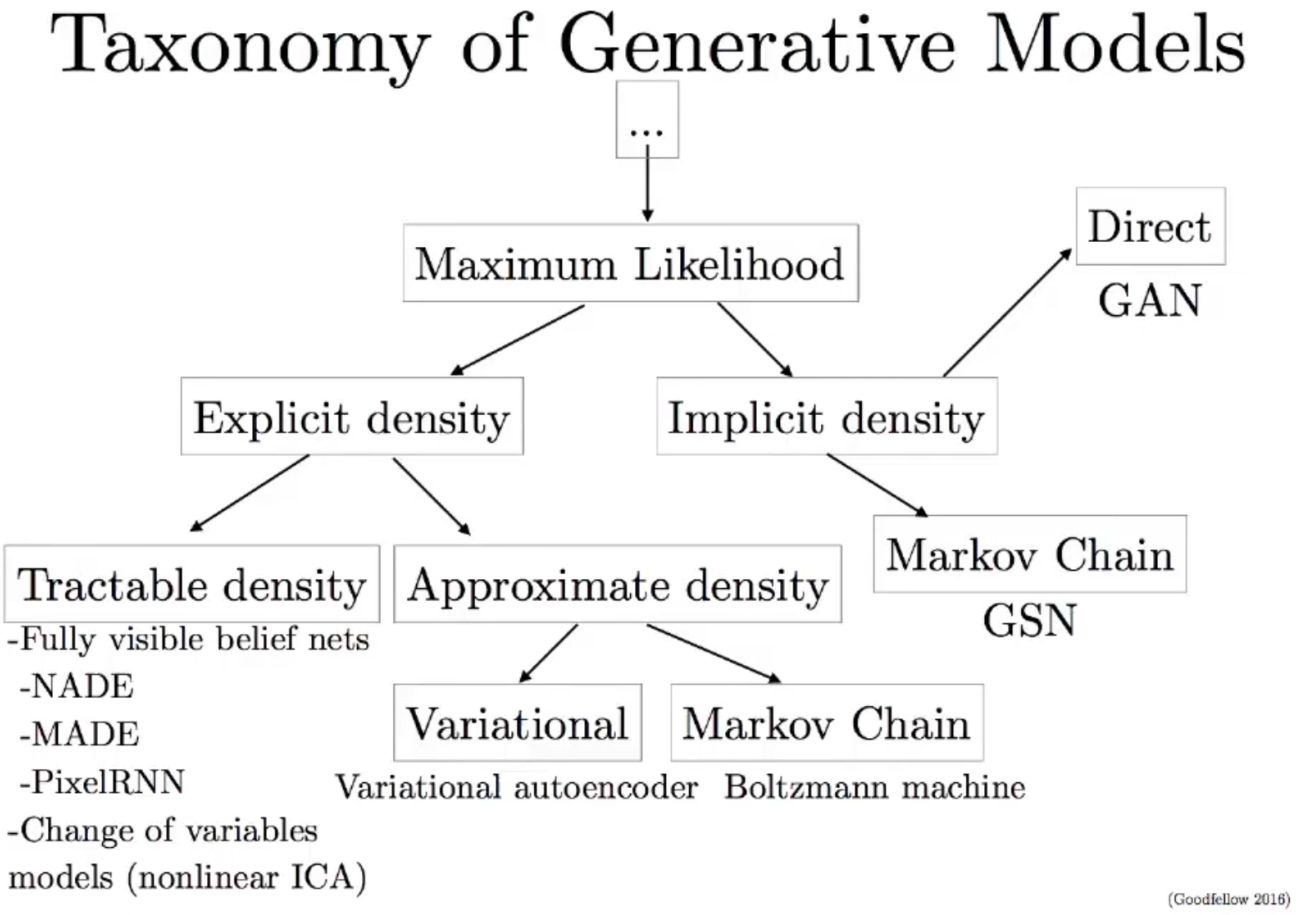
generative adversarial networks

- the focus of GANs is to generate similar data points from a certain distribution that we have as a dataset. the focus is not so much explicitly capturing the pdf/the distribution itself.
- uses of generative modeling:
  - for simulation to build other agents (RL agents, self driving cars etc)
  - an agent can use generative models to generate possible future states (for planning)
  - able to handle missing data and fill in missing inputs according to their distribution in observed data
  - realistic generation tasks, eg. speech generation (text to speech)
  - multi-modal generation. eg. lets say we want to generate the next video frame from the current video frame. traditional methods (eg. mean squared error) might take a few possible frames and average them to reduce loss, resulting in a blurry mess. another example of this is super-resolution of images.
  - mathematically, they are also a tool to capture probability distributions in very high dimensional spaces.
- Most of generative modeling is formulated as maximum likelihood problems, just like we saw with variational auto-encoders.
- here's an overview of the space of generative models that use maximum likelihood:



- The right hand side of the tree is a family of methods that design a structure from which we can draw samples from the probability distribution without explicitly modeling and optimizing the probability distribution, like we did in VAEs. GANs belong to this category.
- which branch of the tree should we use?
  - there are examples of the tractable density class of models that produce super good samples but take a very long time to compute (because of burning time in MCMC?). an example is wavenet (deepmind later launched a version of wavenet in google assistant) → look into WaveNets since deepmind now claims to make them work.
  - Ian provides a very nice critique of VAEs, for which this inequality is true:
    $log\, p(x) \geq log\, p(x) - D_{KL}(q(z)||p(z|x))$
  - Since the RHS above is the function we are optimizing, unless q, which is a gaussian with $\mu(X), \sigma(X)$ is the right model, even if we have a perfect optimizer and infinite data, we will not get to the right probability. in practice, this results in producing much lower quality results.
    For VAEs to be asymptotically consistent, we need space of distributions of q to contain the actual posterior of z.
