Lesson 19: Deep Learning Foundations to Stable Diffusion, 2022
https://youtu.be/ItyO8s48zdc

00:00.000 --> 00:01.000
JEREMY: Okay.

00:01.000 --> 00:02.580
Hi everybody.

00:02.580 --> 00:08.140
And this is Lesson 19 with extremely special guests, Tanishq and Johno.

00:08.140 --> 00:09.140
Hi guys.

00:09.140 --> 00:10.140
How are you?

00:10.140 --> 00:11.140
TANISHQ: Hello.

00:11.140 --> 00:12.140
JOHNO: Hey Jeremy.

00:12.140 --> 00:13.660
Good to be here.

00:13.660 --> 00:23.220
JEREMY: And it's New Year's Eve, 2022, finishing off 2022 with a bang, or at least a really cool

00:23.220 --> 00:25.960
lesson.

00:25.960 --> 00:33.140
And most of this lesson is going to be Tanishq and Johno, but I'm going to start with a quick

00:33.140 --> 00:39.860
update from the last lesson.

00:39.860 --> 00:49.420
What I wanted to show you is that Christopher Thomas on the forum, what I want to show you

00:49.420 --> 00:58.980
is that Christopher Thomas on the forum came up with a better winning result for our challenge,

00:58.980 --> 01:04.580
the Fashion-MNIST challenge, which we are tracking here.

01:04.580 --> 01:11.260
And be sure to check out this forum thread for the latest results.

01:11.260 --> 01:17.960
And he found that he was able to get better results with Dropout.

01:17.960 --> 01:24.340
Then Piotr on the forum noticed I had a bug in my code.

01:24.340 --> 01:30.100
And the bug in my code for ResNets, actually I won't show you, I'll just tell you, is that

01:30.100 --> 01:34.460
in the ResBlock, I was not passing along the BatchNorm parameter.

01:34.460 --> 01:38.140
And as a result, all the results I had were without BatchNorm.

01:38.140 --> 01:45.020
So then when I fixed BatchNorm and added Dropout at Christopher's suggestion, I got

01:45.020 --> 01:46.820
better results still.

01:46.820 --> 01:51.980
And then Christopher came up with a better Dropout and got better results still for 50

01:51.980 --> 01:52.980
epochs.

01:52.980 --> 01:59.660
So let me show you the 93.2 for 5 epochs improvement.

01:59.660 --> 02:04.820
I won't show the change to BatchNorm because that's actually, that'll just be in the repo

02:04.820 --> 02:05.820
now.

02:05.820 --> 02:08.880

So the BatchNorm is already fixed.

02:08.880 --> 02:13.740
So I'm going to tell you about what Dropout is and then show that to you.

02:13.740 --> 02:22.460
So Dropout is a simple but powerful idea where what we do with some particular probability,

02:22.460 --> 02:30.100
so here that's a probability of 0.1, we randomly delete some activations.

02:30.100 --> 02:36.100
And when I say delete, what I actually mean is we change them to zero.

02:36.100 --> 02:45.320
So one easy way to do this is to create a binomial distribution object where the probabilities

02:45.320 --> 02:50.460
are 1-p and then sample from that.

02:50.460 --> 02:56.660
And that will give you a 0.1 probability.

02:56.660 --> 02:58.660
So in this case, oh, this is perfect.

02:58.660 --> 03:00.180
I have exactly one 0.

03:00.180 --> 03:03.280
Of course, randomly, that's not always going to be the case.

03:03.280 --> 03:08.340
But since I asked for 10 samples and 0.1 of the time it should be zero, I so happened

03:08.340 --> 03:11.060
to get, yeah, exactly one of them.

03:11.060 --> 03:21.380
And so if we took a tensor like this and multiplied it by our activations, that will set about

03:21.380 --> 03:25.660
a 10th of them to zero because multiplying by zero gives you zero.

03:25.660 --> 03:28.420
So here's a Dropout class.

03:28.420 --> 03:33.440
So you pass it and you say what probability of Dropout there is, store it away.

03:33.440 --> 03:37.900
Now we're only going to do this during training time.

03:37.900 --> 03:43.420
So at evaluation time, we're not going to randomly delete activations.

03:43.420 --> 03:49.220
But during training time, we will create our binomial distribution object.

03:49.220 --> 03:52.220
We will pass in the 1-p probability.

03:52.220 --> 03:57.020
And then you say, how many binomial trials do you want to run?

03:57.020 --> 04:01.060
So how many coin tosses or dice rolls or whatever each time?

04:01.060 --> 04:02.520
And so it's just one.

04:02.520 --> 04:04.020
And this is a cool little trick.

04:04.020 --> 04:10.420
If you put that one onto your accelerator, you know, GPU or MPS or whatever, it's actually

04:10.420 --> 04:13.740
going to create a binomial distribution that runs on the GPU.

04:13.740 --> 04:18.820
That's a really cool trick that not many people know about.

04:18.820 --> 04:25.680
And so then if I sample and I make a sample exactly the same size as my input, then that's

04:25.680 --> 04:32.260
going to give me a bunch of ones and zeros and a tensor, the same size as my activations.

04:32.260 --> 04:38.400
And then another cool trick is this is going to result in activations that are on average

04:38.400 --> 04:41.400
about one tenth smaller.

04:41.400 --> 04:53.300
So if I multiply by 1/(1-0.9), so multiply this case by that, then that's going

04:53.300 --> 05:02.340
to scale up my to undo that difference.

05:02.340 --> 05:03.340
JOHNO: Jeremy.

05:03.340 --> 05:04.340
JEREMY: Yeah.

05:04.340 --> 05:10.060
JOHNO: In the line above where you have probs equals 1-p, should that be 1-self.p

05:10.060 --> 05:12.580
JEREMY: Oh, it absolutely should.

05:12.580 --> 05:15.940
Thank you very much, Johno.

05:15.940 --> 05:21.080
Not that it matters too much because, yeah, you can always just use nn.Dropout at this

05:21.080 --> 05:24.500
point and I only have to use 0.1, which is why I didn't even see that.

05:24.500 --> 05:27.740
So as you can see, I'm not even bothering to export this because I'm just showing how

05:27.740 --> 05:35.620
to repeat what's already available in PyTorch.

05:35.620 --> 05:36.860
So yeah, thanks, Johno.

05:36.860 --> 05:39.260

That's a good fix.

05:39.260 --> 05:45.840
Yeah, so if we're in evaluation mode, it's just going to return the original.

05:45.840 --> 05:53.420
If p=0, then these are all going to be just ones anyway.

05:53.420 --> 05:57.940
So we'll be multiplying by 1 divided by 1, so there's nothing to change.

05:57.940 --> 06:01.100
So with p of 0, it does nothing in effect.

06:01.100 --> 06:05.200
Yeah, and otherwise it's going to kind of zero out some of our activations.

06:05.200 --> 06:11.180
So we can, a pretty common place to add dropout is before your last linear layer.

06:11.180 --> 06:15.700
So that's what I've done here.

06:15.700 --> 06:24.160
So yeah, if I run the exact same epochs, I get 93.2, which is a very slight improvement.

06:24.160 --> 06:33.880
And so the reason for that is that it's not going to be able to kind of memorize the data

06:33.880 --> 06:39.380
or the activations, you know, because there's a little bit of randomness.

06:39.380 --> 06:45.480
So it's going to force it to try to identify just the actual underlying differences.

