[SQUEAKING]

[RUSTLING]

[CLICKING]

PHILLIP ISOLA: So welcome. Happy Halloween. Nice to see a few little costumes. So today, we're going to talk about generative models. Again, this will be our third in the sequence of lectures on generative models.

And last lecture was a little bit math-heavy, a little bit notation-heavy, a little bit of a slog. This one, I hope, will be a bit more fun and lightweight. We're going to get to see some applications of that math.

So here is the topics that we'll cover in this lecture. We'll talk about the idea of structured prediction. And I will go through a few different examples in different domains. All the domains will be images and text because those are the two domains where this kind of technology is best developed. But this technology will, in the future, and is currently starting to be used in other domains, too.

But I'll talk about mapping images to images, text to text, image to text, text to image. And then I'll end with a short section on, can we do these mappings without having paired data, without having supervision of the form of paired data-- in the form of paired data?

So when I first encountered generative models, it took me a little while to understand why they are an important thing, especially if you come across them in this generative AI era when it was like-- your first experience might have been you're making up beautiful images or writing funny stories. And that feels like it might be very useful for entertainment or there's some kind of frivolous art form. But really, I think they're just fundamental. They're core to almost everything.

I think I've made that point in the previous lectures. But I think it will be most obvious in this lecture, where I'll frame all problems that are predicting high-dimensional structured data or information out of a network as being appropriate to be treated with generative modeling.

So we can call these data prediction problems-- in statistics and other areas, this is sometimes called structured prediction. So the problem of outputting something that's not just a single number or a single decision, but is actually some high-dimensional structured object-- this is the problem that generative models are well suited to. And this is the problem, which is basically ubiquitous.

It's hard to imagine almost any problem that you'll come across in your own work which would not be structured prediction. It's a weird historical artifact that we had this term, structured prediction. This is just everything. But I'll define it on the next slide.

But here are some examples. So imagine in and image out-- so this is like the problem of detecting objects. It's a high-dimensional structured object which you're trying to predict, which is an array of pixels that have a class assignment for every input pixel.

Audio-- and it's structured data in, structured data out. The output is a sentence. It's not just a number or a single decision, a class. It's a whole sentence. This is speech recognition or automatic speech transcription. Another one that's quite popular is describing an image in text, or maybe a video in text, and outputting the entire image or video-- so text-to-photo, which you may have played with in things like Stable Diffusion or Midjourney or Dall-E.

Another example is a input sequence of some kind of protein sequence and output the 3D structure of that protein. So again, you're not just trying to give one label to, what is this protein? You're trying to output a structured object. The entire geometry is a high-dimensional object. And it's an object that has structure.

And what do I mean by having structure? This is going to be the definition of structured prediction. It means that I'm modeling a high-dimensional object X. So my output that I'm trying to predict is a high-dimensional object X. And it's an object which doesn't factorize into independent-- a product of these marginals.

So unstructured prediction was if I want to model some high-dimensional data, I will assume that all the dimensions are independent statistically. So independent random variables means that the probabilities factorize as a product. So the joint distribution equals a product over each dimension. And if I don't have independence, I don't have a product. And then we call it structured prediction.

And of course, you-- it's hard to imagine any setting where you really have independence over your dimensions of your high-dimensional structured-- of your high-dimensional object you're trying to predict, like text or, if it's a robot, the trajectory it has to move its arm. And these are all high-dimensional structured objects.

If I'm a self-driving car, how I navigate the city-- it's like, every decision is not independent of all the other decisions. There's joint dependencies between those decisions. If I turned left, that affects what my next decision should be.

So this is the setting that we're in. But let's first go back and see how people approached these data prediction problems with unstructured models because even though it seems like a crazy idea now, it was simple. And it's not terrible. There's reasons why that can work.

So here's the unstructured way of modeling the colorization problem. So I love this problem because it captures all the key concepts in machine learning, in my opinion. I don't actually care that much about colorization as a graphics problem. But I've often used this as an example because it just has all the right properties.

So this is trying to predict the color of this shirt here. So what color do you think it is? Who has the answer? No one. You don't know-- could be red. It could be orange. This is the grayscale version of the shirt.

So what color is it? Well, maybe I know that this shirt is manufactured by some company, and this shirt comes in two different colors. I am going to imagine that the true probability of color of this grayscale shirt conditioned on the luminance value, the grayscale value-- this is going to be the data distribution that is the way the actual universe works. So this is the true distribution that describes this data. I take shirts that are teal and shirts that are pink with equal probability. I convert them to a grayscale image. And that's my observation.

So if I just try to predict the color using the classical regression method, which is just output a single number for-- a single continuous number, what color do I think that shirt is-- let's imagine color is a one-dimensional space that goes from teal to pink. Well, with this point prediction method, you can interpret that, if you want to, as a posterior distribution over all the possible colors that this shirt could take on. But you're outputting a delta function. And it doesn't express anything about your uncertainty over what the range of colors could be. And it can't capture this multimodal distribution.

So in particular, if you are doing least squares regression to output a scalar value, which is going to be your prediction of color, the minimizer of the least squared objective over a data-- a training distribution that looks like this is going to be the average of the two colors. So the average is the minimizer of the sum of squared deviations from a-- the average of a set of data points is a minimizer of the squared deviation from all the data points.

So that's something to lock-- slot away in your brain. It's an important property that comes up a lot. The mean or the expected value of the data distribution, the mean of that data distribution, is going to be the minimizer of the least squares objective regression problem.

But notice the problem is that the mean has zero density under the true data distribution. So the mean is a terrible estimate of the color. Yes, if you predict the color, then your error will-- that's a single prediction that minimizes the difference from the truth. Just say, well, if I'm here, I would sometimes get it wrong because I would have-- it would have been teal, and I predicted pink.

So I'll get half the time right, half the time wrong. But if I square the distance, then the wrong answers will dominate. So halfway in between is going to be the best answer for minimizing my error. But it's a terrible answer because that has zero probability under the data distribution.

So we could do a little bit better by saying, no, let me try to model some uncertainty. So I'm not going to have a true probabilistic model. I'll try to have a max likelihood density fit to my data distribution, my training data. And it will be a Gaussian in shape. And that's great because now I am modeling the variance, the uncertainty over what the range of colors is.

But this is-- this Gaussian is insufficient because it's unimodal distribution. And the true data distribution that I'm trying to model is bimodal. So you can't fit a bimodal distribution with a unimodal distribution.

So one very common trick is to simply switch to a categorical distribution. So this is the hack that I mentioned in the hacker's guide lecture-- that oftentimes, if you just quantize your data, chop it up into discrete classes, and then model a categorical distribution over the probability of each class, it will work really well. And here's one of the reasons why it works really well-- because the categorical distribution, which is just a set of numbers that sum to 1, can express bimodal distributions.

So the categorical distribution is fully expressive over this quantized space. You can represent any number of bumps. So the categorical distribution actually completely worked for solving this problem up to quantization error. So the only issue is that I had to quantize my colors. So now I don't have a continuous exact, precise color I'm outputting. I'm outputting that it's the teal class. And there's some quantization error.

