cat. This leads to results where there's no actual object in a generated image, but the style just looks like picture.

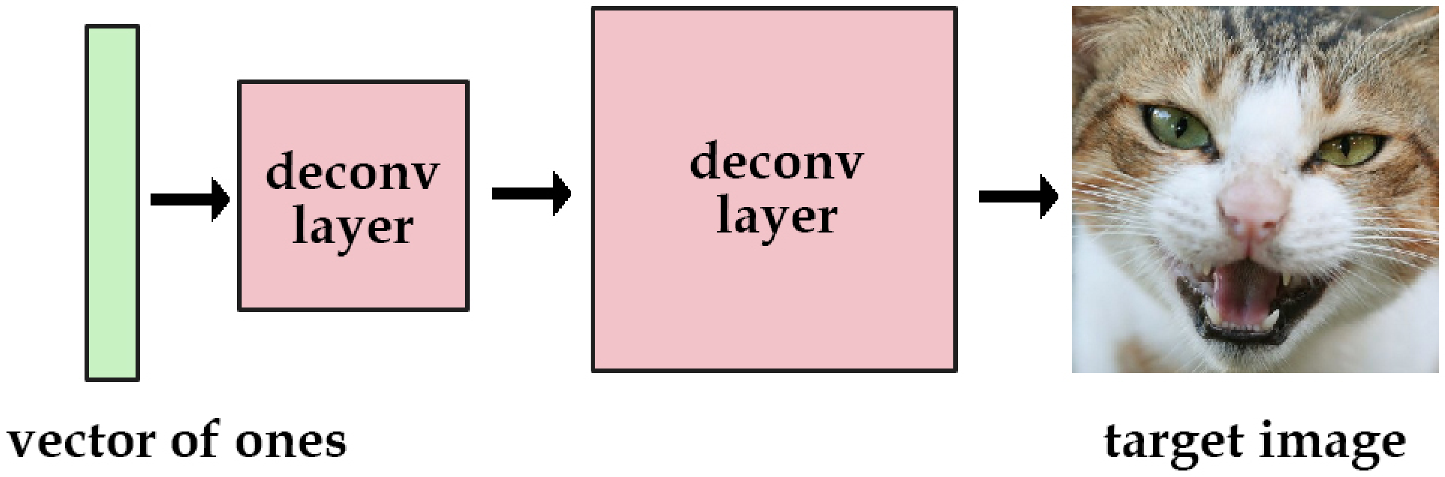
In this post, I'll go over the variational autoencoder, a type of network that solves these two problems.

What is a variational autoencoder?

To get an understanding of a VAE, we'll first start from a simple network and add parts step by step.

An common way of describing a neural network is an approximation of some function we wish to model. However, they can also be thought of as a data structure that holds information.

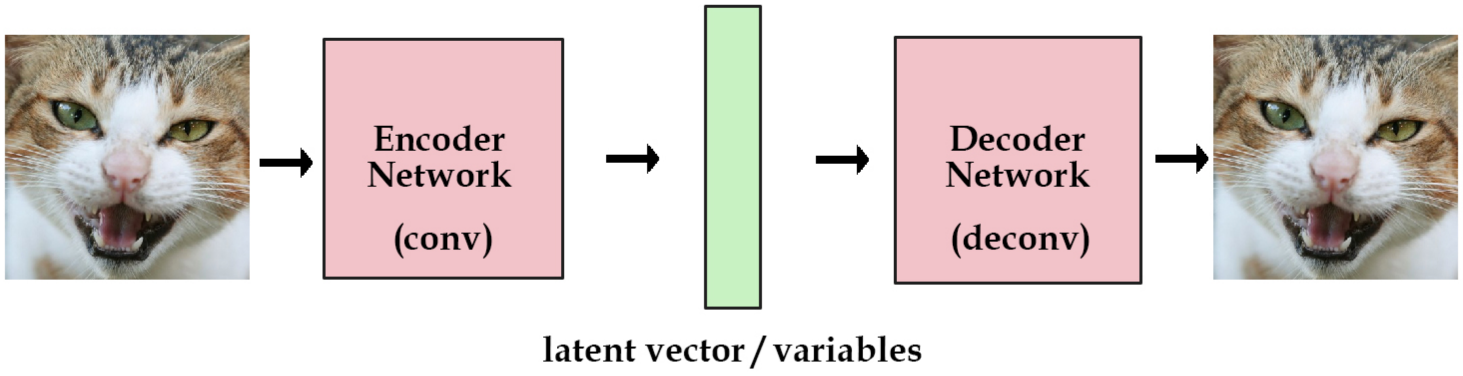
Let's say we had a network comprised of a few deconvolution layers. We set the input to always be a vector of ones. Then, we can train the network to reduce the mean squared error between itself and one target image. The "data" for that image is now contained within the network's parameters.