06:45.480 --> 06:47.480
There's a lot of different ways of thinking about this.

06:47.480 --> 06:50.400
You can almost think of it as a bagging thing, a bit like a random forest.

06:50.400 --> 06:55.320
You know, it's each time it's giving a slightly different kind of random subset.

06:55.320 --> 07:03.100
Yeah, but that's what it does.

07:03.100 --> 07:08.000
I also added a Dropout2d layer right at the start, which is not particularly common.

07:08.000 --> 07:10.300
I was just kind of like showing it.

07:10.300 --> 07:14.880
This is also how Christopher Thomas's idea tried it as well, although he didn't use Dropout2d.

07:14.880 --> 07:17.960
What's the difference between Dropout2d and Dropout?

07:17.960 --> 07:21.800
So this is actually something I'd like you to do to implement yourself as an exercise,

07:21.800 --> 07:24.400
is to implement Dropout2d.

07:24.400 --> 07:35.160
The difference is that with Dropout2d, rather than using x.size() as our tensor of ones and

07:35.160 --> 07:40.420
zeros, so in other words, potentially dropping out every single batch, every single channel,

07:40.420 --> 07:42.760
every single x, y independently.

07:42.760 --> 07:53.080
Instead, we want to drop out an entire kind of grid area, all of the channels together.

07:53.080 --> 07:57.660
So if any of them are zero, then they're all zero.

07:57.660 --> 08:03.440
So you can look up the docs for Dropout2d for more details about exactly what that looks

08:03.440 --> 08:04.440
like.

08:04.440 --> 08:11.440
But yeah, so the exercise is to try and implement that from scratch and come up with a way to

08:11.440 --> 08:12.660
test it.

08:12.660 --> 08:17.600
So like actually check that it's working correctly, because it's a very easy thing to think that

08:17.600 --> 08:23.540
it's working and then realize it's not.

08:23.540 --> 08:31.180
So then, yeah, Christopher Thomas actually found that if you remove this entirely and

08:31.180 --> 08:35.960
only keep this, then you end up with a better results for 50 epochs.

08:35.960 --> 08:39.040
And so he's the first to break 95%

08:39.040 --> 08:47.200
So I feel like we should insert some kind of animation or trumpet sounds or something

08:47.200 --> 08:48.200
at this point.

08:48.200 --> 08:55.000
I'm not sure if I'm clever enough to do that in the video editor, but I'll see how I go.

08:55.000 --> 08:56.000
Hooray!

08:56.000 --> 08:57.000
Okay.

08:57.000 --> 09:00.160
So that's about it for me.

09:00.160 --> 09:04.360
Did you guys have any other things to add about Dropout, how to understand it or what

09:04.360 --> 09:06.000
it does or interesting things?

09:06.000 --> 09:07.960

Oh, I did have one more thing before.

09:07.960 --> 09:12.120
But you go ahead if you've got anything to mention.

09:12.120 --> 09:17.260
JOHNO: So I was going to ask just because I think the standard is to set it, like remove the

09:17.260 --> 09:18.760
dropout before you do inference.

09:18.760 --> 09:23.520
But I was wondering if there's anyone you know of, or if it works to use it for some

09:23.520 --> 09:25.120
sort of test time augmentation.

09:25.120 --> 09:26.520
JEREMY: Oh, dude!

09:26.520 --> 09:27.520
Thank you.

09:27.520 --> 09:28.760
Because I wrote a callback for that.

09:28.760 --> 09:35.200
Did you see this or are you just like (JOHNO: no), okay, just a test time dropout callback.

09:35.200 --> 09:36.400
Nice.

09:36.400 --> 09:48.120
So yeah, before_epoch, if you're a member in Learner, we put it into training mode.

09:48.120 --> 09:53.120
Which actually what it does is it puts every individual layer into training mode.

09:53.120 --> 09:58.760
So that's why for the module itself, we can check whether that module's in training mode.

09:58.760 --> 10:03.980
So what we can actually do is after that's happened, we can then go back in this callback

10:03.980 --> 10:20.520
and apply a lambda that says if this is a Dropout, then… wait, this is, yeah, then put

10:20.520 --> 10:26.240
it in training mode all the time, including at evaluation.

10:26.240 --> 10:33.960
And so then you can run it multiple times, just like we did for TTA, but with this callback.

10:33.960 --> 10:45.760
Now that's very unlikely to give you a better result because it's not kind of showing it

10:45.760 --> 10:51.100
different versions or anything like that, like TTA does that are kind of meant to be

10:51.100 --> 10:53.200
the same.

10:53.200 --> 11:04.420
But what it does do is it gives you a sense of how confident it is.

11:04.420 --> 11:11.020
If it kind of has no idea, then that little bit of dropout's quite often going to lead

11:11.020 --> 11:12.720
to different predictions.

11:12.720 --> 11:17.000
So this is a way of kind of doing some kind of confidence measure.

11:17.000 --> 11:22.180
You'd have to calibrate it by, kind of, looking at things that it should be confident about

11:22.180 --> 11:30.080
and not confident about and seeing how that dropout, test time dropout changes.

11:30.080 --> 11:35.080
But the basic idea, it's been used in medical models before.

11:35.080 --> 11:42.160
I wouldn't say it's totally popular, which is why I didn't even bother to show it being

11:42.160 --> 11:47.200
used, but I just want to add it here because I think it's an interesting idea and maybe

11:47.200 --> 11:54.940
could be more used than it is, or at least more studied than it has been.

11:54.940 --> 12:00.800
A lot of stuff that gets used in the medical world is less well known out in the rest of

12:00.800 --> 12:01.800
the world.

12:01.800 --> 12:03.800
So maybe that's part of the problem.

12:03.800 --> 12:04.800
Cool.

12:04.800 --> 12:07.880
All right.

12:07.880 --> 12:13.920
So I will stop my sharing and we're going to switch to Tanishq, who's going to do something

12:13.920 --> 12:21.680
much more exciting, which is to show that we are now at a point where we can do DDPM

12:21.680 --> 12:26.160
from scratch or at least everything except the model.

12:26.160 --> 12:32.880
And so to remind you, DDPM doesn't have the latent VAE thing and we're not going to do

12:32.880 --> 12:33.880
conditional.

12:33.880 --> 12:40.040
So it's not going to be like, we're not going to get to tell it what to draw.

12:40.040 --> 12:46.080
And the U-Net model itself is the one bit we're not going to do today.

12:46.080 --> 12:49.080
We're going to do that next lesson.

12:49.080 --> 12:56.440

But, but other than the U-Net, it's going to be unconditional DDPM from scratch.

12:56.440 --> 12:59.360
So Tanishq, take it away.

12:59.360 --> 13:00.360
Okay.

13:00.360 --> 13:03.920
Hi, welcome back.

13:03.920 --> 13:05.660
Sorry for the slight continuity problem.

13:05.660 --> 13:07.840
You may notice people look a little bit different.

13:07.840 --> 13:09.920
That's because we had some Zoom issues.

13:09.920 --> 13:14.640
So we have a couple of days have passed and we're back again.

13:14.640 --> 13:18.840
And then Johno over recorded his bit before we do Tanishq's bit, and then we're going

13:18.840 --> 13:20.400
to post them in backwards.

13:20.400 --> 13:25.040
So hopefully there's not too many confusing continuity problems as a result and it all

13:25.040 --> 13:36.100
goes smoothly, but it's time to turn it over to Tanishq to talk about DDPM.