So the categorical distribution is pretty good. And that's one reason why we often use softmax regression, because that models a categorical distribution over a set of classes. So that solved the problem of multimodality.

So one problem with prediction is that there could be a multimodal distribution over outcomes, meaning more than one bump. And if that's the case, then regression-- least squares regression does not work. Point prediction does not work. But actually, classification does work. But there's another problem which classification suffers from.

So this was only to predict one color. And if I really want to colorize that whole shirt, I need to predict the color of every single pixel in that shirt. So my output is actually a high-dimensional object. It's not just one value in this colorization problem. It's going to be one value per pixel in the input image.

So here's where classification fails. So classification, meaning outputting a-- the-- my softmax distribution over the possible classes, will give me, in this case, equal probability of teal and pink. And at this pixel, I'll say, it's slightly more likely to be teal. Maybe this-- the picture actually has a little bit of shading and there's just 51% probability of being teal at this location, for whatever reason, because there's a little shading right there that indicates it's a little darker, in some sense.

Over here, maybe it's a bit brighter illumination. But I now say it's-- well, it's slightly more likely to be pink. There's some kind of dependency between the brightness of the shirt and the color. And that means it could be either color, but it's slightly more likely to be pink.

And this will just always happen. You'll always have your conditional distribution over your predictions being spatially varying to some degree if I actually have underlying-- the truth is almost equal probability, or in this case, I said the truth is exactly equal probability, of teal and pink.

So my estimates of that probability will be a little bit noisy. And they'll vary from pixel to pixel. And what will happen then is I won't have consistency between how I colorize the top of the shirt and the bottom of the shirt because the probabilities smoothly varied. But they crossed this threshold where the argmax probability changed.

So this means that I'm basically failing to capture the joint dependency between this prediction and that prediction. The joint dependency says that they should be the same color. But if I were to classify each pixel independently, I would fail to capture that.

So generative models get around this. The set of generative models we've talked about get around this because they have two important properties. One is-- well, this is almost like the definition-- when we talk about generative AI, generative models, this is what we mean. These are the two properties we mean.

They can model a multimodal distribution. So it's not just one Gaussian, but they can model a complicated posterior. And two is they model joint dependency. So it's a high-dimensional multimodal distribution where the dimensions all have dependencies between them.

And if you have a model that can do this, then we usually will think of that as a generative model. And the VAEs and GANs and diffusion models all can do this. So that's the motivation. Any questions about that?

So to me, it's just strange-- it was only in the last few years that people really had models that can do both of these things in practice. In theory, we've thought about this for decades. But this is just so important. Every problem has this characteristic. So these generative models are just the thing that you really want to use for, I think, almost everything.

So now that you saw those two flaws with unstructured prediction, treating every pixel independently and outputting a continuous real number for the-- those pixels, let's look at some photos that might be a real photo, or it might be a colorized black and white photo that was-- an AI system added color to. And so I want you to guess.

So raise your hand if you think that this is a real photo with real colors. A few people. Raise your hand if you think it's a fake photo with AI-generated colors. A lot of people. So somebody who said fake, tell me, what artifact, what failure mode do you see that's giving that away? Let's go here.

AUDIENCE: It has pink down the paint-- the cup.

PHILLIP ISOLA: Yes.

AUDIENCE: So it might not know what color its should be.

PHILLIP ISOLA: So it doesn't know. So it's oscillating. Exactly. There's pink down here. And there's blue here. And that's exactly this effect. This system is making independent predictions per pixel. And it crossed the threshold of probability. And it switched to the other mode.

It thinks this is maybe red paint or maybe blue paint. It's pretty sure it's one or the other of those things, but it doesn't know exactly which one. And then it just chaotically oscillates when the probability is very, very slightly.

Anyone see any other issues that may-- that give it away? Let's go here. Yeah?

AUDIENCE: There's a colored shadow underneath it.

PHILLIP ISOLA: So I think that's just the resolution of the convolutional network that ran. It just output a low-resolution prediction. So there's other artifacts like that, too.

How about this one? This is Margaret Hamilton, who was working at MIT doing programming in the '60s for the Apollo project. So is this a real photo of her or a colorized photo of her, a real color photo or synthetic? So real-- raise your hand. A few more people-- fake.

Actually, this is split half and half. Good. It was fake, it turns out. It was a black and white photo. So who sees some issues that might give that away? Over here?

AUDIENCE: Were there [INAUDIBLE]?

PHILLIP ISOLA: There were. There were color pictures in the '60s. So it was only 1960s. Sorry. Way in the back over here?

AUDIENCE: There's a watermark in the bottom right that says "COLOR_BYANGELINA."

[LAUGHTER]

PHILLIP ISOLA: That I didn't notice. Good. So there's a watermark. Any others? And then way back?

AUDIENCE: The door has some weird dark coloring on it.

PHILLIP ISOLA: So it's a little hard to see from-- on this projector. But the door, and also the background here, is this mottled white-to-blue. And this is, again, that flipping around the probability thresholds. So it's not realizing that a white surface, like the paint on the wall, isn't-- is-- has a joint dependency. If part of the paint is white, the rest of the paint should probably be white, too.

There's one other issue that-- so this is the issue of that. But there's one other issue, which is this issue here. So that issue is also showing up in this problem. So notice that all the colors look a little desaturated. It's because we're basically doing some averaging that is saying for all the possible colors, we'll collapse the distribution toward the mean because we're not sure what to output.

So you would only know that the system is doing that if you know how that neural net is trained and what its objective function is. I happen to know that. But this is just one of the things you'll see all the time in AI-generated work, at least from a few years ago-- that things look a little bit blurry, a little bit desaturated.

And if you're in an audio domain, it might be everything looks-- the music that's being generated sounds a little bit fuzzy. It's not crisp. These are the effect of averaging that you get when you're fitting a distribution that doesn't have the capacity to approximate the true distribution and averages between all of the different possibilities in the true distribution.

So let's go back to 2012. This is when deep learning really took off, became popular. And I think that the story in 2012 was very much about deep learning as a new hypothesis space with high capacity.

So remember that in machine learning, we always have my objective function, the hypothesis space I search over, and how do I search over that hypothesis space, which is called optimization. So you would have seen this slide. Maybe some of you have seen it. I think it still shows up in various AI courses.

This was a very popular slide in 2012 or a little bit after from Andrew Ng. And he was pointing out that learning algorithms pre-deep learning-- in those days, it might have been a linear support vector machine.

Something that's linear is going to have less capacity. It's just a hypothesis space that can't fit nonlinear data. And that means that as I get more data, then I'll find the best linear fit. But then I won't ever be able to model the nonlinear parts of the data. And so deep learning will just keep scaling with more data, but other methods don't.

So this is an argument about the hypothesis space. But I want to make an argument about the objective function. So let's go to this problem now, which is going to be, assign a class per pixel in an image.

So recognize, is it the sky? Is it a bird? What is going on in this image? And we're just going to do the per-pixel classification. So let's see. So it works. But actually, let me go to this example. I think this will-- this is the better one.

So let's suppose that I am trying to take a map of the world and predict, what does the landscape look like at that location in the world? So this is a Google Map. And this is a satellite photo from Google Maps as well.