Now, let's try it on multiple images. Instead of a vector of ones, we'll use a one-hot vector for the input. [1, 0, 0, 0] could mean a cat image, while [0, 1, 0, 0] could mean a dog. This works, but we can only store up to 4 images. Using a longer vector means adding in more and more parameters so the network can memorize the different images.

To fix this, we use a vector of real numbers instead of a one-hot vector. We can think of this as a code for an image, which is where the terms encode/decode come from. For example, [3.3, 4.5, 2.1, 9.8] could represent the cat image, while [3.4, 2.1, 6.7, 4.2] could represent the dog. This initial vector is known as our latent variables.

Choosing the latent variables randomly, like I did above, is obviously a bad idea. In an autoencoder, we add in another component that takes in the original images and encodes them into vectors for us. The deconvolutional layers then "decode" the vectors back to the original images.



We've finally reached a stage where our model has some hint of a practical use. We can train our network on as many images as we want. If we save the encoded vector of an image, we can reconstruct it later by passing it into the decoder portion. What we have is the standard autoencoder.

However, we're trying to build a generative model here, not just a fuzzy data structure that can "memorize" images. We can't generate anything yet, since we don't know how to create latent vectors other than encoding them from images.

There's a simple solution here. We add a constraint on the encoding network, that forces it to generate latent vectors that roughly follow a unit gaussian distribution. It is this constraint that separates a variational autoencoder from a standard one.

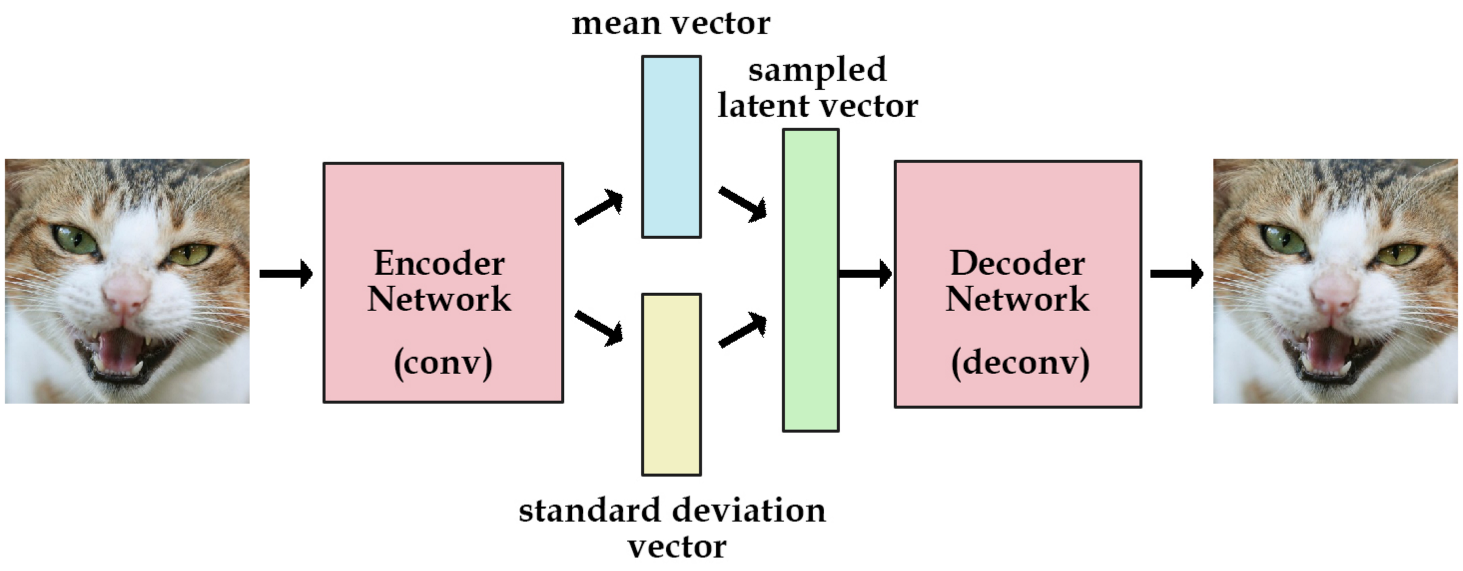
Generating new images is now easy: all we need to do is sample a latent vector from the unit gaussian and pass it into the decoder.

