[SQUEAKING]

[RUSTLING]

[CLICKING]

PHILLIP ISOLA: OK. Yeah. Welcome, everyone. So now we're coming to another section of the course, which is generative models. We'll have three lectures on generative models.

And generative models are-- the new name for them is generative AI. So this is the big thing of 2023, and continuing on to this year, 2024.

OK. So we're going to learn all about what generative models are. Again, this is one of those topics which you could teach without mentioning deep learning at all. Of course, we'll talk about deep generative models. But the place where generative models are having innovation is mostly in deep learning. They kind of go hand in hand.

OK. So I want to come back to this picture because we've had three lectures now on representation, learning, and the way I like to think about generative modeling is it's just the inverse of representation learning. So these two things, representation learning and generative modeling, are very tightly coupled. And we'll see, actually, in the next lecture, models that do both representation learning and generative modeling at the same time.

But we're talking about this direction now. So we've so far, in the last few weeks, talked a lot about going from data to some kind of low-dimensional embedding. And now we'll be going from simple low-dimensional embeddings to data.

OK. So today will be just a tour of a bunch of the fundamentals of generative modeling and some of the popular models like diffusion models and GANs. And then, next week, we'll be looking at variational autoencoders, which are a model that does both directions jointly.

And then we'll talk about conditional models, which is, How can I condition a generative model on some data to predict some other data? And in lecture 16, we'll really get out, What are these things useful for?

So it might be a little confusing when you first come across generative models. You know, like, why do we care about these things? There are things that make random images, random stories. Why would you care?

Well, we care because-- you know, ChatGPT is a generative model, and we know that that's something that's popular. But we'll talk more about the applications of generative models in lecture 16. And yeah, I love generative models.

I just think that-- my favorite topic is representation learning, but my even more favorite topic is generative modeling. And I think they're really the same topic. And this is just, like, the coolest thing in AI.

So to me, generative models are kind of the core of intelligence. And that's kind of played out in the last few years with this generative AI revolution. But it's just super cool stuff.

OK. So I'm going to start by giving a little math background because we're going to be using a different type of math than we've used previously in this class, which will be probability. This will be things that you should have already seen. If you haven't, you should touch up on this.

OK. Then we'll talk about, What's a generative model? We'll talk about density functions, energy functions, and samplers. And then we'll get into a survey of a few of the currently popular generative models.

So one will be the autoregressive model. The next will be the diffusion model. And the next will be the GAN, the Generative Adversarial Network.

So there's a lot. You can have a whole lecture, a whole class on diffusion models. But we're just going to go through the basics of it. And hopefully, you'll have this toolkit that you can read up on in your own work.

OK. And the good news is all generative models use the same principles. So it's not-- there's very tight connections between all of them. You can often cast one model as another model with some exchange of variables. A diffusion model is a type of variational autoencoder, and an autoregressive model is a small variation on a diffusion model. They're all connected.

OK. But you don't know what those terms mean, presumably, yet. So don't worry. We're going to see some connections.

So what I want to start with is, yeah, What is the definition of a generative model? And there's basically two definitions in the field. There's the definition that I will focus on, which is an algorithm that generates data. And there's a definition that you may have come across in a statistics class or a probability class, a slightly older definition, which is the joint distribution of some data, so probability of a joint configuration of x's, y's, and other variables

OK. So these are actually highly related concepts. But I think it's-- our real goal in generative modeling, especially in this area that we would call generative AI, is we want algorithms that make things that look like data.

So what's data? It could be images. We want algorithms that make images.

What's data? Proteins are data. We want algorithms that make proteins, design proteins, design drugs.

We want algorithms that write stories and chat with us. This is generating text. That's data.

So this is in contrast to other forms of AI, which are more about analyzing or classifying or deciding-- making decisions, taking data and saying, this data is of type x, or this data means I should turn left with my car. So going from data to decisions or abstract representations is what we've seen before, but now we're going from something to data.

OK. But most of the time, we'll be generating data by either explicitly or implicitly modeling a distribution. And I think that this is all probably well known to you at this point in history, but these things are all over the world now.

Generative models are what make it so that you can type in a sentence on your favorite text-to-image model and say, a photo of a group of robots building this data center, and you get almost photorealistic pictures. Or you can make videos of this now. This stuff is advancing so quickly.

But what you might not know as much, or not see yet in the media as much, is that generative models also can be used to generate scientific data, data that is not just for entertainment but might be for some kind of engineering application or logistics data of some form. Or in this case, this is a diffusion model, a type of generative model, which is generating some structures that describe the geometry of a protein. So we have this input, which is, like, I want you to make me the-- I want you to tell me what the binding sites are on a protein, and I'll use the diffusion model for that process.

Here's another example. Here's an image-to-image model. So it's generating-- the output of this model is an image. But the output is a CT scan image, and this is conditioned on an input which is an MRI image.

So it's taking it and it's saying, What would it look like if the doctor hadn't scanned with an MRI machine but had scanned with a CT machine? So we'll talk about that setting as well. OK, so generative models are not just for entertainment, but where they became popular was, yeah, writing funny stories and having chatbots and making funny images.

OK. So let's go over a little bit of math background to get us all on the same page. And this is material that if you haven't seen it before, you should review. But you should've taken some kind of probability or statistics class. So we are going to be assuming that.

OK. But we're going to be changing the way we think about neural nets. So, so far, we've mostly thought of neural nets as functions that map from some input data to some output data. And in fact, we've thought of them as the classical definition of a function, which is, for each unique element of the-- for each element of the input space, there is a unique element of the output space.

So it's a many-to-one mapping, or maybe a one-to-one mapping. But it's not a one-to-many mapping. So for every element of the input, there's only one output.

And that's not going to be sufficient for generative models because we want to make we want to make a whole distribution of things. And so we're going to change our perspective a little bit.

Let me just give an example of the function perspective on neural nets. For an image classifier, the input might be a tensor, an "N by M by C" number of color channels, a dimensional tensor. We're going to learn a function f.

It will be a neural network. Our goal is to output a class prediction. So it will be an integer from 1 to d, where d is the number of classes

OK. So instead, we're going to now think of neural nets as-- or the models that we'll consider are going to be mappings from some input space X, not to a object Y, but to a distribution over objects Y. So we call that, with this fancy script, P, the space of probability distributions over the space Y.

And that might seem a little bit weird, but it's like going from data to a distribution over possibilities, not just one answer. And we've seen that, in fact, already. In fact, almost all the things that we've seen or you'll encounter in machine learning can be recast from this perspective.

So one example is classification. We actually model it as softmax regression. This is a standard thing we've been doing in machine learning.

We learn a function from, in this case, let's say, images, to a distribution over the probability that the image contains each of the d classes. So this little triangle is meant to indicate the simplex, which is the space of all categorical distributions over d categories.

And for d categories, it's the d-minus-1 simplex. So if I have a binary classifier, 0 or 1, then my distribution will be, How much probability do I place on class 1 and how much probability do I place on class 2? f that will be a line.

OK. So this is the way we'll think of what we're actually modeling. We're modeling the probability of some output conditioned on the input, which is thought of as a random variable, takes on some value x.

And again, this is something that does characterize the classifiers we've seen in the past. If we have a vanilla regression, where we're doing regression with respect to a Euclidean like squared error loss, then you can recast that as outputting a Gaussian distribution over the possibilities, and the mean is what you're regressing.