So I have paired data. I can just predict the y's from the x's. But I'm regressing or outputting an entire image. And now let's say that I use my unstructured prediction objective, which is to just minimize the squared error between my output. I can just take my output image, flatten it into a big vector, and just take the squared error between that vector and the ground truth vector.

So this is like least squares regression for outputting a high-dimensional object. And this is implicitly treating all of the output dimensions as independent. That's basically what this sum is doing.

So squared error-- remember, it's like log probability under a Gaussian probability model. Sum of logs is like the product of those probabilities. So this is implicitly a unstructured objective where I've assumed that the world factorizes independently over dimensions of the output space, independently over each pixel.

So I could do that. And then I could run my greatest neural network of 2012 on this data. I can make the biggest, highest capacity convolutional network I want. I can use SGD to optimize it. And here's the input. And here's the output.

So it looks OK. This is some buildings. And they're getting colored a nice, bright paint like a city might have. And the roads are asphalt. And they're gray. But it looks blurry. So why does it look blurry? Exactly the same reason again.

This park has trees in it. Now, I don't know if it's an oak tree or a pine tree. I don't know where the trees are. If I average over all the different trees and all the different grass and all the different locations, I'm just going to make an average green in the park.

So by not modeling the joint dependencies between this region and this region, I have to-- or basically, by not modeling the distribution over possibilities, I have to take the average over all possibilities. And that's a blurry image.

So that was deep learning 2012 to 2016. And then deep learning 2016 to 2024 has been about that, to some degree, but more about modifying now the objective function. And that's where the generative AI comes in. So we need an objective function, some way of penalizing my predictions, that can represent and model and penalize errors in structure, in high joint structure between pixels and the deviations from a multimodal posterior, both those kinds of structure

So let's look at how to model that structure with a GAN. And then we'll look at how to model that structure with a VAE. And I'm going to talk about image-to-image problems in this section-- so input image, output predicted image.

So first, we'll talk about GANs that are conditioned on an input image. So here is one way that you can do this. I'm switching the problem now to be-- let's say that I have an architectural layout diagram. Each different color in the left image there is going to be indicating what is the architectural element-- is it a door, is it a window, and so forth. And my output prediction is what that building would look like.

So I could be an architect trying to design new buildings and use this tool. I'll use a neural net. It could be a ConvNet in this project. But it could be a ViT in the modern era. And I will make my prediction.

So I told you about GANs. When I told you about GANs, I said that we're going to take a random latent variable that's Gaussian-distributed, pass it through a generator, output an image such that a discriminator network can't tell whether that image is a real image or a synthetic image generated by the generator.

Here, the noise is replaced with a conditioning variable. So there doesn't have to be noise. In this example, there is just a mapping from an x to a y as opposed to a z to an x. So it's the same exact math as the GAN. G tries to synthesize fake images that fool d. D tries to identify the fakes. They play this adversarial game. And at equilibrium, you're outputting images that are indistinguishable to the discriminator from real images.

So let's go through the math. I showed this very briefly in one of the previous lectures. But let's go through it again in the conditional setting. So here's the game from the perspective of the discriminator network. What it's going to try to do-- it's going to try to assign-- it's a binary cross-entropy objective.

So that means it's going to try to assign high log probability to-- this is going to be like a fakeness classifier, or it is a synthetic data classifier-- will try to classify fake data as synthetic. So it'll try to assign high log probability to data which is output by the generator. That's the definition. Synthetic data is data output by the generator. And it'll try to assign low log probability, 1 minus the probability, to real data, y.

So the training data, y, is real. The inputs are x. The outputs are y. That's the real data. I'm trying to distinguish between data that's created by g and data that's created by the true world, y. So this should be assigned high probability. It's just a binary classifier using cross-entropy objective.

So that's the job of d, just classify synthetic or real. The job of g is to try to make synthetic images that look real according to d, that fool d. So d is trying to maximize the log probability that it places on the true data under this cross-entropy objective. And g is trying to do exactly the opposite, minimize the log probability that-- of a synthetic image being called synthetic. So it's basically trying to say, I want my synthetic data to cause d to be wrong-- so minimizing as opposed to maximizing.

So then we will backprop from d through the parameters of g, update g to make a more realistic photo. Now it becomes a real-looking photo. It doesn't have as many artifacts.

So the GAN objective is this minimax game, this adversarial game where I'm trying to find the parameters of the data generator that will create outputs that a discriminator cannot tell whether they're real or fake. The discriminator will get a score of 0 or-- as low a score as it possibly can.

And we don't want to fool just any old random discriminator. We want to fool the best possible discriminator. So the discriminator is learning to get better and better at detecting the fakes. And the generator is getting better and better at fooling the discriminator.

And at equilibrium, you can prove that this will converge to being-- samples from g are identically distributed as samples from y, the true data-generating process. So that's provable under the condition of the network has actually found the equilibrium-- the Nash equilibrium of this game and that there's sufficient capacity, and so forth. But the Nash equilibrium has that characteristic.

So the perspective that I want to give in this lecture is that from g's perspective, d is like a loss function. It is some kind of penalty. It's a critic or a discriminator. It's a penalty that looks at the outputs and says, are they-- how good are they or how bad are they? And then you can backprop and cause g to update to make better outputs.

So it's like a loss function. It's like the squared error loss function, except it has a new property, which is that it's learned as opposed to being hand-defined. So it's a loss function that can look at structure in the output and not just say, did I get every pixel independently correct, but also, did I get the edges correct? And did I get the higher order statistics of the output correct? And this is what makes it powerful. And it's very hard to hand-define such a loss function. But it can be learned via this game.

So you might have noticed that there's one little error with this formulation. So I gave you the GAN objective. But for a conditional GAN, there's actually a problem. So if I output this photo, should d in this setting call that a real photo or a fake, synthetic-- is it real, basically?

So it is a real photo. I'll just tell you that. This is a real photo. But it doesn't match the input. So the one little trick to make a GAN conditional is you need to let the discriminator look at not only is the output real, but does the output match the input? So it's, is the pair x, y a real pair sampled from the data-generating process P data x, y?

D has to look at both the input and the output. Now it can penalize whether the output is blurry and unrealistic-looking. But it can also penalize whether the output has the windows in the wrong place that don't match where the architect has drawn the windows here.

So that's a very simple modification to the objective. D just has to be conditioned on-- it has to take as input the data x. And it's asking, did g of x create a realistic output that looks like y? And also, did g of x conform to the input so that the relationship between the input and the output matches the true data distribution? Question?

AUDIENCE: What is the definition of x for the ground truth label? Are you assuming you have some way of generating one of those diagrams for [INAUDIBLE]?

PHILLIP ISOLA: So what is the definition of x? X is given. So these diagrams-- you have a data set of diagrams with photos. So you have paired data. You have supervised data, x, y pairs. Yeah, question?

AUDIENCE: Conceptually, at what point are we doing the loss function? Is it like we generate the whole image and then we see that, oh, it's real or fake, or at what point is this game of adversarial happening?