In practice, there's a tradeoff between how accurate our network can be and how close its latent variables can match the unit gaussian distribution.

We let the network decide this itself. For our loss term, we sum up two separate losses: the generative loss, which is a mean squared error that measures how accurately the network reconstructed the images, and a latent loss, which is the KL divergence that measures how closely the latent variables match a unit gaussian.

```
generation_loss = mean(square(generated_image - real_image))
latent_loss = KL-Divergence(latent_variable, unit_gaussian)
loss = generation_loss + latent_loss
```

In order to optimize the KL divergence, we need to apply a simple reparameterization trick: instead of the encoder generating a vector of real values, it will generate a vector of means and a vector of standard deviations.



This lets us calculate KL divergence as follows:

```
# z_mean and z_stddev are two vectors generated by encoder network
latent_loss = 0.5 * tf.reduce_sum(tf.square(z_mean) + tf.square(z_
```

When we're calculating loss for the decoder network, we can just sample from the standard deviations and add the mean, and use that as our latent vector:

```
samples = tf.random_normal([batchsize,n_z],0,1,dtype=tf.float32)
sampled_z = z_mean + (z_stddev * samples)
```

In addition to allowing us to generate random latent variables, this constraint also improves the generalization of our network.

To visualize this, we can think of the latent variable as a transfer of data.

Let's say you were given a bunch of pairs of real numbers between [0, 10], along with a name. For example, 5.43 means apple, and 5.44 means banana. When someone gives you the number 5.43, you know for sure they are talking about an apple. We can essentially encode infinite information this way, since there's no limit on how many different real numbers we can have between [0, 10].

However, what if there was a gaussian noise of one added every time someone tried to tell you a number? Now when you receive the number 5.43, the original number could have been anywhere around [4.4 ~ 6.4], so the other person could just as well have meant banana (5.44).