13:36.100 --> 13:42.720
TANISHQ: So we've reached the point where we have this miniai framework and I guess it's time to

13:42.720 --> 13:48.440
now start using it to build more, I guess, sophisticated models.

13:48.440 --> 13:54.600
And as we'll see here, we can start putting together a diffusion model from scratch using

13:54.600 --> 13:59.060
the miniai library, and we'll see how it makes our life a lot easier.

13:59.060 --> 14:04.960
And also it'd be very nice to see how the equations in the papers correspond to the

14:04.960 --> 14:05.960
code.

14:05.960 --> 14:13.800
I have here, of course, the notebook that we'll be watching from.

14:13.800 --> 14:21.000
The paper, which we have the diffusion model paper, “Denoising Diffusion Probabilistic

14:21.000 --> 14:25.280
Models”, which is the paper that was published in 2020.

14:25.280 --> 14:31.560
It was one of the original diffusion model papers that set off the entire trend of diffusion

14:31.560 --> 14:38.200
models and is a good starting point as we delve into this topic further.

14:38.200 --> 14:45.880
And also I have some diagrams and drawings that I will also show later on.

14:45.880 --> 14:53.240
But yeah, basically let's just get started with the code here and of course the paper.

14:53.240 --> 14:58.820
So just to provide some context with this paper, this paper was published from this

14:58.820 --> 15:04.840
group in UC Berkeley, I think a few of them have gone on now to work at Google.

15:04.840 --> 15:10.500
And this is Pieter Abbeel, he has a big lab at UC Berkeley.

15:10.500 --> 15:17.480
And so diffusion models were actually originally introduced in 2015, but this paper in 2020

15:17.480 --> 15:22.160

greatly simplified the diffusion models and made it a lot easier to work with and got

15:22.160 --> 15:26.720
these amazing results as you can see here when they trained on faces and in this case

15:26.720 --> 15:36.200
CIFAR-10 and this really was very, kind of a big leap in terms of the progress of diffusion

15:36.200 --> 15:37.980
models.

15:37.980 --> 15:42.620
And so just to kind of briefly provide, I guess, kind of an overview.

15:42.620 --> 15:53.760
JEREMY: If I could just quickly step just mention something, which is, when we started this course, we

15:53.760 --> 16:00.680
talked a bit about how perhaps the diffusion part of diffusion models is not actually all

16:00.680 --> 16:01.680
that.

16:01.680 --> 16:05.080
Everybody's been talking about diffusion models because that's, particularly because that's

16:05.080 --> 16:07.960
the open source thing we have that works really well.

16:07.960 --> 16:15.080
But this week, actually a model that appears to be quite a lot better than Stable Diffusion

16:15.080 --> 16:20.360
was released that doesn't use diffusion at all.

16:20.360 --> 16:28.480
Having said that, the basic ideas, like most of the stuff that Tanishq talks about today,

16:28.480 --> 16:35.200
will still appear in some kind of form, but a lot of the details will be different.

16:35.200 --> 16:41.940
But strictly speaking, actually, I don't even know if we've got a word anymore for the kind

16:41.940 --> 16:46.780
of like modern generative model things we're doing.

16:46.780 --> 16:51.640
So in some ways, when we're talking about diffusion models, you should maybe replace

16:51.640 --> 16:55.800
it in your head with some other word, which is more general and includes this paper that

16:55.800 --> 16:57.360
Tanishq is looking at here.

16:57.360 --> 16:59.220
JOHNO: Iterative Refinement, perhaps?

16:59.220 --> 17:00.220
That's what I'd like.

17:00.220 --> 17:02.220
JEREMY: Yeah, that's not bad, iterative refinement.

17:02.220 --> 17:06.760
I'm sure by the time people watch this video, probably, you know, somebody will have decided

17:06.760 --> 17:08.120
on something.

17:08.120 --> 17:10.880
We will keep our course website up to date.

17:10.880 --> 17:11.880
TANISHQ: Yeah.

17:11.880 --> 17:12.880
Yeah.

17:12.880 --> 17:17.280
This is the paper that Jeremy was talking about and yeah, every week there seems to

17:17.280 --> 17:21.320
be another state of the art model.

17:21.320 --> 17:26.600

But yeah, like Jeremy said, a lot of the principles are the same, but the details can be different

17:26.600 --> 17:30.080
for each paper.

17:30.080 --> 17:36.360
And I just want to again, also, like Jeremy was saying, zoom back a little bit and talk

17:36.360 --> 17:42.680
a little bit about what, just to provide a review of what we're trying to do here.

17:42.680 --> 17:45.720
So let me just right next to him here.

17:45.720 --> 17:46.720
Yeah.

17:46.720 --> 17:56.520
So with this task, we were trying to, in this case, we're trying to do image generation.

17:56.520 --> 18:01.080
Of course, it could be other forms of generation, like text generation or whatever.

18:01.080 --> 18:07.360
And the general idea is that of course we have some data points.

18:07.360 --> 18:11.160
In this case, we have some images of dogs and we want to produce more like the data

18:11.160 --> 18:12.400
points that we're given.

18:12.400 --> 18:17.120
So in this case, maybe the dog image generation or something like this.

18:17.120 --> 18:24.840
And so the overall idea that a lot of these approaches take for some sort of generative

18:24.840 --> 18:28.040
modeling task is they try to...

18:28.040 --> 18:30.600
Not over there, I'm going to mark here.

18:30.600 --> 18:31.600
They try to...

18:31.600 --> 18:33.600
Oops, what happened here?

18:33.600 --> 18:35.600
Maybe it might...

18:35.600 --> 18:36.600
Yeah.

18:36.600 --> 18:40.880
So let me use it in a bit.

18:40.880 --> 18:46.240
p(x), which is basically the likelihood...

18:46.240 --> 18:50.840
What's going to happen here?

18:50.840 --> 18:56.240
Likelihood of data point x.

18:56.240 --> 19:00.280
So let's say x is some image.

19:00.280 --> 19:07.280
Then p(x) tells us what is the probability that you would see that image in real life.

19:07.280 --> 19:13.840
And we can take a simpler example, which may be easier to think about, of a one-dimensional

19:13.840 --> 19:17.080
data point like height, for example.

19:17.080 --> 19:20.760
And if we were to look at height, of course we know we have a data distribution that's

19:20.760 --> 19:22.880
kind of a bell curve.

19:22.880 --> 19:29.120
And you have maybe some mean height, which is something like 5'9", 5'10".

19:29.120 --> 19:34.920
I guess 5'10", or something like that, or 5'9", whatever.

19:34.920 --> 19:39.720
And then of course we have some more unlikely points, but that is still possible.

19:39.720 --> 19:45.000
Like for example, we have 7'8", or we have something that's maybe not as likely, which

19:45.000 --> 19:47.000
is like 3', or something like this.

19:47.000 --> 19:53.680
JEREMY: So here's the X axis is height, and the Y axis is the probability of some random person

19:53.680 --> 19:56.680
you meet being that tall.

19:56.680 --> 19:57.760
TANISHQ: Exactly.

19:57.760 --> 20:02.160
So this is basically the probability.

20:02.160 --> 20:06.940
And so of course you have this sort of peak, which is where you have higher probability.

20:06.940 --> 20:11.220
And so those are the sorts of values that you would see more often.

20:11.220 --> 20:15.760
So this is what we would call our p(x).

20:15.760 --> 20:23.920
And the important part about p(x) is that you can use this now to sample new values

20:23.920 --> 20:27.960
if you know what p(x) is, or if you have some sort of information about p(x).