PHILLIP ISOLA: So at what point is this-- are we applying the loss function? So the key idea is I'm not trying to say real or fake photo. I'm trying to say real or fake pair of photos. So what you actually do when you run this is you take x. You run it through your neural network.

It outputs a prediction. And then you run that prediction through another neural network, the discriminator. And that scores your prediction. And then you backprop through that entire computation graph. So the computation graph is d of g of x.

This slide is just saying that I can also add a regular old regression objective, which is just-- this is what regular regression would do-- would say, is the output close to the ground truth label y, or the ground truth image y, in a squared error sense? And that will produce these blurry outputs because it doesn't-- it treats every pixel as independent, conditioned on the input.

But it can be added to this GAN objective for a little bit better performance because the big issue with GANs-- they're not as popular now. And the main reason was they weren't very stable. They didn't converge very well. This adversarial game is very hard to optimize. But this regular regression loss can very much stabilize that process.

So that works. But this is not impressive. By now, we've seen that you can generate all kinds of photos conditioned on all kinds of things. But a few years ago, it was magic. But anyway, this works.

So I want to mention one of the interesting considerations in using discriminators to penalize structure in your data, to penalize errors in the structure of your outputs. And that is that the architecture of the discriminator critically matters in what types of errors it is sensitive to and what types of errors it's not sensitive to.

So one of the things that worked really well in this project that made these images look good for that era was that we used this discriminator architecture called a patch discriminator. What that means is that rather than just looking at-- having a neural net that looks at the entire photo at once and says, is it real or fake, we look at every patch of the photo and say whether that patch looks like a real patch or a fake patch.

So this means that discriminator is going to slide-- it's going to be convolutional. So it'll just slide across the input image. And for every patch, it will be making a prediction. And this gives you-- you can use a very small network to do that.

So that's one advantage. And additionally, you have more supervision because you have a-- your discriminator's being trained on a data set of patches. And there's much-- many more patches than there are images. So if I have 10 images, I might have 10 million patches. And that's just more data.

So one question that you can ask-- but this is-- that was just the architecture of that time. One fun note is I think that this architecture is still part of Stable Diffusion to this day. But maybe in the very latest Stable Diffusions, it disappeared. But it was for quite a few years. So if you have ever played with Stable Diffusion, you had the patch GAN discriminator in there.

But anyway, this architecture could change. But the important point is that this architecture defines the properties of the loss and how it looks at structure. So what happens if I make the receptive field, the patch size that the discriminator is looking at, really, really small?

So if I make it really, really small, just 1 pixel by 1 pixel, I get back to almost the same thing as least squares regression that-- because I'm letting my discriminator only look at one pixel at a time to decide whether it's real or fake. And then it looks at the next pixel and decides if it's real or fake. It just takes a sum over all the errors. And that is, essentially, factorizing my model into a set of independent predictions that are not jointly penalized.

If I make the patch 16-by-16, now I can say, look, each of these predictions might have been pretty good, but it's blurry. Blurriness cannot be seen in a single pixel. You can only see blurriness with two pixels, two measurements. You say that it's not sharp enough. The difference between them is not big enough. It's the second-order statistic of the data.

And if I make it even bigger, I can model higher and higher order statistics, statistics that depend on the joint configuration of many, many measurements, 70-by-70 measurements. And then in this era, if I-- if we make the patch size too big, the whole image size, it worked worse because-- essentially for the reason that the model-- discriminator had too many parameters relative to the number of data points. But in the more modern era, this might not be such an issue because we just have infinite data. But back then, we only had finite data. This was trained on 400 images.

So the story in 2012 was that deep learning is a high-capacity hypothesis space that can-- that many, many functions are in the hypothesis space. And older algorithms had low capacity. So they just could not find complicated nonlinear functions.

What I want to point out with the last few slides is that deep learning with unstructured objectives actually does saturate. So it could be that, yes, in the hypothesis space, there's the correct answer. It's a universal approximator. I just make it big enough, wide enough, deep enough.

But if I'm only incentivizing with my objective function that it's good enough just to model the factorized distribution, treating every different dimension of the output as independent-- if that's considered good enough, there'll be no incentive to model the joint dependencies. And you won't get a good result. But with objective functions, and all generative models have this objective function that can model joint dependencies in the output, you get better performance.

This is a cartoon. But these little swaps of the ordering are meant to be indicative of the fact that with more capacity, you might require more data. So less capacity and a simpler set of modeling assumptions might mean that you can use less data to get a good result. But if you have more data, then you'll get better results if you have more capacity and a more flexible model class.

So here's where we were with just unstructured prediction. Each pixel is independent, penalized with a least-- an L1 loss per pixel. And here's where we are with the patch GAN discriminator and the conditional GAN, where now it says, it was blurry. That looked unrealistic. That's obviously fake. I better add a bunch of trees.

But we enter into the regime of hallucination, where these trees are just a guess. They're just a sample from the true distribution over where the trees might be. So it's not necessarily those trees are actually at that location. It's just this is one of the modes of the distribution that I'm modeling. Questions? Yeah?

AUDIENCE: I don't know if you know this. But what would happen if you did an L2 loss that's like a Gaussian, but with covariance, where you enforce some structure in the data?

PHILLIP ISOLA: Covariance over the full joint-- just the full vector?

AUDIENCE: Two pixels are close or pressed together, and that somehow--

PHILLIP ISOLA: Like some kind of Gaussian process over the entire set of measurements, something like that, or just one big Gaussian? I think it will-- it'll improve things over this result. But I would be pretty surprised if it makes a dramatic difference. I think that that still can only model Gaussian structure. And there might be other structures that you're missing.

AUDIENCE: So the L1 prediction seems to very faithfully represent the map. But the-- looking carefully at the GAN representation seems to have eliminated its-- some of the details that are [INAUDIBLE]. So is this a common thing?

PHILLIP ISOLA: So the question is, the L1 reconstruction is very faithful. And the GAN-- actually, the-- some of the paths might not quite match, like maybe this road disappeared here. So this is a common trade-off-- that the L1 prediction is taking the average. It's very conservative. It's very robust in that particular sense that we can be pretty sure that there's going to be some road here.

And all the modes of the distribution must share that so that the average will also share that road, whereas, again, we'll select a mode. And the mode should share it. But essentially, maybe this just-- this mode is a little deviated from-- I think it's kind of like an ensemble. An ensemble might be a more robust prediction of the truth than one of the modes that might be a little low probability. And maybe this is low probability in that region.

So the rough idea is that the problem with generative AI stuff is that you often will get these hallucinations that are samples from how something might be, but not all aspects of that sample might actually be high-probability. And I think that's what you're seeing here, roughly.

So let's go on to conditional VAEs. So every generative model that we've talked about can be made conditional. Oftentimes, it's quite easy to make it conditional. So let's see how to make a VAE conditional for the same problem.

So one of the issues with the GAN formulation is that this generator function tends to exhibit what's called mode collapse, which is that, yes, it will output one mode of the distribution that looks realistic. But it won't-- it will maybe miss other modes of the distribution. It's not great at actually capturing the full posterior, just maybe at one or a few modes of the posterior.