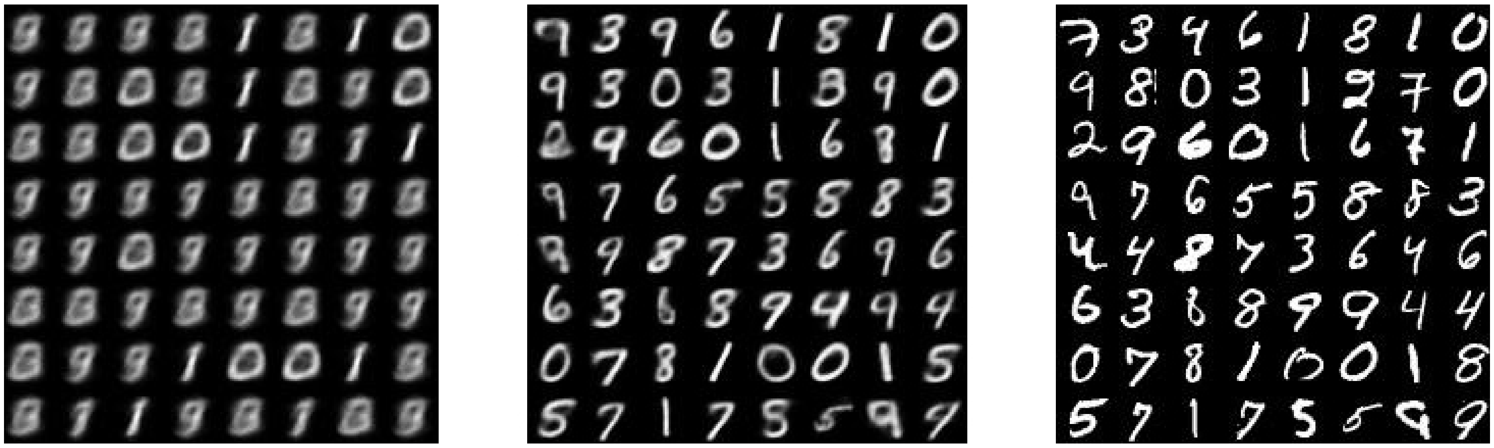
The greater standard deviation on the noise added, the less information we can pass using that one variable.

Now we can apply this same logic to the latent variable passed between the encoder and decoder. The more efficiently we can encode the original image, the higher we can raise the standard deviation on our gaussian until it reaches one.

This constraint forces the encoder to be very efficient, creating information-rich latent variables. This improves generalization, so latent variables that we either randomly generated, or we got from encoding non-training images, will produce a nicer result when decoded.

How well does it work?

I ran a few tests to see how well a variational autoencoder would work on the MNIST handwriting dataset.



> *left: 1st epoch, middle: 9th epoch, right: original*

Looking good! After only 15 minutes on my laptop w/o a GPU, it's producing some nice results on MNIST.

Here's something convenient about VAEs. Since they follow an encoding-decoding scheme, we can compare generated images directly to the originals, which is not possible when using a GAN.

A downside to the VAE is that it uses direct mean squared error instead of an adversarial network, so the network tends to produce more blurry images.

There's been some work looking into combining the VAE and the GAN: Using the same encoder-decoder setup, but using an adversarial network as a metric for training the decoder. Check out this paper or this blog post for more on that.

You can get the code for this post on my Github. It's a cleaned up version of the code from this post.

Comments Community 1 Login

Recommend 94 Tweet Share Sort by Best

Join the discussion…

LOG IN WITH

OR SIGN UP WITH DISQUS ?

Name

Pei Guo • 2 years ago

Best explanation of VAE I have ever seen! Thanks!

31 • Reply • Share ›

Adam Becker • 3 years ago

There's a reason I hate reading research papers, they tend to complicate simple things. We need more articles like this.

25 • Reply • Share ›

acheshkov → Adam Becker • 3 years ago

to write such excelent explanations, someone have to read and deep understand articles you hate)

16 • Reply • Share ›

Adam Becker → acheshkov • 3 years ago

The point is the original writers should make things simpler, like writing papers in a casual style instead of filled with cryptic academia stuff. Saves time and potentially money. However such change would be unlikely to happen :(

9 • Reply • Share ›

Michael Chan → Adam Becker • 2 years ago

The paper actually condenses much more material than what is written in this and any other blog, and that's perfectly fine! Each has its own scope. After reading this very useful article, I can go back to the paper and understand even more.

5 • Reply • Share ›

we evil ➦ Adam Becker
• 2 years ago • edited

Academic papers require certain constraints such as page-limits(CVPR- 8 pages) with additional pages costing more . When you've to fit everything from the intro., background, references, you're left with very little space and it's easier to use complex vocabulary and mathematical equations to explain the paper.

Of course, they could follow it up with a blog post explaining it lucidly. Unfortunately, that isn't the norm.

2 ^ | v • Reply • Share ›

Ryan Daly ➦ Adam Becker
• 8 months ago

I've worked writing simulation science papers for a while. There's a tendency to write too formally sometimes. It frequently gets in the way of the wider audience, which is something completely necessary today in multidisciplinary research.