20:27.960 --> 20:33.240
So for example, here you can think of, if you were to say, maybe have some, let's say

20:33.240 --> 20:36.880
you have some game and you have some human characters in the game, and you just want

20:36.880 --> 20:43.560
to randomly generate a height for this human character, you wouldn't want to of course

20:43.560 --> 20:47.620
select a random height between 3 and 7, that's kind of uniformly distributed.

20:47.620 --> 20:56.160
You would instead maybe want to have the height dependent on this sort of function, where

20:56.160 --> 21:01.860
you would more likely sample values in the middle and less likely sample these sorts

21:01.860 --> 21:03.160
of extreme points.

21:03.160 --> 21:05.600
So it's dependent on this function p(x).

21:05.600 --> 21:12.840
So having some information about p(x) will allow you to sample more data points.

21:12.840 --> 21:16.740
And so that's kind of the overall goal of generative modeling is to get some information

21:16.740 --> 21:23.520
about p(x) that then allows us to sample new points and create new generations.

21:23.520 --> 21:31.200
So that's kind of a high level kind of description of what we're trying to do when we're doing

21:31.200 --> 21:32.640
generative modeling.

21:32.640 --> 21:36.040
And of course there are many different approaches.

21:36.040 --> 21:42.440
We have our famous GANs, which used to be the common method back in the day before diffusion

21:42.440 --> 21:43.440

models.

21:43.440 --> 21:47.720
We have VAEs, which I think we'll probably talk a little bit more about that later as

21:47.720 --> 21:48.720
well.

21:48.720 --> 21:50.840
JEREMY: We'll be talking about both of those techniques later.

21:50.840 --> 21:51.840
Yeah.

21:51.840 --> 21:52.840
TANISHQ: Yeah.

21:52.840 --> 21:53.840
So there are many different other techniques.

21:53.840 --> 21:57.160
There are also some niche techniques that are out there as well.

21:57.160 --> 22:02.560
But of course now the popular one is are these diffusion models or as we talked about, maybe

22:02.560 --> 22:11.240
a better term might be, iterative refinement or whatever the term ends to be.

22:11.240 --> 22:13.360
But yeah, so there are many different techniques.

22:13.360 --> 22:23.040
And yeah, so this is kind of the general diagram that shows what diffusion models are.

22:23.040 --> 22:27.720
And if we can look at the paper here, which let's pull up the paper.

22:27.720 --> 22:32.800
Yeah, you see here, this is the sort of, they call it directed graphical model.

22:32.800 --> 22:34.080
It's a very complicated term.

22:34.080 --> 22:40.600
It's just kind of showing what's going on in this process.

22:40.600 --> 22:46.040
There's a lot of complicated math here, but we'll highlight some of the key variables

22:46.040 --> 22:48.520
and equations here.

22:48.520 --> 22:54.200
So basically the idea is that, okay, so let's see here.

22:54.200 --> 22:58.180
This is an image that we want to generate, right?

22:58.180 --> 23:04.760
And so x0 is basically, these are actually the samples that we want.

23:04.760 --> 23:09.320
So we want to, x0 is what we want to generate.

23:09.320 --> 23:15.680
And these would be, yeah, these are images.

23:15.680 --> 23:17.400
And we start out with pure noise.

23:17.400 --> 23:26.680
So that's what xt, pure noise.

23:26.680 --> 23:30.720
And the whole idea is that we have two processes.

23:30.720 --> 23:37.960
We have this process where we're going from pure noise to our image.

23:37.960 --> 23:40.440
And we have this process from our image to pure noise.

23:40.440 --> 23:45.560
So the process where we're going from our image to pure noise, this is called the forward

23:45.560 --> 23:46.560
process.

23:46.560 --> 23:53.080
Forward, sorry, my typing is still, my handwriting is not so good in it.

23:53.080 --> 23:56.600
So hopefully it's clear enough.

23:56.600 --> 23:57.600
Let me know if it's not.

23:57.600 --> 24:02.500
So we have the forward process, which is mostly just used for training.

24:02.500 --> 24:07.520
Then we also have our reverse process.

24:07.520 --> 24:12.800
This is the reverse process, which I will write up here.

24:12.800 --> 24:15.960
Reverse process.

24:15.960 --> 24:21.480
JEREMY: So this is a bit of a summary, I guess, of what you and Wasim talked about in Lesson

24:21.480 --> 24:26.400
9b.

24:26.400 --> 24:31.120
TANISHQ: And just, it's just mostly to highlight now what are the different variables as we look

24:31.120 --> 24:35.800
at the code and see the different variables in the code.

24:35.800 --> 24:41.040
JEREMY: Okay, so we'll be focusing today on the code, but the code will be referring to things by

24:41.040 --> 24:48.080
name and those names won't make sense very much unless we see what they're used for in

24:48.080 --> 24:49.080
the math.

24:49.080 --> 24:50.080
Okay.

24:50.080 --> 24:51.080
TANISHQ: Yeah.

24:51.080 --> 24:52.320
And I won't dive too much into the math.

24:52.320 --> 24:58.940
I just want to focus on these sorts of variables and equations that we see in the code.

24:58.940 --> 25:04.920
So basically the general idea is that we do these in multiple different steps.

25:04.920 --> 25:13.160
We have here from time step 0 all the way to time step uppercase T. And so there's some

25:13.160 --> 25:17.600
fixed number of steps, but then we have this intermediate process where we're going from

25:17.600 --> 25:22.440
some particular time step.

25:22.440 --> 25:33.560
We have this time step lowercase t, which is some noisy image.

25:33.560 --> 25:36.780
And yes, we're transitioning between these two different noisy images.

25:36.780 --> 25:39.880
So we have this, what is sometimes called the transition.

25:39.880 --> 25:40.880
We have this one here.

25:40.880 --> 25:45.640
This is like something that's called the transition kernel or yeah, whatever it is, it basically

25:45.640 --> 25:50.600
is just telling us, you know, how do we go from, you know, one, in this case, we're going

25:50.600 --> 25:54.820
from a less noisy image to a more noisy image and then going backwards, it's going from

25:54.820 --> 25:57.260
a more noisy image to a less noisy image.

25:57.260 --> 25:59.040
So let's look at the equations.

25:59.040 --> 26:02.480
JEREMY: So the forward direction is driven really easily to make it something more noisy.

26:02.480 --> 26:04.360
You just add a bit more noise to it.

26:04.360 --> 26:09.160
And the reverse direction is incredibly difficult, which is to particularly to go from the far

26:09.160 --> 26:15.300
left to the far right is strictly speaking impossible because none of that person's face

26:15.300 --> 26:17.040
exists anymore.

26:17.040 --> 26:21.320
That somewhere in between you could certainly go from something that's partially noisy to

26:21.320 --> 26:26.400
less noisy by a learned model.

26:26.400 --> 26:28.220
TANISHQ: Exactly.

26:28.220 --> 26:31.240
And that's like one of the little things I've done right now in terms of, you know, in terms

26:31.240 --> 26:33.440
of I guess the symbols and the math.

26:33.440 --> 26:38.480
So yeah, basically I'm just trying to pull out the, just to write down the equations

26:38.480 --> 26:39.480
here.

26:39.480 --> 26:45.040

So we have, let me zoom in a bit.

26:45.040 --> 26:51.120
So we have our two, let's see here.