So conditional VAEs are a method that is a little better at maybe capturing all the modes of the posterior. And here's how they work. So what I meant to show here is that this schematic could match that building. But it also could match a brick building.

I have no idea what the material is. It's not told to me in this schematic, just where the windows and doors are. So what I'll do is I'll simply use this VAE trick where I'm going to have some latent variable z that will get injected into the network in order to model all of the uncertainty over the possible variations in the predictions.

So let me go to this very simple example. We have a pink square here. And it's moving up and to the right over time. So you can see that as I go to the right, the square is actually getting a little bit closer to the corner. And if I do-- if I try to predict the future given the past-- so I'm going to take this frame of a video. This is a moving object. Where will it go next?

If I do that with the least squares objective, then I am going to get averaging. So there's eight possible directions this square could go in this little demo. So here are the eight possible directions it could go. And my prediction will be an average of all eight possible directions, average of the eight modes.

So we want to somehow be able to model each of the modes and sample each of the modes and not just sample the average. The average has zero probability under the data distribution. So that's a very bad prediction.

So we're going to build up something that looks like a VAE. So first, we have what looks like an autoencoder or an-- the VAEs we saw last class. We just are going to take my target. This is the prediction of the future. We're going to try to compress it into a bottleneck, a Gaussian variable z, and then reconstruct that prediction. So I'm trying to predict y given x. This is x. And this is y. Here, I'm just running a VAE on y.

Now, to make it a conditional VAE, I'm going to simply let my reconstruction network, my decoder, observe the data x. What it's going to try to do is say, I want to predict y, the future, from x. But I'm also going to tell you the answer. But I'm going to force the answer, y, to go through an information bottleneck, z.

So what's going to happen is this arrow here is going to say, well, I don't know what direction it will go. That information has to come from z. So z will learn to encode what direction this thing goes.

But z is compressed. Z is trying to be a unit Gaussian. So it's trying to remove information. And so rather than encoding the color in z, the color can be given by x. And there's higher information capacity in this arrow.

So what will happen is this is trying to learn as little as possible in order to make the correct prediction. So I don't need to encode the color, since that's given in my observation. So z ends up only encoding the direction of motion, which is unobserved in the observation.

So if z sees this target, it will compress that into, oh, it-- the box should go down. So z learns to encode whatever information is missing from the observation in order to make a correct prediction.

Now, at sampling time, I can sample from this model by replacing my encoding-- my inference of z given a target image with z as a unit Gaussian because I've trained it as a VAE. And I've imposed that it should be close to a unit Gaussian.

So I can just sample from z. And now I'll sample from z. Maybe I sample the down z vector. That will cause me to predict that the object goes down. So I'm sampling all the possible ways it can move by sampling from z. Now I sample up. It goes up. It goes right.

So you can train this for all kinds of data. On this example, I'm introducing these latent variables that specify everything that's unspecified by the input x. So z learns to model everything that's necessary in order to pin down what the output should look like that's not explicitly given in x, all the uncertainty.

So P of y given x is uncertain. So I'm going to add a latent variable z, just like we did with VAEs, where we're trying to learn all-- in unconditional VAEs, z has to encode everything about the data. In conditional VAEs, z only encodes information that's not given in the conditioning variable, x.

So one concept to give you here is now we really have two different ways of controlling the outputs of a generative model. In the previous lectures, we just had the latent variables z. And there were knobs that specify all the attributes of the data. And those knobs are here, controlling the lighting and the materials.

But now we also have conditioning variables, x. And these also can control the way that the data looks. And for a lot of modern generative models, actually, the latent variables might play a very small role because you're just conditioning on more and more data.

So in text-to-image models, if you tell it a whole string of text, now you don't need a lot of latent variables because the string almost-- very precisely could pin down what you're going to see. So let's look at text models next.

So first, we'll talk about text-to-text. Luckily, text-to-text is basically nothing new because text models, language models, are typically autoregressive. Let's look at the autoregressive class of language models.

Autoregressive models are already, by construction, conditional generative models because an autoregressive model-- if you recall, we model the data as a sequence of conditional predictions. The probability is a product over all of the conditionals P of x2 given x1, P of x3 given x1 and x2, and so forth. So autoregressive models are natively conditional. And you don't have to modify them or do something new to turn them into conditional models.

So let's look at this with a conditional language prediction problem. So conditional prediction is take text as input, predict some other text as output So the way you train it is the way we've always described training these things. You just do next-word prediction.

"Colorless green ideas sleep"-- what's the next character or next word? "Furiously." And now I can use this to solve conditional prediction problems by simply giving the language model whatever input I want. And that's the conditioning variable, x.

The stochasticity of the latent variables of this model-- there's not explicit latent variables. But it's stochastic because every next word I predict will be sampled from a categorical distribution over the next word if I'm doing a autoregressive model where every conditional density is a categorical distribution.

So it's still stochastic. But these conditioning variables are now made explicit in the training process. And you can do all kinds of things. And this is often called prompting language models, which I'm sure most of you have played with. This is what you do when you just talk with your favorite LLM, Claude or ChatGPT.

So let's say I want to turn my language model into a classifier of the sentiment of a review. I have a product review. I want to now predict conditionally, conditioned on the review, what-- did the user like it or not?

So here's how you do it with an autoregressive language model. You simply take the review. You concatenate, you prefix that onto the input to your question. So that's like conditioning on x the review.

And then you add whatever question you want, like this-- what's the sentiment in this review, or the sentiment in this review is, and then the next-word prediction should be your predicted y. So it's P of y given x, where x is the review and y is the sentiment-- so really easy way to do all kinds of conditional predictions with language models.

This became evident, I would say, in the GPT-3 era. Now this is-- people use language models this way all the time. But it was-- it took a while for folks to realize that language models are not just about writing paragraphs from scratch, like writing funny stories from scratch. They're actually most powerful for conditional prediction. Given an instruction or a prompt or a question, they can answer that question for you. It's a conditional prediction problem. And that's where they really became powerful.

So here, I gave Chat-- I gave GPT-3 this review. And then it says the next character-- or the next word should be "negative." And that's a prediction of the sentiment in the review. Question, yeah?

AUDIENCE: Is there a reason "five" has underscores [INAUDIBLE] it?

PHILLIP ISOLA: No. I think that just people put italics with two underscores. Just, it was how that review was written. I don't know. I think I probably made this one up myself. So I was just indicating italics. It doesn't really matter. Yep?

AUDIENCE: Is this similar to retrieval-augmented generation?

PHILLIP ISOLA: So retrieval-augmented generation would-- is related to this. But I would say for now, treat that as a separate concept. There's no explicit retrieval in what I've shown so far. Yeah?

AUDIENCE: In this case, for training, do I just generate a bunch of instructions and try to make the GPT answer to these instructions?

PHILLIP ISOLA: How do you train this thing? So there's two ways. So one is that you just-- you're modeling all of the words that any human has ever spoken or all the words on the internet or some huge corpus of sentences. And probability of sentiment given review might be somewhere in that space of all possible things people have ever talked about.

When I'm just training a language model, it will learn all these different conditional distributions if they exist in the data. So this is one conditional that exists in the data, the conditional of predicting the next sentiment given the previous paragraph.