  - Boltzmann machines are another example of intractable (because the normalization constant is intractable to compute) family of models but with explicit representation of the distribution. These have an energy function as discussed in the previous chapter. Inference can be done with monte carlo methods (like MCMC) is poor because of problems with mixing between modes and in general, they don't scale to high dimensional spaces.
- The above described flaws express design requirements for GANs:
  - use a latent code (just like VAEs and Boltzmann machines)
  - asymptotically consistent (unlike variational methods)
  - no markov chains needed, no mixing → this was motivated by the fact that ian found these to bring down other models like boltzmann machines.
  - samples need to be really good (empirically)
- GANs have two networks: a generator and a discriminator that are adversaries, in the game-theoretic sense → each has well defined payoffs.
- the generator takes a latent vector z (which is a source of randomness which helps the generator generate a wide range of images/whatever we are interested in generating). its job is to generate realistic looking things that fool the discriminator
- the discriminator's job is to assign a probability of whether the generator's output is realistic or not.
- Formally, we start with latent random vector z (this z is sampled from a prior and is the source of randomness), the generator computes its image $G(z)$. The discriminator outputs $D(G(z))$ which is a probability in $[0, 1]$.
- For asymptotic consistency of the learnt distribution, we need to make sure that z has equal or higher dimension than x. This is to ensure that z has enough capacity to capture the actual distribution.
- Jensen-Shannon divergence is a symmetrized, smooth version of KL divergence defined as:
  $M = \frac{1}{2}(P + Q)$
  $JSD(P||Q) = \frac{1}{2}D_{KL}(P||M) + \frac{1}{2}D_{KL}(Q||M)$
- "As a rule of thumb, if you're goal is to generate something which is of high likelihood, then you are better off using a VAE. If the goal is generate realistic samples, GANs are usually better". - goodfellow. This begs the question: *why does high likelihood not capture the realistic-ness goal? This is because the GAN is designed with the explicit goal of fooling the discriminator. The VAE on the other hand is designed to increase the likelihood of the produced images. if our notion of likelihood is not perfect (realistic looking), we can get a bunch of "high likelihood" images that don't look realistic.*