OK. So the main perspective here is that the outputs of neural nets are, either implicitly or explicitly, going to be distributions of possibilities, as opposed to just one, like, point prediction. So this was in that hacker's guide lecture. I talked about this idea that in a lot of deep learning, we're going to be predicting something from something.

And this little trick, this hack, is that you make the thing you're conditioning on-- so you're going to predict the class of an image. But better than that would be to take your whole experience of the photographer's life before they took that photo. If you condition on more information, like a video, then you get less uncertainty over the output.

And the neural nets we had looked at before were making these point predictions, just saying, for this input, I'm going to predict, What is the possible label Y, or what is the possible outcome Y? But in reality, there's some set of possibilities. It's not always that there's only one answer.

There could be-- this image could be a cat with probability of 50% and a dog with another probability. And it becomes a complicated distribution to model. And I said, just put more information into X to reduce that uncertainty.

But now I'm going to tell you, if you don't have that luxury, if you can't put more data into X, what do you do? How do you model a complicated output distribution that's not just a point? OK.

OK. So that's what generative modeling tries to deal with. So a little bit more notation-- so we'll talk about random variables. They'll be characterized by a distribution.

So a random variable is a set of possible atoms, and there's a distribution over what values that random variable will take on. And that distribution is either discrete or continuous. If it's discrete, we call it a probability mass function. And if it's continuous, we will call it a probability density function.

And a realization of that random variable will be indicated with a lowercase variable name. So a sample from the distribution of possible settings of that random variable will be a realization x. And we'll call that the probability of x. And you can think of that as a mass, in the discrete case, or a probability density of x, in the continuous case.

OK. So uppercase letters, in this course, are random variables. They're characterized by distributions. Lowercase letters are realizations. They're characterized by scalars, just the probability of that realization.

OK. So just, formally, a probability mass function is a mapping from data to scalars. And the scalars are between 0 and 1. So the probability can be 0 up to a hundred percent, up to 1. And the sum of all the probabilities over all possibilities sums to 1. That's just the definition of probability.

So if you ever have a vector that sums to 1-- like, let's say you do L1 normalization of a vector of activations in a neural network, and all of the-- and also, you have a sigmoid, so that the numbers are all between 0 and 1-- well, that vector, you can call that a probability mass function. That's just mathematically what a probability mass function is.

Probability density is the same thing, except it's over a continuous space and it integrates to 1. And the densities don't have to be-- they can go to infinity. So it's just that the densities are nonnegative. OK.

So again, this should be stuff from probability, intro probability. But it's a little bit of a different way of thinking about things, and it's the fundamental language of generative modeling. So you should refresh yourself on that.

And the notation that is linked in the schedule for the very first class, and that you should be using on your problem sets and we're using throughout the class, has a section on probability. And we have some shorthands.

So one shorthand that might confuse you a little is, if I write p of capital letter X, I mean a distribution of this random variable. And if I write p of lowercase letter x, I mean the scalar probability of random variable X taking on value x.

So this is shorthand for the distribution over X, and this is shorthand for the distribution-- sorry, the scalar probability of the realization random variable X takes on the value little x. So this is on this slide as well, from the notation doc.

OK. Cool. Any questions about the math background before we now get into generative models? Yeah?

AUDIENCE: When you say continuous probability function, how can you define the density of X? I mean, my understanding is that it's always zero.

PHILLIP ISOLA: Yeah. Great question. So the question is, For probability density functions, how do you define probability? The probability of an infinitesimal thing shouldn't go to 0. So the definition is you always only define the probability of a variable taking on some value over an interval, so the probability of x being between 0 and 0.1.

So it's all defined with respect to integrals. But usually, we don't worry about that. This is, like, when you start thinking about measure theory and the proper way to define things.

But just think of it like the probability that X takes on that value. It's really defined as some integral. Yeah?

AUDIENCE: Could you explain one more time how we can think of a neural network as an implicit distribution?

PHILLIP ISOLA: Yeah. How can we think of a neural network as implying a probability distribution? Well, I'll show a bunch of examples of this. But basically, let's say that I'm doing regression with an L2 loss. I'm going to predict some number. I will interpret that as, I am predicting a distribution over outcomes, parameterized by the mean equals my output of my neural network.

So that's like-- and that interpretation is valid because the probability of the data under that Gaussian model is equal to the squared error loss function. So this is something-- you can look into the connection between Gaussian max likelihood modeling and least squares regression.

AUDIENCE: OK. So it's not that the network is sampling of a distribution, it's that the output is a distribution itself?

PHILLIP ISOLA: We'll see both. So there's two possibilities. One, the network is outputting samples from a distribution, and the other is the network is outputting parameters of a distribution. So I'm going to talk about both.

OK. So let's build up some intuition. So that was math background. We're going to get some intuition, and then we're going to get back to the math.

So I like to start with generative models from this perspective, that we already know about things like classifiers take data and output a label. And so again, I like thinking of generative models as the inverse of that, the opposite direction. Take a label or some abstract description, like, I want you to make me a bird, and output the thing that we would normally call data.

There's no definition to label versus data. But you kind of have an intuition about what that means. So, yeah, precisely, mathematically, there's really no distinction here. But generative models are when the outputs are something you would think of as data.

OK. So what is the problem? The problem is that in this direction, there's many possible outputs that match the input.

So in this direction, there's only one possible label that is the correct label for this bird. But in this direction, that robin is just one possibility out of many. So there's the distribution over correct answers.

So how do we model the distribution? So we're going to see a few different ways. But here's one of the ways.

We take a function generator, called a generator. That's a neural net. It's a deterministic function, but we condition it on some random variable.

So we make multiple outputs by taking the function and conditioning it on some random variable. And that will be these dice. So we roll the dice. We input that into our system. We get one outcome. We roll the dice. That's a different setting of numbers, and that gives us another outcome. And this is the way that you can turn a normal neural network into something that can sample from a distribution, by conditioning it on random variables.

OK. So sometimes, people will call these random variables-- these dice, in this diagram-- as noise. I think that's a confusing term. They're noise in the sense that they are random variables, so they're randomized. But they're not noise in the sense that they're meaningless or something that you want to get rid of or ignore.

These dice rolls, you should instead think of them as specifying everything about the data that's not specified by your input command. So if I say, I want to make a bird, well, everything else that's not specified by that-- like what color is the bird, what species is the bird-- has to be specified, made deterministic by, determined by these random variables.

So they're specifying, What's the color of the bird? If I roll that, I'll change the color of the bird. What's the angle of the bird? If I roll that, I'll change the angle of the bird. And what's the size of the bird? OK. So that's kind of how a generative model will output randomized possible birds for the input command, make me a photo of a bird.

OK. So rather than thinking of them as dice rolls, another way, you could think of them as these sort of control-- the underlying control variables that control the output that you're going to get. And I could spin this knob randomly, and then it's like noise, a random variable, that gives me a random bird.

But I could also set that knob, that dice-- now I'm going to think of it as a knob-- to a value of the bird I want. And this is something that we'll see in a lot of the applications of generative models, gearing the models by controlling the latent variables that are input into the models. OK.

OK. So this is all kind of to build up this intuition about what's going on. Now let's-- I want to have you make a generative model really quickly. So one of the places where generative modeling got started is statistics, but another place where it got started is computer graphics, actually.