But now if your conditional distribution you want to model is not just implicit in the data, then you can collect your own data set of reviews and sentiment labels. And then you could fine-tune on that.

Cool. So we'll have another lecture on language models where we go a little bit more into all the different ways you can prompt and tune and adapt these things. But basically, autoregressive models are just conditional by default.

So now let's look at an autoregressive model for image-to-text. So again, it should be pretty simple. But we need to make sure we can map between two different data types. And the way we do that is we use transformers, which are a way of putting all data types onto the same equal footing. Just turn everything into tokens, and then just do token-to-token mappings.

So I'm showing images. I'm showing text. But whatever data format you want to work with-- molecules, financial data, spreadsheets-- if you just can convert it to tokens, little bundles of information with an embedding, a token vector, then you'll be able to work-- you'll be able to run all the same technology that I'm showing you here.

So here's an architecture for how to map from an image to a sentence. Sometimes, this problem is called image captioning if I'm trying to take an image and train the system to output the caption that somebody would have written underneath the image online, on websites, or maybe the alt text of the image.

So here's how to do that with a transformer. I have my branch down here, which is the image encoder. It will take an image. And we talked about this in the transformers lecture-- that typically, we use these vision transformers, which chop up the image into a set of patches, embed each patch into a vector, and then just run a transformer over those tokens now and get an output set of tokens. So now I've tokenized the input into these nice, abstracted representations of the-- of what's going on in that image.

So now this part here is just a GPT regular-- GPT is one name for it. But it's just an autoregressive language model with a transformer. So what is that doing? It's taking in the prefix to a sentence and predicting the next word, or predicting the next-- with causal attention, for every prefix, I have a causal dependency between that and the next token in the sequence.

So this word here, "sitting," only depends on all the previous stuff. "Bird" does not depend on "sitting." It only depends on the previous stuff. That's the causal attention trick. And I just train with this next-token prediction objective with self-attention, but causal masked.

Now, to stick them together, all I have to do is-- they're all just in token space now. So all I have to do is prefix the set of language tokens with the set of image tokens. And now I will be basically reading off a-- given this token, this token, this token of vision, what is going to be the next word after this word, this word, this word?

And one of the very simple ways of stitching these networks together is what's called cross-attention, which is just to say, this GPT language model is going to simply be prefixed with additional tokens that can submit keys and values that are queried by these tokens. So the query is emitted from the word "yellow." That will then match the key for the bird because it's yellow.

Potentially, that would be a good strategy. That will submit a value that gets integrated into the token representation at this layer of the network, which is now like a multi-modal representation of both the words and the images.

So cross-attention is just attention, but going-- taking keys from-- and values from one set and queries from a different set. Any questions about that mechanism, a really easy way to stitch things together? Yeah?

AUDIENCE: Question on the text-- a yellow bird-- are they-- at what point are they provided?

PHILLIP ISOLA: So this would be, if I have training data of the form x, y-- we're still in that setting-- the x is the image, and the y is the caption. And I'm always going to be extracting tokens from the image. And then I will be autoregressively predicting the next word, given the words I've so far predicted in the caption. Does that answer your question?

AUDIENCE: So at the very beginning, there will be no text?

PHILLIP ISOLA: Yeah. This is, like, three steps of autoregression in. But you train it all-- you-- at inference time, you do it sequentially. But you do it in parallel at training time. This is like training time. Yeah, question?

AUDIENCE: At the cross-attention layer, if you use a-- say if you use the embeddings of the text as a query instead of just the token-- so a learned representation-- would that make the learning quicker, or it doesn't matter? Instead of using the tokens of the text, you use them and learn embedding another model, and you use cross-attention to learn embeddings of the text and learn the image versus just the tokens that were not used. Does that accelerate the learning?

PHILLIP ISOLA: So I'm not sure I'm fully following. Maybe we can talk about it offline. But there's a lot of different strategies for how to integrate this information. Do I learn the-- typically, you have a query matrix which applies to the text and another query matrix that applies to the visual data. And then you can submit keys, values, and queries from both modalities.

So one of the trending industrial things this year is multimodal chatbots. So now all of the language models actually are not just language models. They take images as inputs. Some of them even take videos as input.

This is ChatGPT. And here's the slide. I just take a screenshot. I think some of you pointed out to me that you can interpret our slides better by just doing this. And in fact, it actually works decently well. So go ahead and do it if you want.

So what is this slide showing? And how does it relate to how I work ChatGPT? And it's pretty good. You have an image encoder, cross-attention layer. But it did get this wrong. So how does it relate to how I work?

It says, well, I'm not a language-- I'm only a language model. I don't process visual data. And unless OpenAI is doing some crazy hackery-- it was looking at the visual data. It had to be in order to make the correct assessment. So I think this is wrong. It doesn't know how it works. It doesn't know how its own brain or itself works.

So let's go in the other direction now and look at text-to-image. And the interesting thing here is that text-to-image models of the current generation, or maybe one generation back, actually are a combination of GANs, VAEs, and diffusion models. They're all three. So it's a nice-- people call them diffusion models. But they have all three components in them. So I'll tell you about that.

So here's the more general strategy for translating between two different modalities or two domains. It's just a general conditional prediction of one type of data from another type of data. So now we have our data distribution. That goes from encoder to this latent distribution.

And then if it's a-- if it's in the VAE autoencoder style, then I decode from that latent space now not back to in autoencoding, like back to x, but back to another domain, y. And I penalize the difference between my prediction and my-- the ground truth, y.

So this is just a general way to think about it. So here's what that looks like in text-to-image, which we talked about before. So I go from image space to latent space to text space. And that would map onto this, image space to latent space, and then also text space.

Where do you define the latents is not clear because it's autoregressive here. That's not explicit latents. But this would be more like the VAE way of doing it. But now let's look at the text-to-image problem. So go from text, "A yellow bird sitting on a branch," to the image. And this one is going to follow this diagram a little bit more precisely.

So I already told you about how to train a conditional VAE. So how do you train a conditional VAE when you're conditioning on text? So rather than conditioning on the observation of that moving pink block, I'm going to condition on a scarlet macaw sitting on a perch.

So what's going to happen is the decoder is going to be informed by the observation as well as my latent variable z, that explains everything-- that has to explain everything about the data to make this a deterministic process. So the decoder is deterministic. So this latent variable has to model all the random factors that could affect the output appearance given the input.

So this input does not say that the scarlet macaw is facing to the left. So therefore, the latent variable has to learn to encode that the macaw is facing to the left. The latent variable is trained by inferring the latent variable from the target prediction.

So this is like a VAE, and then conditioning. It's a conditional VAE-- so same idea that I showed on the previous slides. And now the only new thing is I need this to be a language model, f, that encodes text into tokens. And I need this to be a model that could be a ViT. A transformer decoder takes a set of tokens. And the latent variables might also be tokens.

And at inference time, I simply sample from the unit Gaussian because the VAE is trained to make the latent distribution dense and compact. And therefore, samples from the Gaussian are going to all map to valid predictions that are all in distribution with respect to how the model was trained. Yeah, question?