26:51.120 --> 26:58.360
Q of xt, x (t minus 1)

26:58.360 --> 27:08.240
Or actually, you know what, maybe it's just better if I just snipped it from here.

27:08.240 --> 27:21.040
So the one that is going from our forward process is this equation here.

27:21.040 --> 27:24.080
So I'll just make that a little smaller for you guys.

27:24.080 --> 27:26.040
So right there.

27:26.040 --> 27:37.200
So that is going, and basically to explain, we have this sort of script, a little bit

27:37.200 --> 27:41.380
of a, maybe a little bit confusing notation here, but basically this is referring to a

27:41.380 --> 27:45.320
normal distribution or a Gaussian distribution.

27:45.320 --> 27:49.560
And this is just saying, okay, this is a Gaussian distribution that's describing this particular

27:49.560 --> 27:51.140
variable.

27:51.140 --> 27:58.720
So it's just saying, okay, N is our normal or Gaussian distribution, and it's representing

27:58.720 --> 28:10.040
this variable x of t, or x, sorry, xt. And then we have here is the mean, and this is

28:10.040 --> 28:14.920
the variance.

28:14.920 --> 28:20.920
So just to again, clarify, I think we've talked about this before as well, but this is a,

28:20.920 --> 28:29.800
this is of course a bad drawing of a Gaussian, but our mean is just, our mean is just this,

28:29.800 --> 28:33.280
the middle point here is the mean, and the variance just kind of describes the sort of

28:33.280 --> 28:37.780
spread of the Gaussian distribution.

28:37.780 --> 28:43.840
So if you think about it a little further, you have this beta, which is one of the important

28:43.840 --> 28:49.920
variables that kind of describes the diffusion process, beta-t

28:49.920 --> 28:53.800
So you'll see the beta-t in the code.

28:53.800 --> 28:59.060
And basically beta-t increases as t increases.

28:59.060 --> 29:06.220
So basically your beta-t will be greater than your beta-(t minus 1).

29:06.220 --> 29:12.000
So if you think about that a little bit more carefully, you can see that, okay, so at t-1,

29:12.000 --> 29:18.360
at this time point here, and then you're going to the next time point, you're

29:18.360 --> 29:20.120
going to increase your beta-t.

29:20.120 --> 29:25.360
So you're increasing the variance, but then you have this 1 - beta-t and take the

29:25.360 --> 29:29.440
square root of that and multiply it by x(t minus 1)

29:29.440 --> 29:35.320
So as your t is increasing, this term actually decreases.

29:35.320 --> 29:39.880
So your mean is actually decreasing and you're getting less of the original image

29:39.880 --> 29:43.880
Because the original image is going to be part of x(t minus 1)

29:43.880 --> 29:51.600
JEREMY: And just to let you know, Tanishq, we can't see your pointer.

29:51.600 --> 29:56.680
So if you want to point at things, you would need to highlight them or something.

29:56.680 --> 30:00.800
TANISHQ: So yeah, I'll just, let's see.

30:00.800 --> 30:01.800
Yeah.

30:01.800 --> 30:06.480
Basically, I haven't pointed anything in specific.

30:06.480 --> 30:14.120
I was just saying that, yeah, basically if we have our xt here, as the time step

30:14.120 --> 30:20.680
increases, you're getting less contribution from your x(t minus 1)

30:20.680 --> 30:24.240
And so that means your mean is going towards zero.

30:24.240 --> 30:28.480
And so you've got to have a mean of 0 and the variance keeps increasing and basically

30:28.480 --> 30:32.500
you just have a Gaussian distribution and you lose any contribution from the original

30:32.500 --> 30:34.840
image as your time step increases.

30:34.840 --> 30:41.320
So that's why when we start out from x0 and go all the way to our xt here,

30:41.320 --> 30:42.680

this becomes pure noise.

30:42.680 --> 30:45.780
It's because we're doing this iterative process where we keep adding noise.

30:45.780 --> 30:54.760
We lose that contribution from the original image and that leads to the image having pure

30:54.760 --> 30:57.720
noise at the end of the process.

30:57.720 --> 31:08.040
JEREMY: Something I find useful here is to consider one extreme, which is to consider x1.

31:08.040 --> 31:15.280
So at x1, the mean is going to be root 1 - beta-t times x0.

31:15.280 --> 31:18.800
The reason that's interesting is x0 is the original image.

31:18.800 --> 31:27.140
So we're taking the original image and at this point 1 - beta-t will be pretty

31:27.140 --> 31:28.280
close to 1.

31:28.280 --> 31:34.120
So at x1, we're going to have something that's the mean is very close to the image

31:34.120 --> 31:36.540
and the variance will be very small.

31:36.540 --> 31:41.680
And so that's why we will have a image that just has a tiny bit of noise.

31:41.680 --> 31:44.360
TANISHQ: Right, right.

31:44.360 --> 31:48.220
And then another thing that sometimes it's easier to write out is sometimes you can write

31:48.220 --> 31:58.520
out, in this case, you can write out q(xt) directly because these are all independent

31:58.520 --> 32:02.920
in terms of q(xt) is only dependent on x(t minus 1)

32:02.920 --> 32:05.960
And then x(t minus 1) is only dependent on x(t minus 2)

32:05.960 --> 32:10.580
And each of these steps are independent.

32:10.580 --> 32:18.160
So based on the different laws of probability, you can get your q(xt) in close form.

32:18.160 --> 32:20.840
So, yeah, that's what's shown here.

32:20.840 --> 32:22.440
q xt given the original image.

32:22.440 --> 32:29.920
So this is also another way of kind of seeing this more clearly where you can see that.

32:29.920 --> 32:34.760
Anyways, so I'm going back here.

32:34.760 --> 32:39.600
So this is another way to see here more directly.

32:39.600 --> 32:45.240
So this is, of course, our clean image.

32:45.240 --> 32:54.520
And this is our clear, our noisy image.

32:54.520 --> 33:02.040
And so you can also see again, now alpha bar t is dependent on beta t.

33:02.040 --> 33:05.680
Basically it's like one minus the cumulative.

33:05.680 --> 33:09.280
JEREMY: I mean, we'll see the code for it, I guess.

33:09.280 --> 33:10.280
So maybe.

33:10.280 --> 33:11.280
TANISHQ: Yes, yes.

33:11.280 --> 33:15.160
So it might be clear to see that this is alpha bar t or something like this.

33:15.160 --> 33:26.200
But basically, basically the idea is that alpha bar t is going to be, again, less.

33:26.200 --> 33:30.540
This is what is going to be less than alpha bar t minus one.

33:30.540 --> 33:33.660
So basically alpha, this keeps decreasing, right?

33:33.660 --> 33:36.640
This decreases as time step increases.

33:36.640 --> 33:40.640
And on the other hand, this is going to be increasing as time step increases.

33:40.640 --> 33:45.720
But again, you can see the contribution from the original image decreases as time step

33:45.720 --> 33:51.800
increases while the noise, as shown by the variance, is increasing while the time step

33:51.800 --> 33:52.800
is increasing.

33:52.800 --> 33:56.840
Anyway, so that hopefully clarifies the forward process.

33:56.840 --> 34:04.200
And then the reverse process is basically a neural network, as Jeremy had mentioned.

34:04.200 --> 34:11.560
And yeah, screenshot this.

34:11.560 --> 34:19.400
That's our reverse process.

34:19.400 --> 34:25.560

And basically the idea is, well, this is a neural network and this is also a neural network.