So in computer graphics, the name for a generative model is-- the classical name is procedural graphics. It's a way of making random content for your video game or movie. I want to draw a random map for my video game, so I will write down a procedure that creates a random map. That's a generative model.

So let's make a generative model like that. So we're going to take in a random variable. It will be a coin flip, so a Bernoulli random variable. And we're going to try to make a generative model of rivers.

So here's how it's going to be. Does anyone have a coin? Somebody has a coin? An object you can flip. Google can do it for you, if you want. You can search "coin flip."

AUDIENCE: I have a coin.

PHILLIP ISOLA: OK, good. So why don't you do this? So here's the start of the river. So flip the coin, and it will determine if the river goes left or right.

AUDIENCE: Heads is left?

PHILLIP ISOLA: OK. Yeah. Heads is left.

AUDIENCE: Heads.

PHILLIP ISOLA: OK. Left. Again.

AUDIENCE: Heads.

PHILLIP ISOLA: OK. Left. Again.

AUDIENCE: Heads.

[LAUGHTER]

PHILLIP ISOLA: So I don't know if you can still see it, but anyways, going that way. One more time?

AUDIENCE: Tails.

PHILLIP ISOLA: OK, cool. So I think that's a pretty cool river. And then my computer rendering engine would probably make it a little thicker, smooth out the borders, paint it blue. And then there I go. I have randomly generated a river from a distribution.

You could do the same trick with trees. You could flip two coins. One coin is, like, how many branches-- maybe roll a dice, how many branches, and then the other coin is, like, What's the angle of the branch? So you can write little scripts like this. You probably have done it in some of your coding experience.

OK. So I wrote a little program that does that. It's just exactly what we did. So what is the program doing? It's flipping a coin N times and drawing the river by just changing the heading of the river by some 10-degree angle. Depending on if I flip heads or tails, it will choose left or right.

OK. So what is that? That's a mapping from random variables to generated imagery. And that's how generative models tend to work.

So here's a more complicated mapping. But again, you can specify so many details of the world just via choosing a random variable that determines each of the attributes of the image.

So in this case, I have a Bernoulli variable, N of them, for the river turns. I have a Gaussian variable for the color of the grass. It's brighter or darker. I have a uniform variable for the number of trees. This is a top-down look at some river.

And you could say those random variables are noise. But what we'll think of them as is as control knobs that specify all the attributes of the data. And a formal name for that is latent variables.

So a latent variable is any variable that determines how the data looks that is not directly observed. OK. So these latent variables are the thing that underlie the causal process of generating the data.

OK. So yeah, that's one concept to start with, is that generative models that we'll see are going to be taking noise as input and turning it into data. And you should really think of the noise as the latent variables.

OK So now let's talk about how to learn such models. So you could handcraft this, write down a Python script or play a game in a class. But how do you actually learn that program and represent it as a neural net?

And this gets to the question that was asked earlier. There's going to be two different approaches. One, I'll call the direct approach.

The direct approach is I'm going to learn a function that directly samples random generated data points. OK. It'll start from z, which is going to be our name for the noise, or the latent variables, so the dice rolls or the coin flips. And it will output data.

And the other approach will be, we'll call, the indirect approach, which is, rather than learning a function that goes from random variables to data, I'll learn a function that scores the probability of a data point. So this is a function that should assign a high score, a high scalar, to data that I like and a low number to data that I don't like.

And there's a lot of different varieties of what that scoring function could be. The most basic one would be, it's the likelihood of the data, the probability of the data, is a mapping from the data to a scalar. If it's a density, it's a nonnegative scalar.

OK. Then, if I have such a scoring function, I can generate data by searching for data points that score highly, or by sampling from the distribution given by that score. So it's like the direct approach directly give me data points. Indirect approach gives me a score that I can optimize to find good data points, or data points that are high probability.

Notation wise, there is kind of a historical name for this top one, which is called an implicit generative model. I'm not sure that that name is still around or sticking around, but you might come across it. And if you do, don't worry. Implicit sounds like it's kind of indirect, but this is the direct approach.

So what it means is that the sampling procedure implies a probability distribution you're sampling from. That's why it's called implicit. So this direct sampler implies a distribution that you could have drawn samples from.

And this distribution, this scoring function, implies a sampler that can generate data directly. So we'll see both of these types of models in more detail.

OK. So let's start with the direct approach. The direct approach is the one that is kind of the more popular characterization right now, whereas the indirect approach is the more classical characterization of generative models.

So the direct approach is, I start with-- I train the model by giving it a lot of examples. It's just regular machine learning. Right, I give you examples, and I try to make more stuff like the examples.

And I'll take the data. I'll output parameters of a parameterized function that can map from random noise or from latent variables that are randomized to-- that have simple distributions, to data samples that look like the training data. So the objective is to make more data that looks like the training data.

OK. So this, in our class, will be a neural network. But it doesn't have to be a neural network. But that's the thing that tends to work the best.

And we call these two phases training. And now we use a different term. Rather than testing, we might call it sampling.

But it's the same as the testing phase for any neural net that's fit to data. It's just, when I run it, you might call it inference or deployment. But from a generative model, we can also just call it sampling from the model. OK.

So the indirect approach is-- I already described it, but let's look at it in a little bit more detail here. So it is, take some data-- and this is what you would've seen in your statistics classes or probability classes. You take some data and you learn a scoring function that places a high score where the data lives.

OK. So the most standard one would be a probability distribution that has high probability where the data lives, so that the samples that you observe are samples from the distribution that you learn. But it could also be what we'll call an energy function, which I'll show you, or something more exotic, some other type of score function.

And then, sampling is no longer direct. Now, sampling is, I have my function that scores the probability of every possible value of x. So this is a one-dimensional distribution, the x-axis.

What's the probability of coming up with this value of x? It's high probability. And I can sample from that distribution using another algorithm, which will be called a sampling algorithm.

And we won't go into too much detail about these. But you probably-- you may have seen these before as-- Markov chain Monte Carlo is one type of sampling algorithm.

Another type is called rejection sampling. You just take a random sample and reject it in proportion to its probability under that scoring function. So there's a lot of sampling algorithms for how to go from this distribution to samples from that distribution. OK. So this is the indirect approach.

So what's the goal of generative modeling? The goal is to make synthetic data that looks like real data, make more stuff that looks like your training data.

So the modeling questions to answer are, What does it mean to look like the training data? What does it mean for it to be making more stuff that's like the training data? It doesn't have to be an image. It could be "looks like" in an abstract sense.

And there's a lot of answers to this question, that lead to different types of model. But we've kind of converged or one main one in machine learning, which is that you should make more data that has high probability under a density model fit to your training data.

So this is called the max likelihood objective. We're going to go through that in a little bit more detail now.

So here's, again, the density model. It's a mapping from data to positive numbers-- or nonnegative numbers, in fact. And it's going to be parameterized by parameters theta, which are the weights and biases in a neural network.

And think of that as assigning a likelihood, a probability, to every point along some data space X. So here, data is just a one-dimensional space. And here's my training data. My training data is examples of what the data looks like, and I want to make more like it.

So in order to maximize the probability that my model P theta places on the training data, I will start with some model. Maybe I have-- my neural network is going to output this shape, so it's going to output a structured object of this shape. Now, how does it actually do that?