- one way to sample data from dataset (for discriminator training) and from z space is to do uniform distribution sampling from both.
- **Basic formulation:** GANs are trained using a minimax game between the discriminator and the generator. The discriminator's score is based on how well it differentiates between whether an image came from the data or whether it came from the generator. The generator's score is the negative of the discriminator's score

$$J^{(D)} = -\frac{1}{2}\mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}} \log D(\boldsymbol{x}) - \frac{1}{2}\mathbb{E}_{\boldsymbol{z}} \log\left(1 - D\left(G(\boldsymbol{z})\right)\right)$$
$$J^{(G)} = -J^{(D)}$$

- In other words, the generator minimizes the log probability of the discriminator being correct.
- Note that cross entropy loss functions are designed with the property that when its doing badly, the gradients don't vanish and they get small when its doing well.
- There is a problem with the above formulation, which is that when the generator is not doing well and the discriminator is doing very well, since the loss of the generator is just the negative of the loss of the discriminator, its gradients also vanish and this is bad because the generator just gets stuck at a bad point.
- **A better formulation:** In order to get around the above described problem of bad gradients, we can instead formulate the generator loss to maximize the log probability of the discriminator being mistaken (now only on the images generated by the generator). This is in contrast to minimizing the log probability of the discriminator being right about images that G generated.

$$J^{(D)} = -\frac{1}{2}\mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}} \log D(\boldsymbol{x}) - \frac{1}{2}\mathbb{E}_{\boldsymbol{z}} \log\left(1 - D\left(G(\boldsymbol{z})\right)\right)$$
$$J^{(G)} = -\frac{1}{2}\mathbb{E}_{\boldsymbol{z}} \log D\left(G(\boldsymbol{z})\right)$$

- With both the above formulations, the magnitude of loss slopes in the same direction, its just that now the slopes are different.
- The above is derived from a heuristic argument as stated above. This is the default cost function that Ian Goodfellow usually uses/advocates. it is also referred to as the "non-saturating cost".
- A lot of GANs work well only on domains with a few output modes (like a GAN trained for pictures of faces, one for pictures of bedrooms, etc.)
- Goodfellow shows an example where we can do algebra in latent space and have that correspond to semantic meaning. For example, take the latent space representation of an image of a man with glasses, subtract the latent space representation of an image of a man, add the latent space representation of an image of a woman and run that thru the generator, and we get an image of a woman with glasses!
- In general, you want the discriminator to be very very good at what it does.

- Practical tips and tricks:
  - labels seem to improve sample quality. this means that learning conditional samples $p(y|x)$ gives better samples for all classes rather than learning $p(y)$. this defines three categories of models → no labels, trained with labels, generating conditioned on labels.
  - batch norm seems to really helps, except in the last layer.
  - Use of one-sided label smoothing (smooth the data labels). Label smoothing has been shown to help against adversarial attacks → which is what the generator can be thought of as, an adversarial attacker. The intuition behind why label smoothing might be working well has to do with the fact that a classifier gets really confident by some linear increases in some inputs. the label smoothing basically kind of tell the classifier to “yo chill, don’t be over confident”.
  - The reason label smoothing needs to be one sided is because in a region where $p_{data}(x)$ is very small and $p_{model}(x)$ is larger, $D^*(x)$ will have a peak near the spurious mode of $p_{model}(x)$. The discriminator will thus reinforce incorrect behavior in the generator; the generator will be trained either to produce samples that resemble the data or to produce samples that resemble the samples it already makes.
- one reason why overfitting may not be a problem in GANs is because the generator never directly sees the training examples. it only sees them via the gradient communicated by the discriminator. so if you can regularize the discriminator to not just memorize the training data, then you should be good.