34:25.560 --> 34:29.800
Neural network.

34:29.800 --> 34:33.120
And we learn it during the training of the model.

34:33.120 --> 34:40.480
But the nice thing about this particular diffusion model paper that made it so simple was actually,

34:40.480 --> 34:47.480
we completely ignored this and actually set it to constants just based on, you know, big

34:47.480 --> 34:48.480
numbers.

34:48.480 --> 34:49.480
JEREMY: We can't see what you're pointing at.

34:49.480 --> 34:52.440
So I think it's important to mention what this is here.

34:52.440 --> 34:53.440
TANISHQ: This term here.

34:53.440 --> 35:07.600
So this one, we just kind of ignore and it's just a constant dependent on beta-t.

35:07.600 --> 35:12.240
So you only have one neural network that you need to train, which is basically referring

35:12.240 --> 35:14.320
to this mean.

35:14.320 --> 35:21.560
And when the nice thing about this diffusion model process is that it also re-paraphrases

35:21.560 --> 35:27.520
the mean into this easier form where you do a lot of complicated math, which we'll not

35:27.520 --> 35:30.260
get into here.

35:30.260 --> 35:38.720
But basically you get this kind of simplified training objective where, let's see here.

35:38.720 --> 35:42.360
Yeah, you see the simplified training objective.

35:42.360 --> 35:46.080
You instead have this epsilon-theta function.

35:46.080 --> 35:52.600
And let me just screenshot that again.

35:52.600 --> 36:00.240
This is our loss function that we train and we have this epsilon-theta function.

36:00.240 --> 36:02.440
You can see it's a very simple loss function, right?

36:02.440 --> 36:06.020
This is just a, let me just write this down.

36:06.020 --> 36:08.560
This is just an MSE loss.

36:08.560 --> 36:11.760
And we have this epsilon-theta function here.

36:11.760 --> 36:12.760
That is our-

36:12.760 --> 36:16.160
JEREMY: …maybe here we're less mathy, it might not be obvious that it's a simple

36:16.160 --> 36:21.480
thing, because it looks quite complicated to me, but once we see it in code, it'll be

36:21.480 --> 36:22.480
simple.

36:22.480 --> 36:23.480
TANISHQ: Yes, yes.

36:23.480 --> 36:28.800
Basically you're just doing like, and you'll see it in code how simple it is.

36:28.800 --> 36:30.440
But this is like just an MSE loss.

36:30.440 --> 36:35.480
So we've seen MSE loss before, but you'll see how, yeah, this is basically MSE.

36:35.480 --> 36:40.840
So the nice, so just to kind of take a step back again, what is this epsilon-theta?

36:40.840 --> 36:45.840
Because this is like a new thing that like seems a little bit confusing.

36:45.840 --> 36:54.960
Basically epsilon, you can see here, basically, yeah, so this here is saying, this is actually

36:54.960 --> 37:02.320
equivalent to this equation here.

37:02.320 --> 37:03.320
These two are equivalent.

37:03.320 --> 37:10.200
This is just another way of saying that, because basically it's saying, that's xt

37:10.200 --> 37:13.680
So this is giving xt just in a different way.

37:13.680 --> 37:24.540
But epsilon is actually this normal distribution with a mean of 0 and a variance of 1.

37:24.540 --> 37:30.060
And then you have all these scaling terms that changes the mean to be the same as this

37:30.060 --> 37:33.080
equation that we have over here.

37:33.080 --> 37:40.120
So this is our xt. And so what epsilon is, it's actually the noise that we're adding

37:40.120 --> 37:43.520
to our image to make it into a noisy image.

37:43.520 --> 37:47.120

And what this neural network is doing is trying to predict that noise.

37:47.120 --> 37:54.280
So what this is actually doing is this is actually a noise predictor, and it is predicting

37:54.280 --> 37:59.080
the noise in the image.

37:59.080 --> 38:02.920
And why is that important?

38:02.920 --> 38:12.840
Basically the general idea is, if we were to think about our distribution of data, let's

38:12.840 --> 38:17.560
just think about it in a 2D space.

38:17.560 --> 38:26.020
Just here, each data point here represents an image, and they're in this blob area, which

38:26.020 --> 38:27.700
represents a distribution.

38:27.700 --> 38:38.600
So this is in-distribution, and this is out of the distribution.

38:38.600 --> 38:45.360
Out-of-distribution.

38:45.360 --> 38:53.600
And basically the idea is that, okay, if we take an image and we want to generate some

38:53.600 --> 38:58.840
random image, if we were to take a random data point, it would most likely be noisy

38:58.840 --> 38:59.840
images, right?

38:59.840 --> 39:05.120
So if we take some random data point, it would be more, the way to generate random data point,

39:05.120 --> 39:06.800
it's going to be just noise.

39:06.800 --> 39:13.560
But we want to keep adjusting this data point to make it look more like an image from your

39:13.560 --> 39:14.560
distribution.

39:14.560 --> 39:17.560
That's kind of the whole idea of this iterative process that we're doing in our diffusion

39:17.560 --> 39:18.560
model.

39:18.560 --> 39:26.660
So the way to get that information is actually to take images from your dataset and actually

39:26.660 --> 39:28.160
add noise to it.

39:28.160 --> 39:30.520
So that's what we try to do in this process.

39:30.520 --> 39:34.740
So we have an image here and we add noise to it.

39:34.740 --> 39:38.360
And then what we do is we try to plan a neural network to predict the noise.

39:38.360 --> 39:43.260
And by predicting the noise and subtracting it out, we are going back to the distribution.

39:43.260 --> 39:47.920
So adding the noise takes you away from the distribution and then predicting the noise

39:47.920 --> 39:50.100
brings you back to the distribution.

39:50.100 --> 39:59.220
So then if we know at any given point in this space how much noise to remove, that tells

39:59.220 --> 40:08.840
you how to keep going towards the data distribution and get a point that lies within the distribution.

40:08.840 --> 40:10.540
So that's why we have noise prediction.

40:10.540 --> 40:15.280
And that's the importance of doing this noise prediction is to be able to then do this iterative

40:15.280 --> 40:20.220
processor, we can start out at a random point, which would be, for example, pure noise and

40:20.220 --> 40:25.400
keep predicting and removing that noise and walking towards the data distribution.

40:25.400 --> 40:26.400
Okay.

40:26.400 --> 40:27.400
Okay.

40:27.400 --> 40:33.860
So yeah, let's get started with the code.

40:33.860 --> 40:38.980
And so here we of course have our imports and we're going to load our dataset.

40:38.980 --> 40:45.320
We're going to work with our Fashion-MNIST dataset, which is what we've been working with

40:45.320 --> 40:48.720
for a while already.

40:48.720 --> 40:54.280
And yeah, this is just basically the same code that we've seen from before in terms

40:54.280 --> 40:57.900
of loading the dataset.

40:57.900 --> 40:59.360
And then we have our model.

40:59.360 --> 41:01.540
So I've removed the noise from an image.

41:01.540 --> 41:07.220
So our model is going to take in, is it's going to take in the previous image, the noisy

41:07.220 --> 41:10.520
image and predict the noise.

41:10.520 --> 41:13.920
So the shapes of the input and the output are the same.

41:13.920 --> 41:16.280
They're going to be in the shape of an image.

41:16.280 --> 41:23.880
So what we use is we use a U-Net neural network, which takes in kind of an input image.