It's going to output the parameters of a parameterized function-- like, a Gaussian is a parameterized function specified by the mean and the variance. So I could output the mean and the variance, and my neural net is telling me that that's the distribution it thinks explains the data.

But it could be a mixture of Gaussians, or it could be any-- yeah, it could output, like, the shape of some spline. So the network can output a set of numbers that parameterizes the distribution.

OK. So here's what I'll do with max likelihood modeling. I'll simply-- so the definition of a probability distribution is constant mass, right, so this area under the curve has to be constant, 1.

OK. So I will push up the probability I place where the data lives. I'll increase the probability my model places on the data. And because it's constant mass, I'll have to take away mass from somewhere else, where the data does not live.

OK, so I'm outputting a set of parameters that characterize a parametric family of distributions that integrate to 1. So every possible set of the parameters specifies a curve, where the area under the curve is 1. And if I increase the curve where the data lives, I necessarily decrease the curve where the data does not live.

And that's the idea of a density model. So a gradient update to my density model will increase the probability I place on the data and, therefore, decrease the probability a place elsewhere. So the output of my neural network is a set of parameters that characterize this curve. Yeah? Question.

AUDIENCE: Can you go back one slide, please?

PHILLIP ISOLA: Yeah.

AUDIENCE: One thing I'm just a little confused about is you have p sub theta mapping each data point to some probability between 0 and infinity, but normally, I think of probability distributions as summing to 1. So it's hard to reconcile, like, how you can have points with very large probability, potentially.

PHILLIP ISOLA: Yeah. So I think the question is, if the probability density can go to infinity, how can I-- like, how does that make sense numerically? How can that still integrate to 1?

This is that kind of subtlety of continuous probabilities. We characterize the distribution with probability density function, which is not actually a probability. The probability is an integral of the probability density over some interval, and that specifies the probability of being in that interval.

And if the interval is infinitesimal, then the density can go to infinity over an infinitesimal interval. So it's like infinity divided by infinity, roughly. You get into the subtleties of continuous math. Yeah.

OK. Cool. So what is our objective going to be to fit my density to data? I gave you the intuition. But formally, we like to minimize the difference between the world's true data generating distribution-- so these data points, these black data points, came from some unknown process.

We're going to call that unknown process pdata. We'll use that all over the place in the next few lectures. pdata is just the probability from which-- the distribution from which the training data was sampled. But we don't know that. We don't have access to that.

And then we're going to fit our model to be close to that distribution, according to the KL divergence. It's just a measure of the difference between two distributions. And we want our model distribution to be the same as nature's distribution that generated the training data.

So we're going to see that it will come out to just pushing up probability where the data points live. But mathematically, here's what it looks like. So this is the definition of KL divergence.

It'll be negative log of how much probability the model places on a data point, divided by how much probability nature placed on the data point. OK, so we want these to be the same. So we want these to go to 1, so that log of 1 will be 0. And then I will be minimizing this, and I'll eventually get these to be equal.

OK. So we can separate out the log of the ratio into how much log probability I place on the data-- sorry, how much log probability the model places on the data, minus how much log probability nature placed on the data. I want that difference to be 0.

We can't actually measure how much probability nature placed on the data. We don't have access to that. We only have access to samples from pdata. We don't have access to the function pdata. That's just the universe.

OK. So the nice trick is that in order to maximize-- so, OK, we're minimizing the difference. But now we got rid of the negative sign, so going to maximize this term, this equation. Well, in order to maximize this with respect to the parameters theta, the parameters theta don't appear in the second term, so that's just a constant.

So we don't actually have to consider that in the maximization. So we drop that second term. And that's great because we didn't even have a way of evaluating that second term, since we don't have access to pdata directly.

And instead, we are going to maximize the probability the model places on samples from the distribution. And whenever we try to maximize an expectation, in machine learning, we're going to tend to always approximate that with a finite set of samples.

So this is the probability the model places on the population. And this is the probability that the model places on the set of samples from the population, which are our training data. And this is called max likelihood learning-- maximize the likelihood, the probability, that the model places on your training data, which are samples from some unknown data generating process, pdata.

OK. So you should recognize this as being the same form as we saw with softmax regression, or it's kind of-- max likelihood is usually the thing that we're doing in data modeling. It's often the thing we're doing. OK. Yeah? Question.

AUDIENCE: If you don't have a regularization term, is this just going to lead to, like, Dirac delta spikes on your data points?

PHILLIP ISOLA: Yes. OK. If you don't have a regularizer, will this just lead to Dirac delta function on your data points? That's going to be the next slide. Wonderful.

So this is the question that comes up all the time in generative modeling. It's kind of confusing. So here's a question for you.

Is the filing cabinet-- remember the filing cabinet from the lecture on generalization? Is the filing cabinet a good generative model?

OK, so what is the filing cabinet? I'm going to define it as follows. I will train the generative model by taking my data set X, and I'll just put every data point into a file. So I'll just add to the cabinet.

And then to sample from my learned generative model, this cabinet, I will simply take a random drawer, or random file, and pull it out. So is this a good generative model. What do you think?

AUDIENCE: It doesn't make new stuff.

PHILLIP ISOLA: It doesn't make new stuff. OK. But it does make data which is identical to the training data. So that's like-- that kind of seemed like what we wanted. We wanted to make data that's identical to the training data, right?

This does maximize the likelihood. This is the max likelihood solution, or this is getting the highest likelihood you could possibly get. Just memorize the data and output the same data.

You're right, but any other answers? Want to elaborate on this? How could we avoid this solution? This is intuitively a bad solution. I didn't do anything. Any thoughts?

OK. So here's how I like to think about it. It's exactly the same as overfitting in classical machine learning. So if you understand overfitting, then you can understand why the filing cabinet is a bad generative model.

So here's my training data. And exactly as you said, the optimal generative model-- and the max likelihood generative model, if I have infinite capacity, I can fit any wiggly curve-- is to put a delta function on all the data points, on all the training points.

But the true data generating process is something unknown, which here, as a cartoon, it could look like that. So these blue dots are just samples, my training data samples, from some underlying process.

So what went wrong here? If I take new samples from that same process, they'll land somewhere else. And I'll have placed zero probability, under this model, on those samples.

So what matters in generative modeling is the probability you place on the test data, on new samples from the data generating process. We optimize the probability you place on the training data, but what matters is the probability you place on the test data-- exactly the same as overfitting in machine learning, where you actually care about test performance, not training performance.

But here, test performance is going to be the density you have placed on new samples from the true generating process. And you have to control capacity or regularize in order to avoid the memorization solution.

OK. So that's what overfitting looks like for a generative model. So the goal of a generative model is not to make exactly the training data or, actually, maximize likelihood on the training data. It's to make things that are new but capture the important statistics and realistic properties of the data, but are infinitely new.

We can quantify that as likelihood of the test data under a model. Now, if I don't have an explicit representation of the density, then it's not easy to quantify this. And that's why these methods, the direct methods, that just produce samples but not densities, are a little bit hard to calibrate.

It's a little bit hard to know what the test data likelihood is. But with density models, I can just cross-validate by having a test set and measuring the likelihood I place on the test set. OK.

So here's the way that you can measure the quality of your generative model. You can evaluate it. You can't fit on this.

This is like the test score, the validation score. You sample new samples from the world. You make a split, your test split or your validation split. And you measure generalization error as the likelihood your model places on the test data. OK.