^ | v • Reply • Share ›

Zack Chase Lipton ➦ Adam Becker
• 3 years ago • edited

I don't mean to dismiss your complaint but to me it seems a bit misguided. The post that gives you the "gist" would be a lousy substitute for a research paper. Writing tutorials is valuable, and I've done lots of it myself. But its also a lot easier to tell someone how to do something than to justify and derive it from scratch. This comment is also just way too broad. There are very dense papers, say in theory. But that content would also be very difficult to spoon-feed in a tutorial. Then there are the high quality applications papers. And among these, the good ones tend to be exceptionally clear and easy to read.

9 ^ | v • Reply • Share ›

StoneCypher ➜ Zack Chase Lipton
• 2 years ago

I don't think this actually holds water

One of the biggest problems with research papers is that they stay so far in abstract jargon that practicioners often cannot implement what is being discussed

It took me three hours to figure a paper out this morning which can be usefully represented in two sentences

If papers started with those two sentences then carried on to the same style, the impact to accessibility would almost certainly be profound

27 ^ | ˅ • Reply • Share ›

Zack Chase Lipton ➜ StoneCypher • 2 years ago

I don't think you're accurately characterizing deep learning papers. Maybe some, but it's not the general trend. Take the batch normalization paper for example. The equations you need are even put in a box with a big square around it. Same is true for the GAN paper, the adversarial training paper. There are exceptions - getting all the implementations to replicate the DQN is actually hard, and they could have made that much clearer in a few ways. But it's also much harder to explain because there are so many essential implementation details (doesn't fit in 1-2 lines). If you really don't want to understand the paper and want the punch line it's not that hard to find in many/most of the good papers.

What the field really needs is more great textbooks. You don't learn math as an amateur or grade-school student by reading the primary literature. And perhaps it's not necessary for machine learning either. We're working on an effort to explain a good chunk of deep learning / machine learning clearly (and to teach MXNet at the same time) here:

Currently in 2nd week of development but moving fast.

https://github.com/zackchas...

21 ^ | ⌄ • Reply • Share ›

Cico ➜ Adam Becker • a year ago • edited

Sorry, I just disagree. The things that you find easy to understand are often simplified.

1 ^ | ⌄ • Reply • Share ›

Hadi Kazemi • 2 years ago

There is still sth that I didn't get it. The encoder gives us mean and std of the latent variables and we are forcing them to be a unit normal. So eventually, the encoder should only give us a zero and a one for all images ? and how can we expect the decoder to generate the exact image in the input when we randomly sample from a normal distribution? I think I didn't get how the mean and variance change for different picture !

6 ^ | ⌄ • Reply • Share ›

Mårten Nilsson ➜ Hadi Kazemi • 2 years ago

I was confused about this too. This blog post does not seem to mention the key aspect that makes variational autoencoders work. I looked further and found a nice illustration about this in this article: https://arxiv.org/abs/1606.... (Figure 4). The point is to apply the reparametrization trick (yes I know the term is used in this blog post, but the author seem to have misinterpreted it since the reparametrization is not applied to learn the KL divergence, it is used for the reconstruction loss) which allows gradients to flow from the decoder to the encoder.

The idea is this, instead of sampling from the mean and standard deviation of the encoder sample from a unit gaussian. Then multiply with the mean and standard deviation form the encoder (which is a differentiable operation) to obtain the random sample. Now the encoder can be trained both with the reconstruction loss and KL divergence. The reconstruction loss will constrain the encoder to not only output unit gaussian values.

4 ^ | ˅ • Reply • Share ›

Mårten Nilsson ➦ Mårten Nilsson • 2 years ago

I strongly recommend looking at the illustration in the article rather than reading my explanation. One short look was all I needed to get the "Ahaaa that is why it works" experience, and I think my explanation takes a bit longer to digest.

1 ^ | ˅ • Reply • Share ›

Totoka ➦ Hadi Kazemi • a year ago

The distribution of the latent variables over the entire population is to be unit normal, not the outputs for one specific sample.