41:23.880 --> 41:25.800
JEREMY: And we do see your pointer now, by the way.

41:25.800 --> 41:27.800
So feel free to point at things.

41:27.800 --> 41:28.800
TANISHQ: Yeah.

41:28.800 --> 41:32.700
So yeah, it takes in an input image.

41:32.700 --> 41:40.360
And in this case, a U-Net is the purpose, but they can also be used for any sort of image

41:40.360 --> 41:47.000
to image task, where we're going from an input image and then outputting some other image

41:47.000 --> 41:48.000
of some sort.

41:48.000 --> 41:49.000
And we'll talk about...

41:49.000 --> 41:52.200
JEREMY: So this is a new architecture, which we haven't learned about yet, and we will be learning

41:52.200 --> 41:55.000
about in the next lesson.

41:55.000 --> 42:02.840
But broadly speaking, those gray arrows going from left to right are a lot like ResNet,

42:02.840 --> 42:05.840
very much like ResNet skip connections.

42:05.840 --> 42:10.040
But they're being used in a different way.

42:10.040 --> 42:14.800
Everything else is stuff that we've seen before.

42:14.800 --> 42:19.700
So it's basically, we can pretend those don't exist for now.

42:19.700 --> 42:28.680
It's a neural network that the output is the same size or a similar size to the input.

42:28.680 --> 42:35.080
And therefore you can use it to learn how to go from one image to a different image.

42:35.080 --> 42:37.000
TANISHQ: Yeah.

42:37.000 --> 42:40.360
So that's where the U-Net is.

42:40.360 --> 42:45.280
And yeah, like Jeremy said, we'll talk about it more.

42:45.280 --> 42:50.480
The sort of U-Net that are used for diffusion models also tend to have some additional tricks,

42:50.480 --> 42:54.580
which again, we'll talk about them later on as well.

42:54.580 --> 43:03.040
But yeah, for the time being, we will just import a U-Net from the Diffusers library,

43:03.040 --> 43:06.880
which is the Hugging Face library for diffusion models.

43:06.880 --> 43:13.080
So they have a U-Net implementation and we'll just be using that for now.

43:13.080 --> 43:17.280
And so, yeah, of course, JEREMY: strictly speaking, we're cheating at this point because we're

43:17.280 --> 43:21.600
using something we haven't written from scratch, but we're only cheating temporarily because

43:21.600 --> 43:24.280
we will be writing it from scratch.

43:24.280 --> 43:25.280
TANISHQ: Yeah.

43:25.280 --> 43:31.020
And yeah, so, and then of course we're working with one channel images, our Fashion-MNIST

43:31.020 --> 43:32.880
images are one channel images.

43:32.880 --> 43:35.360
So we just have to specify that.

43:35.360 --> 43:41.100
And then of course the channels of the different blocks within the U-Net are also specified.

43:41.100 --> 43:45.100
And then let's go into the training process.

43:45.100 --> 43:55.480
So basically the general idea of course is we want to train with this MSE loss.

43:55.480 --> 44:04.000
What we do is we select a random timestep and then we add noise to our image based on

44:04.000 --> 44:05.000
that timestep.

44:05.000 --> 44:08.900
So of course, if we have a very high timestep, we're adding a lot of noise.

44:08.900 --> 44:16.040
If we have a lower timestep, they were adding very little noise.

44:16.040 --> 44:20.240

So we're going to randomly choose a timestep.

44:20.240 --> 44:24.640
And then, yeah, we add the noise accordingly to the image.

44:24.640 --> 44:29.700
And then we pass the noisy image to a model as well as the timestep.

44:29.700 --> 44:35.400
And we are trying to predict the amount of noise that was in the image and we predict it

44:35.400 --> 44:36.900
with the MSE loss.

44:36.900 --> 44:40.120
So we can see all the...

44:40.120 --> 44:46.340
JEREMY: I have some pictures of some of these variables I could share if that would be useful.

44:46.340 --> 44:47.920
So I have a version.

44:47.920 --> 44:50.960
So I think Tanishq is sharing notebook number 15.

44:50.960 --> 44:52.180
Is that right?

44:52.180 --> 44:55.520
And I've got here notebook number 17.

44:55.520 --> 44:59.720
And so I took Tanishq's notebook and just, as I was starting to understand it, I

44:59.720 --> 45:02.740
like to draw pictures for myself to understand what's going on.

45:02.740 --> 45:10.900
So I took the things which are in Tanishq's class and just put them into a cell.

45:10.900 --> 45:17.200
So I just copied and pasted them, although I replaced the Greek letters with English

45:17.200 --> 45:18.980
written out versions.

45:18.980 --> 45:21.920
And then I just plotted them to see what they look like.

45:21.920 --> 45:30.040
So in Tanishq's class, he has this thing called beta, which is just linspace.

45:30.040 --> 45:33.140
So that's just literally a line.

45:33.140 --> 45:37.220
So beta, there's going to be a thousand of them and they're just going to be equally

45:37.220 --> 45:43.340
spaced from 0.0001 to 0.02.

45:43.340 --> 45:50.940
And then there's something called sigma, which is the square root of that.

45:50.940 --> 45:55.580
So that's what sigma is going to look like.

45:55.580 --> 46:07.420
And then he's also got alphabar, which is a cumulative product of 1 minus this.

46:07.420 --> 46:09.660
And there's what alphabar looks like.

46:09.660 --> 46:19.020
So you can see here, as Tanishq was describing earlier, that when t is higher, this is t

46:19.020 --> 46:29.820
on the x-axis, beta is higher, and when t is higher, alphabar is lower.

46:29.820 --> 46:37.420
So yeah, so if you want to remind yourself, so each of these things, beta, sigma, alphabar,

46:37.420 --> 46:41.700
they're each, they've each got a thousand things in them.

46:41.700 --> 46:44.540
And this is the shape of those thousand things.

46:44.540 --> 46:52.540
So this is the amount of variance, I guess, added at each step.

46:52.540 --> 46:54.140
This is the square root of that.

46:54.140 --> 46:59.900
So it's the standard deviation added at each step.

46:59.900 --> 47:06.660
And then if we do 1 minus that, it's just the exact opposite.

47:06.660 --> 47:10.300
And then this is what happens if you multiply them all together up to that point.

47:10.300 --> 47:14.940
And the reason you do that is because if you add noise to something, you add noise to something

47:14.940 --> 47:17.700
that you add noise to something that you add noise to something, then you have to multiply

47:17.700 --> 47:22.180
together all that amount of noise to say how much noise you would get.

47:22.180 --> 47:25.820
So yeah, those are my pictures, if that's helpful.

47:25.820 --> 47:26.820
TANISHQ: Yep.

47:26.820 --> 47:27.820
Yep.

47:27.820 --> 47:35.620
Good to see the diagram or see how the actual values and how it changes over time.

47:35.620 --> 47:36.620
So yeah.

47:36.620 --> 47:38.620
Let's see here.

47:38.620 --> 47:39.620
Sorry.

47:39.620 --> 47:40.620
Yeah.

47:40.620 --> 47:45.480
So like Jeremy was showing, we have our linspace for our beta.

47:45.480 --> 47:48.260
In this case, we're using kind of more of the Greek letters.

47:48.260 --> 47:53.780
So you can see the Greek letters that we see in the paper as well as...

47:53.780 --> 47:56.140
Now we have it here in the code as well.

47:56.140 --> 48:01.040
And we have our linspace from our minimum value to our maximum value.