OK. Any questions about density models? And then we're going to do energy models. Yeah?

AUDIENCE: How exactly is p sub theta parameterized? Like what is it actually doing?

PHILLIP ISOLA: So theta, how is it parameterized? You can parameterize it in a lot of different ways. So one example would be the network outputs two numbers, two scalars, so it's a two-dimensional vector.

That would be sufficient to parameterize a one-dimensional Gaussian. The first number is the mean. The second number is the variance.

Another option is the network could output a thousand numbers, and that would be sufficient to characterize-- or, let's say, a hundred numbers, or a hundred plus 10 numbers, that would be sufficient to characterize a 10-dimensional Gaussian where I have a covariance matrix which is 10 by 10, so a hundred numbers, and a mean which is 10 dimensions.

So I have to output the number of-- the vector output by my network has to be all the parameters of the family of distributions that I am considering. And if I want to do a mixture of Gaussians, then I have to-- if I want to output two Gaussians, I can do that with 2 times 100 plus 10 numbers. And if I wanted to output other, funny-shaped curves, I can do that with other parametric families. It doesn't have to be a Gaussian. But that's the general idea.

AUDIENCE: So assuming that you have a finite set of validation data-- I assume your procedure would be to train this until your validation loss starts to increase, and then you do early stopping to try and find a good point at which you've got a good generative model. Is there an analogy similar to what we learned before about inductive bias in these models?

Presumably, if you're trying to generate images, you should somehow be able to add a form of bias or regularization that says, like, images should not look like static. You should not be able to have just a zero-value pixel next to something that's a max-value pixel, so images have this sort of smooth quality. So does that show up in how you regularize in the loss? Or is that a [AUDIO OUT]?

PHILLIP ISOLA: Yeah. So yeah, great question. So can we add inductive biases to generative models to prevent them from finding bad solutions and kind of encourage them to find a good solution that actually generalizes to the test data? And all the tricks that we have from other deep learning can apply here.

So you can use an architecture which has some kind of spatial smoothness. So a convolutional architecture has baked-in spatial locality. You can use positional codes in transformers that have baked-in kind of inductive bias towards spatial locality. You can use regularizers that-- you can do all the tricks, basically. There's a lot to say, but you can do everything. Yeah. OK. Question in the back, and then I'll move on.

AUDIENCE: Is there a connection between max likelihood estimation and empirical risk minimization?

PHILLIP ISOLA: Yeah. Is there a connection between max likelihood estimation and empirical risk minimization? Generally, yes. I guess I would say empirical max likelihood, which is the maximized likelihood you place on the data-- that's what empirical means, is on the training data-- that is a form of empirical risk minimization for the risk function which is the likelihood of the data.

But you could have more general risk functions that might not have interpretations as likelihood. Yeah. But typically, the ones we use, like squared error or L1 error, these all have interpretations as some type of likelihood function.

So let's talk about energy-based models. So energy-based models are-- I said that the indirect approach-- this is another possible indirect approach. The indirect approach is you try to score your data and then sample from or optimize the score, find data points that have high score.

Energy-based models say, Well, what if the score is not a probability, but it's just an unnormalized probability? So remember that the definition of a probability density is it integrates to 1. An energy function is just a probability that is unnormalized.

So for any probability, I can represent-- or, yeah, for a broad family of functions, I can represent them like this. This is sometimes called the Boltzmann distribution. So it's an exponential of the negative energy divided by a normalization.

And any energy function, any function E, that is not necessarily normalized, will, through this map, imply a probability function p that is normalized, because I just divide by the normalization constant. So whatever E is, if I divide by Z, which is this integral, p will be normalized. Yeah?

AUDIENCE: What is an energy function?

PHILLIP ISOLA: Energy function is just a map from data to a scalar, between negative infinity and infinity.

AUDIENCE: So kind of like a loss, in a sense?

PHILLIP ISOLA: Just like the score-- so think of it as an unnormalized probability.

AUDIENCE: OK.

PHILLIP ISOLA: Yeah. Unnormalized probability. Yeah.

AUDIENCE: Kind of like the thing that goes into softmax?

PHILLIP ISOLA: Yes. So the logits in a softmax classifier can be considered the energy. And then, softmax is just, you divide by the normalization-- you exponentiate and you divide by the normalization. OK.

OK. So why are energies good? Because they're often sufficient. So, OK, so first of all, why are probabilities bad? Because probabilities need to be normalized.

So my family of functions I search over is just normalized functions. And if I search over arbitrary curves-- like, let's say the neural network doesn't parameterize a Gaussian, which is normalized, but it just outputs some polynomial curve. Or maybe it just outputs a set of numbers that are, like, the control points on a spline. Whatever. Or maybe it just outputs, like, a number per data point.

So I could have a network that takes in data and just outputs a number between negative infinity and infinity. Well, how would I normalize that? I'd have to do this integral over all possible settings of the data. And that's intractable. I don't want to do that.

So that's why probability densities are bad, because I have to consider families that are normalized, and that's very restrictive. But energies are sufficient because energies, energy ratios are equal to probability ratios OK. So the probability that I place on-- let's say I want to say, Which is more likely, that it's going to snow tomorrow or there'll be a hurricane tomorrow?

So that's some probability ratio. And maybe that's the type of thing I need to decide for how to prepare for some disaster. Oftentimes, relative judgments are what matter.

And so relative judgments, the relative probability of x1 divided by x2, is equal to-- here's the definition of my energy function. And notice that the normalization functions cancel out. So that means that exponential energies provide the same ratio as probabilities. And I can evaluate whether a hurricane is less likely or more likely than snow by just looking at the energy ratio, as opposed to the probability ratio, which is intractable unless I know the normalization. Yeah?

AUDIENCE: So it's the energy ratio?

PHILLIP ISOLA: Yeah.

AUDIENCE: So I guess energy ration includes exponentiation.

PHILLIP ISOLA: Yeah.

AUDIENCE: So it's quantum difference, or?

PHILLIP ISOLA: Yeah, you could do like-- so the question is like-- I said energy ratio. I meant exponential. e to the negative energy divided by e to the negative other energy is equal to the probability ratio. If you took a log of that, you could see, it'll have another form. It'll be some difference. OK.

So this is-- relative probabilities are often all you need for decisionmaking. In particular, they're also all you need for a lot of samplers.

So Markov chain Monte Carlo only requires relative probabilities, so you don't need to have a normalized density or output. You can just output arbitrary unnormalized numbers and still sample from that, the implied distribution. OK. So energy functions are often all you need. Yeah?

AUDIENCE: Can you explain the negative sign in front of the energy?

PHILLIP ISOLA: Yeah, so that's just a definition. You could have-- you don't need that. It's just the definition.

Because E is just a function that outputs negative infinity to infinity, so the definition is energies are related to negative probabilities. So low energy means high probability. It's just a definition.

OK. So why would you actually want a probability? Why wouldn't you always work with energies? Anyone have some thoughts on that?

Why would I ever care about using normalized probabilities, if what I'm saying is true? Yeah, any thoughts on that? Yeah?

AUDIENCE: How would you scale them to different concepts, perhaps?

PHILLIP ISOLA: How would you scale them?

AUDIENCE: To different concepts.

PHILLIP ISOLA: To different concepts. Yeah. So how would you scale them to different concepts? I think that's getting at what I think I agree with, that probabilities put everything on the same kind of unit, on the same playing field.