2 ^ | ˅ • Reply • Share ›

Arash Moaddel • 2 years ago • edited

Thanks for the interesting post. So what I get is that the encoder doesn't actually produce the latent variable, but instead produces the mean and standard deviation of a normal distribution that we do sample from and then map it to image space. Right ?

What if I want to give the image to the encoder and get the latent variables themselves as the output ? I mean this learnt manifold plot, it is not actually plotting the data points based on the latent variables? Correct ? I assume you are plotting the mean of guassian distribution for each dimension ?

4 ^ | ˅ • Reply • Share ›

kvfrans Mod ➦ Arash Moaddel • 2 years ago

Yeah, the means are in general a good estimate for the latents given an input image

6 ^ | ˅ • Reply • Share ›

leonard Strnad • 2 years ago

Hey Kevin,

First, thanks for the code and article. Your tf implementation is really helping me see things from both sides: paper to end-to-end VAE. You mentioned:

"A downside to the VAE is that it uses direct mean squared error instead of an adversarial network, so the network tends to produce more blurry images."

If you refer to the original paper, you will see that the L2 norm or mse as the decoder loss is a consequence of specifying the assumed distribution on the likelihood is normal. That is, for the expected log likelihood component of the loss function we see that the log prob is actually c*exp(k*(x-x*)^2) (the gaussian) where x is the generated and x* is the actual. The log likelihood of this probability is c + k(x-x*)^2 ~ (x-x*)^2 which is were the mse comes from. We could assume a Bernoulli distribution over x (pixel on or off like black and white pixel images) and get a decoder loss like the cross entropy.

I just wanted to clarify it is not always the mse.

here is an example of a post that discusses cross entropy: http://blog.fastforwardlabs...

Cheers dude! keep i up!

3 ^ | ˅ • Reply • Share ›

Alessandro • 3 years ago

First, great article. My only suggestion is to explain what variational means: since you start with image generation, I would expect you to introduce, at a certain point, the concept of autoencoder and expand it to the concept of variational autoencoder.

Secondly, I am genuinely surprised by the results you get after only 9 epochs: today I was playing with a regular convolutional autoencoder, just 5 layers or so, and it took me 5k iterations to get a decent quality. I'm pretty sure I cannot get that result after only 9 epochs! I looked into your code and maybe it is because you're using 500 hidden units (I'm using 30 in the smallest hidden layer). Also, the article you mention at the end, from jmetzen, takes about 60 epochs. Do you have tips on improving the training time?

Thanks again, cheers!

2 ^ | ˅ • Reply • Share ›

Jenny • a year ago

Thanks for the nice post!
I have one question on the relationship between the input image and the latent variable though. It seems that during training the latent variable is sampled from a priori distribution. How does it guarantee that an input image is mapped to the same latent variable? For example, in the first iteration, image A is mapped to z1. Is it possible that in the fifth iteration, image A is mapped to z2 by encoder?

1 ^ | ˅ • Reply • Share ›

Martin Lang ➜ Jenny • a year ago

I'm not an expert in this field either but I think you are correct with your assumption that the mapping will change.

Since the a-priori distributions of the latent variables evolves during training, for a large portion of input images the Gaussian (latent variable) with the highest response will be a different one after training compared to before training.
And the more or less random initial mapping of an image to the index of latent variable with largest response will change therefore.

And I think that this is not a problem.
In the end this behavior is exactly what you are looking for.
After training similar Images will generate the largest response in the same latent variable.
Dissimilar images will generate their largest response (hopefully) in different ones.

Please let me know if i'm missing something here.

^ | ˅ • Reply • Share ›

Blessen George • a year ago

Great explanation!! :)

1 ^ | ˅ • Reply • Share ›

Joshua Patterson • 2 years ago • edited

Fantastic article. I've been pouring over VAE papers and tutorials and there is still something I'm not grasping. Hopefully you can help me understand.