AUDIENCE: If you go back to the previous slide, what's the optimal strategy to incorporate this information? Is it just, for example, to sum the two embedding vectors or, for example, to concatenate and then process through an MLP to have something in low-dimension space?

PHILLIP ISOLA: The question is, what is the optimal strategy for how to combine this information with this information, I believe? And I would say the vanilla, most standard thing right now seems to be concatenation, like convert everything into a set of tokens or a set of embedding vectors, and then just concatenate and do more cross-attention between them, and then repeat.

So what I showed on this slide is the current standard way of combining between different streams of data. Now, it's not the only way. You could also sum them together, take an average, multiply. There's all kinds of other ways. But I would say this is the current one.

And how do you combine different streams in a transformer in a more intelligent way? That is an active area of research-- so maybe a good final project, compare different ways of combining these things.

So that's how a conditional VAE can solve language-to-image. And this is roughly the way that the Dall-E 1 model works. So the Dall-E 1 was probably the first text-to-image model that made a splash. But then Stable Diffusion, Dall-E 2, Midjourney, all these other things came out. And they switched to using a diffusion model. So how do I make a diffusion model do text/image generation?

So it turns out with diffusion models, it's really, really simple. So here's my diffusion model. I start with random noise, unit Gaussian noise. I try to predict step by step. And I've supervised it with the opposite direction. My data point is here. I add noise. And that's my supervision for how to denoise. I try to undo the noise I added.

So each step is a supervised learning problem of, here's the target, the less noisy-- here's my generated noisy image. I try to invert that by predicting what the less noisy image would look like, given the more noisy image. And to make it conditional on text, all I have to do is let my denoiser observe the text.

Why does that work? That works for the following reason-- because the text will be informative toward what the true image I started with is. So if the text says "A robin on a yellow background," well, here's a noisy image. I don't know.

It looks like it might be a little bit of a yellow background, or maybe it's a white background or a pink background. The text disambiguates that if the text is a-- is actually-- if there is a dependency in the training data between the text and the image, which there should be.

So to the degree to which there is that dependency, the text will help you make a more correct prediction. Therefore, f will learn to use the text and learn to denoise in the direction guided by the text. If the text were "A robin on a pink background," that would be a hint as to what the correct answer is. And therefore, my denoiser would head more toward pink.

So the text will guide how the denoiser maps to different attributes of the data to the degree to which the text actually does describe the image content. Does that make sense? So it's really easy to make a diffusion model into a conditional diffusion model. Just let the denoiser take text as input.

It's really easy to make a GAN into a conditional GAN. Just let the discriminator take the text as the input. It's really easy to make an autoregressive model into a conditional model. It already is a conditional model. The VAE was the only one that was a little bit tricky.

Here's how Stable Diffusion from a few years ago worked. Stable Diffusion has five or six different versions. So I don't know which version this is. But this was one of the versions. So what it did is at every step of denoising, the denoising network was a neural net called a U-Net. It's just like a ConvNet with these skip connections.

So it was a convolutional architecture. Now people are-- use what's called diffusion transformers, where this is a ViT. This is a transformer model. But back a few years ago, this was a-- called a U-Net. And the text information comes into the network now not by just concatenating tokens, because this was not like a token architecture. This was a architecture before transformers.

And so you actually took the vectors coming out of the text encoder. And I think you modulated the batch normalization or something. There's ways of integrating-- scaling each of the activations by, somehow, the outcoming vectors here. So you could look up the details. But I don't know if it matters that much. But there's a lot of interesting ways of integrating information from one data stream to another data stream.

And of course, once you have this ability to plug together different modalities and solve general conditional prediction problems, it's-- this is a very general thing. I think that's clear from ChatGPT and language models, where you can use them for so many purposes.

But here's just an example for what's called the Visual Question Answering problem, where I have a text encoder that takes a question as input, turns it into a set of tokens. I have an image encoder. That turns it into a set of tokens across a tent. And I have a decoder that tries to predict the answer to the question.

And if this is trained on tuples of the form question, image, answer, then it will learn to maximize the conditional probability of the answer, given the question and the image. And that's general-purpose image understanding. Whatever question you want to ask can be framed in this way.

So this is a very general-purpose technology. And just to show you that I'm not-- I don't think I'm hiding too much under the hood here, this is one of the current popular image-to-text-- or this is one of the current popular VQA models-- so image question answering, or visual question answering. It's called LLaVa.

So here's my image encoder, a set of tokens. Here's my question encoder, or they call it instruction because it could be an instruction, not just a question. That's a set of tokens. They cross-attend. And they out-- threw a transformer, a GPT, and the output of language response. That's pretty simple.

And I said that in the text-to-image direction, actually, you're combining a lot of tricks. All these models are very related. And often, the tricks are actually being shuffled between them. So this was one of the important text-to-image models from a few years ago. This is still the core of the Stable Diffusion line of work, at least up until they might now be doing things with these flow-based models-- but just up to 2023, let's say.

So you actually first trained what's called a VQGAN. That's a VAE with a patch GAN discriminator that tries to make the output look sharp. So it's like, I take-- I encode. I decode. I put that through a discriminator.

So it's like a combination of a VAE and a GAN. And then in the latent space, I have either a diffusion model, or here it's actually an autoregressive model over the latent variables. So you can just combine all of these things together. And that sometimes gives you better results. Question? Yeah?

AUDIENCE: How do you creates these GANs? It sounds like a [INAUDIBLE] separately, all together, or [INAUDIBLE]?

PHILLIP ISOLA: Great question. How do you train such a system? It looks complicated-- backprop work. So you can potentially backprop through the whole system. That's one approach. You can also pre-train subcomponents separately if you have data for-- if you have some data which is text paired with images and other data that's just raw, unlabeled images, raw images without text, you might pre-train the image-only part of the system. You might pre-train this VQGAN with just the images and then backprop through a interface with a language model only on the paired data.

So there's a lot of ways of doing this. I'm not going to go into this detail. But there is what's called vector quantization in the latent space in this particular model, which makes backprop hard. And so you have to deal with this vector quantization. But I think-- I won't go into that today. Yeah, question?

AUDIENCE: We have seen example of generating an image or a text. But what if-- for example, on ChatGPT, we have this interaction of, given a text, generate image and text. Do they go through one model, or actually it's an engineering trick that [INAUDIBLE] give you two models?

PHILLIP ISOLA: Good question. So in modern chatbots, usually, they take image and text as input and output image and text. And now some of them take audio as input, too. And they'll output audio. Eventually, they'll take all modalities as input and output all modalities that we can think of.

I don't know how they work because these are all now companies that don't tell us how they work. There's one-- Llama 3.2, I believe-- which takes at least images and text, and maybe audio as well. And that does have a paper that describes the details. So I would look at that if you're interested. So I don't remember how that one integrates information.

So any more questions on conditional generative models for learning from paired data x, y? Yeah?

AUDIENCE: As you increase the modalities, it seems that they probably get increasingly data-hungry.

PHILLIP ISOLA: As you increase the modalities, you would get increasingly data-hungry?

AUDIENCE: It seems like that. Is that a problem we're running into?