Like, you're kind of scaling things to have the same units. So 90% probability could be-- in one domain, it's 90% probability in another domain, but the energies could be arbitrarily scaled or arbitrarily offset.

So I think probabilities are often more interpretable and more exchangeable between different problems. If you go to the hospital and they tell you your energy of having cancer is 10 million, what are you going to do with that, right? But if they say your likelihood is 0.1%, you're going to be like, OK, that's fine. OK. So human interpretability is the most obvious, but I think there's other reasons beyond that, too.

So how do you optimize an energy-based model? How do you fit an energy-based model to data? So here's how you do it.

So we do the same thing we did before. We find our data and we decrease the energy where the data lives. Remember, decrease energy means increase probability, because energy is related to probability by negative. e to the negative energy is probability.

OK. So what's wrong with this? We're going to decrease our energy where the data lives. What's wrong with this? What's wrong with this?

AUDIENCE: It's not constant.

PHILLIP ISOLA: Sorry. Yes. Not constant mass. What's wrong with this is I could just decrease my energy to be negative infinity everywhere.

If my only goal is decrease energy where the data lives, I can just say, well, I'll also decrease energy everywhere else. I'm not forcing it to be normalized. So there's no necessity to increase energy where the data does not live.

So the simple solution is I will add a term to increase energy where the data doesn't live. So the way you do this is you decrease energy where the data lives and you increase energy where the model places high energy. You actually sample from the model.

So I said you can sample from an energy-- you can sample from the distribution implied by an energy without having the normalization constant. That's what things like Markov chain Monte Carlo do.

So I can sample from my energy function, but I can't evaluate the probability. But I can take these points, and I'm sampling where there's low energy because that means high probability. I'm going to increase energy there, and I'll get this solution.

If I repeat this process, I decrease energy where the data lives and I increase energy where my current energy function implies high probability via sampling. I go like that. And eventually, I get this.

And here, I decrease energy where the data lives, and I increase energy where the model has placed high probability. And they cancel out. So eventually, I get a delta function on the data points. And that convergence of this algorithm, the positive term, which is decrease in energy, equals the negative term, which is increasing energy, because the model places energy exactly where the data lives.

OK. So that should, therefore, kind convince you graphically that this thing will converge to a solution that fits the data. It will overfit in the same way that other generative models overfit. But we'll just do early stopping or regularization of various forms.

OK. So at convergence, the green arrows and the red arrows cancel out. I get zero delta between them, and the thing stops. So let's look at that mathematically.

This algorithm is called contrastive divergence. And variations on this come up a lot. It's even connected to contrastive learning, but a little bit indirectly.

It's another contrast, the contrast between where the data lives and where the model has placed high probability. You want that to cancel out so that the model places high probability exactly where the data lives.

So mathematically, here's how you derive that. This is the kind of thing that you might want to work through yourself after the lecture. But I'll go through the steps for you here.

So we're going to do max likelihood learning with energy-based models. So max likelihood learning is, I want to update the parameters, find the gradient of the parameters that will increase the probability I place on the data. And I'm writing it in terms of the expectation over samples, but you would really do this on an empirical training set.

OK. So I can rewrite my probability distribution as an energy function divided by a normalization. OK. And now I'm parameterizing the energy function. So the neural net E is just a mapping from data to a number. And I'm going to parameterize that by theta, which are the weights and biases

OK. Now I'm going to take the log and split it into two terms. It'll place high energy where the data lives, and then have this extra term related to the normalization integral Z.

OK. So that's called the positive term. That's those green arrows. And this is the negative term. That was the red arrows.

So this one's easy, right? This one is just, I have my samples from the data, and I just do backprop to increase the energy I place on those samples-- or, sorry, decrease the energy. Remember, negative energy means high probability.

But how do you measure the normalization? How do you measure the gradient of this normalization function? So here's the full derivation, which you might want to kind of walk through after the lecture. But here's what it is.

It turns out that this thing has a very simple form. So the gradient of this normalization function, sometimes called the partition function, is-- so grad of log of f of x is equal to 1 over f at x grad of f of x. That's an identity from calculus. It's a good thing to remember.

So this is equal to 1 over Z grad of Z. Now I can write the definition of Z. The definition of Z is the integral over x of-- this is like we're doing the softmax.

This is exactly what we do. This is just the more general setting. This is the normalizing constant, so the integral over how much energy I place on all the data.

OK. Oops. Sorry. So then, the next line is going to be chain rule. So this is just the gradient of e to the negative function of x, is-- here, gradient of e to the x is e to the x times the gradient of whatever was in the exponent. That's just the chain rule.

And then this line here is just rearrangement of terms. I'm taking this ratio, dividing by Z, pulling it into the integral. So now I get this term, which simplifies to p again. So it's just rearrangement of terms. That's the definition of p is equal to this term. I found it again.

But now I have an integral of p of this thing. And this is-- the whole point is to rearrange the terms until I get something that looks an expectation. That's an expectation. An expectation is integral of p of x f of x. That's expectation of f of x under the distribution p of x.

OK, so I got an expectation. So this is all equal to expectation of samples from the model of the gradient of the energy. So what does that-- why did I do that? Because if I get things into the form of an expectation, I know how to approximate expectations on finite data.

I don't have to compute an integral over some analytical infinite set. I approximate expectation with samples, and we know how to do that. That's just sample from p, evaluate this, and backprop.

OK. So here we go. We converted this funny-looking term into now just another expectation. Both of these expectations can be approximated with samples. This is samples from my training set. And this is samples from my model.

And that's the gradient that will maximize the likelihood of the density implied by the energy for an energy-based model. It's called contrastive divergence. Any questions? Yeah?

AUDIENCE: Just one question on the derivation. You pulled out, like, a minus sign, but it wasn't-- if I understand correctly, it was not a subtraction. It was times negative 1.

PHILLIP ISOLA: That-- I think-- oh, yeah, sorry. That's a little-- this is a times negative 1. Yeah, that's not subtraction. That's probably confusing, yeah. OK.

OK. So that's the math. But you can also just kind of keep in mind the intuition. Push down energy where the data lives. Push up energy where the model currently places high probability. Eventually, those cancel out, and the model has fit the data.

OK. So we've seen a few different families of generative models now. We've seen that generative modeling takes data, and you can learn a density function that assigns a normalized score to the data. You can define an energy function that gives you a unnormalized score to the data. And you can also directly sample by learning a mapping from random noise or latent variables to data.

So you can represent the data generating process in any of these ways, either directly generate data from initial random variable or sample from a density or energy fit to the data. And each of these things, then, will have a family of generative models and deep generative models that have one of these forms. And a lot of deep generative models have multiple of these forms simultaneously.

So, finally, we're going to get to the popular, named, deep generative models. All of that was really just background on generative modeling and probability theory. So let's start. We're going to talk about three now.

We're going to talk about autoregressive models, which we've actually seen before. We're going to talk about diffusion models. And we're going to talk about GANs. And then tomorrow-- not tomorrow, but next week, we'll talk about VAEs, variational autoencoders.

So autoregressive models are kind of the simplest one to understand, so we'll start there. So autoregressive models are these next word predictors, or next data point predictors.

So I do "once upon" and I try to predict what's the next word. "A." And then I repeat the process. So we've seen this a few times. "Time."