Say you encoder compresses the input data to 2 vectors of length 3.

mean = [1.1, 2.2, 3.3];
variance = [4.4, 5.5, 6.6];
latent_variables = mean + variance; //[5.5, 7.7, 9.9]

As I understand it, each index of these vectors corresponds to a 'feature' in the data.

eg:
[0]=colour, [1] = line thickness, [2] = roundness .

Assuming the VAE trained well. When an input is passed through the encoder the mean vector will contain [mean_colour, mean_line_thickness, mean_roundness] of the input image (similarly for the variance).

I hope I'm correct so far.

However, during training, how does taking the imposing an zero mean, unit variance across the latent variables impose a cost on 'similar features'. Another way of phrasing it. Why does having a normally distributed latent variable vector promote the separation of features?

1 • Reply • Share ›

kvfrans Mod Joshua Patterson • 2 years ago

Since the decoder can't highly depend on any single number in the vector (since noise is added from variance), the decoder isn't likely to memorize any combinations of latent vectors, so the natural solution is to have each feature be relatively independent

• Reply • Share ›

Pratyush Tiwary • 2 years ago

This is hands down the best tutorial I have come across on VAE after struggling last ~3 weeks. Thanks so much! The code works and is very useful too.

1 • Reply • Share ›

Chien Duong • 2 years ago • edited

Recently, published paper "Autoencoding beyond pixels using a learned similarity metric" as mention in the "Check out paper" combined the advantage of both GAN and variational-autoencoders. The reconstructed image will be compare with original image by features wise metric (in Discriminator of GAN) instead of pitxel-wise metric .

1 ^ | ˅ • Reply • Share ›

ccc • 2 years ago

Nice post. Yet I feel it lacked a bit on key issues specifically related to VAEs over barebones AEs. Would have been nice to see not so much the reconstruction of the original samples (AEs can do this just fine) but generation of new ones by sampling the latent space. A bit on discussing how latent space dimensionalty affects the quality of newly-generated samples would have been a plus.

1 ^ | ˅ • Reply • Share ›

Alfredo Canziani • 3 years ago • edited

This was gold, Kevin. Thank you. I think I'll teach it in my next class.
P.S. Could you add a paragraph about the adjective "variational"? Is it about the fact that σ > 0?

1 ^ | ˅ • Reply • Share ›

Rouhollah • 3 years ago

Very easy to understand. Thanks a lot.

1 ^ | ˅ • Reply • Share ›

Suffian Khan • 3 months ago

Great explanation, thanks!

^ | ˅ • Reply • Share ›

JasOlean • 5 months ago

Great explanation. Do you think this VAE is good for noisy input?

^ | ˅ • Reply • Share ›

Antoine Savine • 5 months ago

A very clear explanation for a topic generally described in rather blurry terms. There are plenty of tutorials on GANs out there: GANs are generally easier to conceptualize and understand, although they may be tricky to implement. But very few clear and expressive tutorials like this one on VAEs.

^ | ˅ • Reply • Share ›

Elias Hasle • 10 months ago • edited

Thank you for a great and simple explanation of VAE. Just what I needed what I needed.

I am making an AE to learn TensorFlow and Sonnet. While training it, I encountered the problem that in many training runs it would learn to output a constant regardless of the input. Then I made a new version using the VAE trick of sampling the encoding from a distribution that depends on the input. I achieved 95% accuracy on MNIST (predicting from the encoding) very fast, so it worked. But then I discovered that I had not added the KL divergence to the loss. I suppose this means the network will still be motivated to learn the means right, and also to narrow down the standard deviations according to confidence. Early during optimization the larger standard deviations will contribute to some exploration, I suppose (similarly to Simulated Annealing).

Disappointingly, adding in the KL divergence gave much poorer results, or at least not in the same short time. I

Kevin Frans

Read more posts by this author.

Share this post

READ THIS NEXT

A intuitive explanation of natural gradient descent

A term that sometimes shows up in machine learning is the "natural

YOU MIGHT ENJOY

Simulating Twitch chat with a Recurrent Neural Network

Is it possible for a neural network to learn how to talk like humans?

gradient". While there hasn't been much of...

Recent advances in recurrent neural...