PHILLIP ISOLA: That's a really interesting question. I think it can go in either direction, in some sense. So as you get more modalities, maybe there's a more complicated-- maybe the mapping from x to y-- if I'm trying to map from input image to both sound and text, maybe that's a function that just is more wiggly, is more complicated function to learn. So I need more data to learn it.

That's one possibility. But it's also possible that there are-- there's shared information between modalities. So if I have more than one modality, then they can-- a pure language model that is just trained on language might be worse at language modeling than a language model that's trained on language and videos because there's shared information in the video about language information. So that's a sense in which you get an advantage, a statistical advantage, by modeling more modalities.

So in this last little section, I want to say, wait a second. So far, all of this lecture is about P Y given X for arbitrary high-dimensional X and Y. And I've said that you can-- but the thing that I haven't really explicitly said, but has been the underlying assumption, is that you have data that comes paired. So my P data distribution is over x, y pairs. And I can sample from that, get pairs, and then model the conditionals.

So what happens if I actually just have data x and data y, but I have no examples of x paired with y? Can I do anything? And it turns out you can. And oftentimes, you can do conditional modeling with unpaired data. And this is sometimes called unpaired translation. So can I translate this cartoon into this more nice Valentine? This is from 1968, this cartoon.

So here's the setting. So paired data for a colorization problem-- I have black and white and color. That's really easy to obtain. I just take a color image. I remove the color channels. I get black and white. It's paired data. I can model the conditional distribution of color given black and white using the techniques that we saw.

But what if I want to predict, what would Cézanne have painted if he had been standing at this river? I can't ask Cézanne to go and paint a million different rivers. And I don't even know. There's no photograph of this house that he was-- he painted here.

So I don't have paired data. And this is a potentially more common situation than the situations in which we have paired data. Having paired data-- it's a big constraint. Not only do I have to have both the x's and the y's, I also have to have the mapping between them, or the pairing between them.

Unpaired data-- it comes across all the time. Here's a simple example. I have a set of Cézanne photos-- a set of Cézanne paintings, a set of photos. And as a human, you can reason about the stylistic differences between those. And you can imagine what Cézanne would have painted for each of these photos.

How did you do that, despite never having seen a side-by-side photo of a place and what Cézanne painted? This is the problem of unpaired learning. And there's a number of different methods. So I'm going to talk about one in a little detail called-- doing this with GANs.

So here's a really cool thing about GANs and why they're still a neat model, even though they're less flashy these days. So with a GAN, the pairing between the input and the output is only coming through the discriminator, which looks at, does the output go with the input? But I could drop that. Oh, sorry. The animation is a little off. I could drop-- I could-- so let's see.

So the problem with not having paired data is I don't know what the Cézanne painting of this location would be. But let me go back and just drop that dependency. The animations are a little off. But let me drop that dependency where the discriminator looks at the pair and say, what if the discriminator actually only looks at the output and asks, does this look like a Cézanne painting or not?

Now this is the traditional unconditional GAN objective. I'm just trying to look at the outputs, my synthetic imagery generated from z or x. And I'm just trying to say, does it look like a real Cézanne painting, or does it look like a synthetic Cézanne painting made by the generator? And this is a cool property of the GAN loss. It's actually just checking whether the output belongs to the set of good answers, not checking whether the output matches the input.

So it's suitable for unpaired learning. But how do I actually make the output not just any random Cézanne photo, but actually match the input? Sorry. So here is what looks like a Cézanne painting that matches the input. But here is a-- well, the animations are off here. But here's also a perfectly valid Cézanne painting. And the discriminator will say that's real, too.

So the trick that works in this case is to use this idea of cycle consistency. So I'm going to impose the constraint that the mapping from X to Y is consistent with a backwards mapping from Y to X. So there's some kind of bijectivity between these two domains.

So I'm going to say I'll train a network to map from photos to Cézanne on paintings according to a discriminator that says, does it look like a Cézanne painting? Did I get to that domain output?

And then I'll train a discriminator in the opposite direction to go from Cézanne paintings back to photos. And that should fool a discriminator that will ask, does it look like a photo? So the discriminators will make sure I map to the domain of Cézanne paintings and the domain of photos.

But I'll train these things to be what's called cycle-consistent, which just means that if I go from domain X to domain Y and then I go back, I should arrive where I started. And if I go from domain Y to domain X and I go back, I should arrive where I started.

So this is kind of like-- in language, people use this, too. Actually, there's an old Mark Twain story about trying to translate between a French speaker and an English speaker, where you would translate into French and then back translates into English and check that-- and you know that the system is consistent with itself if you arrive where you started. If I say an English word, then I say, this is the French word, and then somebody that speaks French tries to translate that French word back into English, if I arrive back where I started, that's a property that should be satisfied by natural languages because natural languages tend to have a bijection between the different languages, like every French word has a roughly direct translation to an English word-- not perfectly.

So sorry, French speakers, if maybe there's concepts you can only express in French-- but roughly. So that actually works decently well. And that leads to results that look like this, where I've converted this photo into a synthetic Cézanne painting without having paired data.

So one question you might have is, wait a second. There must be many cycle-consistent mappings between any set of images x and any set of images y. And indeed, that's true. But there is some theory around this that's been developed to say that, well, under the constraints of a particular neural net architecture in a particular optimizer, only some of the possible mappings from inputs to outputs are going to be found.

So as long as I have some constraints that say that I need to find a simple mapping, in a certain sense-- and neural networks do find simple mappings. We've talked about that. So here's a ReLU network with a two-dimensional hidden state. It's trying to map this line segment A to this line segment B.

And there's infinite possible mappings. But a ReLU network trained with gradient descent with two hidden units can only find two mappings in this case, will converge on two mappings. One is the identity mapping. And the other is the flipping of the lines, like negative identity.

So why is that? There are also complicated wiggly mappings where every single point on A maps to a different random point on B. But it's very hard for those to be found by the architectures that we use and the optimizers that we use.

So under the constraint of having a simple mapping, it will tend to be the case that all map the green stuff to green stuff. That will be an easier to learn mapping. And that disambiguates that it should be green to green instead of green to blue.

And these ideas show up also in unpaired language translation. And potentially in other domains, you could apply them, too, where we try to impose what they call back translation or cycle consistency between the mapping from one language to the other and back.

And there's a number of other tricks that you can apply, as well, in order to solve unpaired translations. So I'm not going to have time to go into those other tricks. But the general idea is if I have a distribution of data in one modality indicated-- this is like the data distribution of the red modality. This is the data distribution of the blue modality. Well, I can just align their shapes. It's like a rigid transformation problem.

So under this idea of distributional-- the two distributions should somehow be the same. And I can align those two distributions, or they should be the same up to an isometric transformation. So if you're interested in other approaches to unpaired translation, look at this paper and look at distributional alignment ideas from natural language processing.

So the point I'll end on is this concept that you don't always need paired data to solved conditional prediction. And one name for pairing is grounding, grounding one modality in another. And there's just really interesting implications, like I could potentially identify the meaning of a word without ever having seen a paired example of what that word corresponds to in the physical world. And I'll have more to say on that in some future lectures. Thank you.