So how is this a generative model? What probability distribution is it sampling from? And I don't have to go in temporal order. That's just a particular choice.

So, recap. I think I've shown this slide before. We trained an autoregressive model with standard supervised learning.

We have a data set of sentences. We just chop the sentence into the prefix and the final word, or the next word at some point in the sentence. That's a supervised learning problem. I can just try to treat it as classification. Predict the next word

OK. So that's my learning algorithm. And my sampling algorithm-- now I'm going to call it sampling-- is I run-- I predict the next word, but I'm not going to necessarily just predict one word. I'll predict a distribution over words. So it's like, if each word is treated as a class, I can output a categorical distribution. I can do softmax regression and sample from that distribution.

OK. So why is this a valid probability model? Why does this actually do max likelihood density modeling of a distribution? The reason is because of this equation, which is, I think, sometimes called the chain rule of probability. But it says that the probability a set of words-- or a vector X, so a set of data points X-- is equal to the probability of x2 given x1.

So x1 through xn are the elements of my vector X. So I just do probability of x1 times probability of x2 times x1. So that becomes probability of x1 and x2. And then I multiply that by probability of x3, given x1 and x2.

OK. So this is an identity, an identity from probability. OK. So I can just write that in the shorthand form as this is product over all the conditionals of the next token given the previous tokens, or the next word given the previous word, the next data point given previous data points.

OK, so here's what that looks like for the probability of "once upon a time" is equal to the probability of "once" times the probability of "upon" given "once" times the probability of-- right. OK. So it's the product of all those conditional probabilities. And autoregressive modeling is simply modeling all of those conditional probabilities in a sequence and trying to maximize the likelihood.

So how do I model probability of "time" given "once upon a"? I just do next word classifier. So I just output a categorical distribution over my vocabulary size. The number of categories is the number of words I could possibly predict.

So "once upon a," I output a probability mass function, a categorical distribution, over all the possible next words. There's a few other ways you could do it in natural language processing. But this is kind of the basic idea.

OK. What about in image processing? Here's how you would do it in image processing for an autoregressive model.

I am going to start drawing this bird, and at any point in my drawing, I'm going to predict the color of the next pixel given all the previous pixels. OK. So I can quantize my pixels-- just like I showed in one of the colorization examples earlier, I can quantize my pixels into classes-- red, green, blue, and so forth-- and output a categorical distribution over the classes. That's just softmax regression, that we've seen before.

And then I sample from it by just sampling from a categorical distribution. So if I have-- it's easy to sample from a categorical distribution, right? I can just take the unit interval 0 to 1, I can take a uniform sample from 0 to 1, and I can walk through my probability vector until I get to-- until I get to the segment that has been sampled from.

So we can easily sample from a categorical distribution. And then we just repeat the process. So I go from nothing to next pixel, next pixel, next pixel. It's the same as predicting the next word in a sentence.

OK. And this is just, again-- I think one of the early slides in this course was classifying images into categories, and here, I'm classifying next pixel into categories. And my loss function is the likelihood that my model-- my model, which is not really specified here, but anyway-- f theta places on the ground-truth class. This is the amount of probability my model places on the ground-truth class. I want to maximize that, max likelihood modeling.

OK. So yeah, just spelling it out for a different data type. So now my supervised data I learned from is examples of patches of pixels and the next pixel in the sequence.

I do supervised softmax regression, and I can predict the next pixel in the sequence given the previous pixels, and I can actually sample to create a randomized variety of birds in this way. Yeah? Question.

AUDIENCE: So would you characterize this as a density model? Can you draw that correspondence lower?

PHILLIP ISOLA: Yeah, so what would I characterize this as? A density model, sampler, energy model? What do you all think?

Who thinks density model? Raise your hand. OK. Who thinks energy model? Who thinks sampler or the direct approach?

So I think density model was the highest vote. I think that these things are a little fuzzy. You can actually-- I'm not sure there's a correct answer here.

But I would characterize it as a density model because we're outputting the density over the next pixel given the previous pixels. And we're actually going to do that for every pixel in a sequence or every word in a sequence to get a probability that's explicitly represented. So this is actually-- you can multiply all of those probabilities together, and you'll get the probability of the entire image.

So the product of all the conditional probabilities is the probability of the entire image. That's a density that integrates to 1. So I would call it a density model.

OK. And people do this in a lot of sequential data modeling problems. Time series modeling is often autoregressive. So here's another example, where we're going to predict the next sound wave in a sound. So this is, like, for music modeling or audio modeling.

I simply predict the next-- I have some representation of sound, and I predict what the sound will be on the next time step. So it looks exactly like language modeling. This is just autoregressive modeling on another domain.

OK. Right. So maybe it was a little-- let me-- sorry. Let me just go up here.

So just to clarify, I'm trying to maximize the likelihood I place on my training data. So the likelihood I place on my training data is the product over the likelihood I place on each next observation in this sequence. And if I'm using log probabilities, it becomes the sum over the log probabilities.

So the likelihood that the autoregressive model places on a training example is the sum of the log probabilities it places on every next pixel in the sequence. And I just sum all those up and backprop. And that's how you do optimization.

So that's kind of what this picture is showing, is I will just make a prediction, then I'll compare it to the ground truth, measure the probability under my categorical distribution. Then I have output of that ground-truth point, and I'll sum up all those probabilities and backprop through this graph. So that's how you actually train the thing. OK.

OK. So same as what we saw in other sequence modeling in previous lectures, like RNNs. We talked about, you output predictions, you measure the error. But now I'm just giving it this probabilistic language.

So next model that we're going to talk about is diffusion model. So diffusion models are really clever because they make the following observation. They say that-- you know, it's pretty hard to-- these are one of these more direct generative models. So they're directly mapping from a random dice roll to a sample, as opposed to going through a density or an energy function.

So diffusion models make the very clever observation that it's very hard to convert noise, random variables, into data, structured objects, but it's really easy to convert structured objects into noise, into random variables with simple distributions.

So just imagine that these birds are like balloon animals filled with some kind of colored gas, and then you pop the balloon. What happens? All the particles spread out randomly, and eventually, they become just uniformly distributed in a space.

Physically, that's called a diffusion process. And diffusion models are exactly the same thing, except you're diffusing in some high-dimensional data space. Whatever your data is, it might be a tensor, and you're diffusing by just randomly perturbing it until it becomes uniform, or it has a simple distribution, typically Gaussian.

So converting images to noise is really easy. Here's how it looks like. I just take an image and I let the pixels diffuse outward.

So I simply take every pixel and I add a little bit of noise. And I add a little more noise, add a little more noise. Eventually, I get to a state which is, like, fully entropic, a very simple distribution.

Maybe it's uniform. In the models that we'll see, it's actually going to be Gaussian. If I just add a sequence of Gaussian random perturbations, I end up with a unit Gaussian at the very end of that sequence.

So diffusion models convert data to noise and then just learn to reverse the process. So easy, right? So it's extremely hard to know how to do that mapping from noise to data, but I supervise it by telling it the exact path to take.

So every little step of denoising, of going from noise to a little less noise, is a very simple-- just remove a little bit of noise, right? It's a very easy thing to model. I can model it with supervised learning.

So I simply-- I've created a lot of noisy sequences like this on my training data. I turn them all around, and I then chop it up. And for every noisy to less noisy pair, that's like an xy target, a noisy to a less noisy. And we'll index the noise level with t, so higher noise means higher t.

I'll just use supervised learning to reverse the process, and this learns a mapping from random noise to images. And to sample an image, I simply can sample a random noise vector and put it through that mapping that I've learned, that reverse diffusion process, as it's known.

OK, so here, again, is what it looks like. I create a data set of noisy and less noisy images by having started with less noisy images and adding noise. And I do supervised learning on that.

That could be as simple as just, like, mean squared regression, try to predict, according to L, which might be the squared loss, the less noisy image from the more noisy image. And you could do this with any data, not just images. It's just an example.

OK. And this converts generative modeling into a bunch of supervised, tiny little prediction problems. OK. I'll get to the question in just a second.

So-- yeah. OK. Actually, I'll do the question now. Yeah. Go ahead.

AUDIENCE: I wanted to ask, so in each step, is each step the same model? Or are there a bunch of different models doing each step individually?

PHILLIP ISOLA: So is each step the same model? Is f the same function for all t, or is it different for t? Either option is fine.

The most common option is you make f a function of x, and you condition on t. So you just tell the network what timestep you're on. But that's a modeling choice. It's not fundamental.

OK. So if I-- I'll get to that equation in a minute. But anyway, if I want to sample a new image, well, I've converted all of my data at the end, at t equals big T-- meaning, like, as t goes to infinity, it turns-- if I'm adding Gaussian noise, it asymptotically becomes a unit Gaussian, if I do it with the right scaling.

And that means that I can now, if I reverse the process, start with a unit Gaussian sample, roll the dice, which are unit Gaussian variables, and then I denoise with my learned function. And every unit Gaussian sample gives me a different noisy image, which are like the latent variables that specify all the properties of the data that you will generate in the end.

So sampling is really easy. It's just slow because you have to do this, run the denoising neural network, over and over and over again, like, for t time steps. And when these first came out, t was large, like, a thousand. Now people are showing how to do this with t being pretty small, like, even going down to 1. But you typically have to do a few time steps.

OK. So here's the Gaussian diffusion model mathematically. So we're going to define a data generating process that starts from a simple distribution, unit Gaussian, and maps to data. And we're going to train it by training the denoiser that predicts the less noisy image from the more noisy image, to invert the noiser, which is a simple deterministic process that adds Gaussian noise to an image.

And we can condition the denoiser on the timestep so it knows, What is the current noise level? And that helps it be a better predictor of what the true data point should have been.

So the forward process is simply add a Gaussian noise, which is a unit Gaussian, epsilon t, and scale that noise according to some scale beta, which can be time-varying as well. And there's some engineering details about how you select beta. But in general, you can do this in such a way that at the end of the day, you will get unit Gaussian noise, asymptotically, as t goes to infinity.

OK, the reverse process. In fact, with the reverse process, you're going to treat every little step as a small Gaussian max likelihood density modeling problem. So the reverse process, f is not going to just predict the noiseless image-- or, not going to just predict the less noisy image. It's going to predict the mean of a Gaussian distribution over the less noisy image.

This is just-- there's various reasons, but this gives kind of a nice, mathematical, probabilistic interpretation to what's going on. OK, so the next denoised timestep will be a sample from a Gaussian whose mean is the output of my neural network denoiser.

So people like to use these terms, p for one of the processes and q for the reverse process. And it connects to the standard notation and things like called variational inference. But you don't have to worry about it too much.

We'll just say the forward process is q, which, x of t is a Gaussian centered on x of t minus 1. And the reverse process I'm trying to learn is another Gaussian, but it's centered on a function of the noisy image.

OK. So the forward process is just add noise, and the reverse process is predict the mean of a reverse Gaussian, so predict the mean and then sample from that Gaussian.

OK. There's a lot of variations and other ways you could do this, this kind of vanilla Gaussian diffusion model. And if you want, I wrote a very vanilla version of this. So you can play with the Colab if you want. But now, I think there's probably better ones out there.

I want to make the connection between diffusion models and autoregressive models. So they're almost the same thing. So both of them use a very simple trick. You change generative modeling of a high-dimensional structured object into a sequence of very, very simple modeling problems. In autoregressive modeling, you convert your data into a sequence of simple categorical density modeling problems, a.k.a. classification problems.

OK. Again, you're free to choose other types of simple sequences, but that's just the most common choice.

So think of it this way. You start with a structured data object, and you remove one data point at a time until you get to nothing, and then that supervises the reverse process.

Diffusion model, you start with a structured, complicated object. You add noise kind of globally, to every data point, until you get to a very simple distribution you can sample from, and then you supervise it to reverse the process.

This is your training data that's going to supervise the reverse process. This is your training data that will supervise the reverse process. This is easy to sample from. You're just sampling one pixel color.

OK. Let's say I have 256 possible colors. I just take a random-- I have a categorical distribution over the 256 possible colors that occur in natural images. Super easy. And then I just reverse the process by trying to classify the next pixel given the previous pixel. So the whole point is that a common strategy in generative modeling is turn a complicated structure of distribution into a sequence of simple distributions, and model that sequence of simple distributions. OK.

OK. So in the last two minutes, I'll just do GANs. They're very simple. Don't worry. I know there's a lot in this lecture, but I'm hoping that-- I want to give you the intuitions, and then in the problem sets and in the readings, you can dig a little more into the math and you can work it out.

So GANs are a whole other class of generative model. GANs are a-- OK. I'll tell you what they are, and then I'll ask you, Are they direct or indirect? So they start with a latent variable, which is a vector of random numbers, and they map it through a generator network to an image.

And then they play this game where they ask another neural network to decide if the image looks like a real data point or a synthetic data point sampled from the model. So is g a direct or indirect approach? Who thinks direct? OK, who thinks indirect?

Ooh interesting. Maybe it's more subtle. OK. I would have called it direct because we're directly mapping from a random variable to a sample from a distribution. But with d, it becomes a little more confusing. So maybe either answer could have an interpretation which is correct.

So g tries to synthesize synthetic images that fool d and-- that's a typo. g-- sorry, that should be d. d tries to identify the fakes

OK. So why should that work? That should work because if g is able to make output data points which are indistinguishable from your training data points, then it must be that you're outputting things that are from the same distribution. You can prove that, in fact.

So d plays this game. d is a classifier for, Is it a sample from g, my synthetic data, or is it a sample from the real world, a real photo of flamingos?

This is just a binary softmax classifier. I'm going to try to maximize the probability I place on real and minimize the probability I place on fake, and that will be a classifier of real versus fake.

And g tries to fool d by trying to minimize the accuracy that d will get on fake, synthetic images. So g just minimizes the exact same objective that d was maximizing. So d is going to try to place, basically, low probability on this being a real image. And g will simply try to place-- cause d to place high probability on this being a real image. So it'll try to update its parameters to make an image that causes d to get a low score on that, on that synthetic photo.

So this is a min-max game. It's an adversarial game. But think of it more like a student and a teacher. d is the teacher, and g is the student.

The teacher is saying, hey, you need to make a realistic painting, and you didn't get it quite right. The shadows are wrong. So g will update its parameters to make realistic shadows. And then d will say, oh, that looks indistinguishable from what I consider realistic art, so therefore, you've done well.

So that's a GAN. I think-- this is just a summary figure of it. I should stop there. And we'll continue with two more lectures on generative models next week.