48:01.040 --> 48:02.780
And we have some number of steps.

48:02.780 --> 48:04.620
So this is the number of timesteps.

48:04.620 --> 48:10.020
So here we use a thousand timesteps, but that can depend on the type of model that

48:10.020 --> 48:11.580
you're training.

48:11.580 --> 48:16.300
And that's one of the parameters of your model or hyperparameters of your model.

48:16.300 --> 48:18.620
JEREMY: And this is the callback you've got here.

48:18.620 --> 48:27.400
So this callback is going to be used to set up the data, I guess, so that you're going

48:27.400 --> 48:34.920
to be using this to add the noise so that the model's then got the data that we're trying

48:34.920 --> 48:37.660
to get it to learn to then denoise.

48:37.660 --> 48:38.900
TANISHQ: Yeah.

48:38.900 --> 48:45.980
So the callback of course makes life a lot easier in terms of, yeah, setting up everything

48:45.980 --> 48:52.620
and still being able to use, I guess, the miniai learner with maybe some of these more

48:52.620 --> 48:57.360
complicated and maybe a little bit more unique training loops.

48:57.360 --> 49:09.060
So yeah, in this case, we're just able to use the callback in order to set up the batch

49:09.060 --> 49:11.140
that we are passing into our learner.

49:11.140 --> 49:19.460
JEREMY: I just want to mention, when you first did this, you wrote out the Greek letters in English,

49:19.460 --> 49:21.560
alpha and beta and so forth.

49:21.560 --> 49:27.860
And at least for my brain, I was finding it difficult to read because they were literally

49:27.860 --> 49:30.860
going off the edge of the page and I couldn't see it all at once.

49:30.860 --> 49:36.540
And so we did a search and replace to replace it with the actual Greek letters.

49:36.540 --> 49:41.180
I still don't know how I feel about it.

49:41.180 --> 49:44.460
I'm finding it easier to read because I can see it all at once.

49:44.460 --> 49:49.460
I don't know if it's a scroll and I don't get overwhelmed.

49:49.460 --> 49:55.960
But when I need to edit the code, I kind of just tend to copy and paste the Greek letters,

49:55.960 --> 50:02.780
which is why we use the actual word beta in the init parameter list so that somebody using

50:02.780 --> 50:06.060
this never has to type a Greek letter.

50:06.060 --> 50:10.260
But I don't know, Johno or Tanishq, if you had any thoughts over the last week or two,

50:10.260 --> 50:14.580
since we made that change about whether you guys like having the Greek letters in there

50:14.580 --> 50:15.580
or not.

50:15.580 --> 50:20.220
JOHNO: I like it for this demo in particular.

50:20.220 --> 50:23.820
I don't know that I do this in my code, but because we're looking back and forth between

50:23.820 --> 50:29.140
the paper and the implementation here, I think it works in this case just fine.

50:29.140 --> 50:31.860
TANISHQ: Yeah, I agree.

50:31.860 --> 50:38.260
I think it's good for when you're trying to study something or try to implement something,

50:38.260 --> 50:44.740
having the Greek letters is very useful to be able to, I guess, match the math more closely

50:44.740 --> 50:50.620
and it's just easy just to pick the equation and put it into code or white-source style

50:50.620 --> 50:53.180
looking at the code and try to match it to the equation.

50:53.180 --> 50:59.780
So I think for educational purpose, I tend to like, I guess, the Greek letters.

50:59.780 --> 51:02.580
So yeah.

51:02.580 --> 51:12.940
Yeah, so we have our initialization where we're just defining all these variables.

51:12.940 --> 51:16.460
We'll get to the predict in just a moment, but first I just want to go over the before_batch

51:16.460 --> 51:24.820
where we're setting up our batch to pass into the model.

51:24.820 --> 51:32.780
So remember that the model is taking in our noisy image and the timestep.

51:32.780 --> 51:39.620
And of course the target is the actual amount of noise that we are adding to the image.

51:39.620 --> 51:43.180
So basically we generate that noise.

51:43.180 --> 51:44.180
So that's what…

51:44.180 --> 51:45.420
JEREMY: So epsilon is that target.

51:45.420 --> 51:49.700
So epsilon is the amount of noise, not the amount of, is the actual noise.

51:49.700 --> 51:50.700
TANISHQ: Yes.

51:50.700 --> 51:53.620
Epsilon is the actual noise that we're adding.

51:53.620 --> 51:57.700
And that's the target as well because our model is a noise predicting model.

51:57.700 --> 52:00.100

It's predicting the noise in the image.

52:00.100 --> 52:04.780
And so our target should be the noise [unintelligible] that we're adding to the image during

52:04.780 --> 52:05.780
training.

52:05.780 --> 52:11.380
So we have our epsilon and we're just generating it with this random function.

52:11.380 --> 52:16.140
The random normal distribution with a mean of 0, variance of 1.

52:16.140 --> 52:21.220
So that's what that's doing and adding the appropriate shape and device.

52:21.220 --> 52:29.520
Then the batch that we get originally will contain the clean images.

52:29.520 --> 52:32.060
These are the original images from our dataset.

52:32.060 --> 52:34.520
So that's x0.

52:34.520 --> 52:37.160
And then what we want to do is we want to add noise.

52:37.160 --> 52:43.100
So we have our alphabar and we have a random timestep that we select.

52:43.100 --> 52:47.900
And then we just simply follow that equation, which I can, I'll just show in just a moment.

52:47.900 --> 52:50.220
JEREMY: That equation, you can make a tiny bit easier to read.

52:50.220 --> 52:55.380
I think if you were to double click on that first alphabar underscore t, cut it and then

52:55.380 --> 53:00.420
paste it, sorry, in the xt equals torch dot square root, take the thing inside the square

53:00.420 --> 53:06.960
root, double click it and paste it over the top of the word torch.

53:06.960 --> 53:13.540
That would be a little bit easier to read.

53:13.540 --> 53:17.900
And then you'll do the same for the next one.

53:17.900 --> 53:24.900
TANISHQ: There we go.

53:24.900 --> 53:26.900
Put those parentheses.

53:26.900 --> 53:27.900
Yep.

53:27.900 --> 53:36.260
TANISHQ: Yeah, so basically, yeah, so yeah, I guess I'll just pull up the equation.

53:36.260 --> 53:43.700
So let's see, there's then, so there's a section in the paper that has the nice algorithm.

53:43.700 --> 53:44.700
See if I can find it.

53:44.700 --> 53:45.700
No, no, here.

53:45.700 --> 53:48.700
It's, I think earlier.

53:48.700 --> 53:50.340
Yes, training.

53:50.340 --> 53:56.780
Right, so this, we're just following these same sort of training steps here, right?

53:56.780 --> 54:01.540
Where we select a clean image that we take from our data set.

54:01.540 --> 54:08.380
This fancy kind of equation here is just saying, take an image from your data set, take a random

54:08.380 --> 54:11.740
timestep between this range.

54:11.740 --> 54:16.660
Then this is our epsilon that we're getting, just saying, get some epsilon value.

54:16.660 --> 54:21.360
And then we have our equation for Xt, right?

54:21.360 --> 54:22.620
This is the equation here.

54:22.620 --> 54:27.600
You can see that it is square root of alphabar t x0 plus one square root of one minus

54:27.600 --> 54:29.820
alphabar t times epsilon.

54:29.820 --> 54:34.100
So that's the same equation that we have right here, right?

54:34.100 --> 54:37.420
And then what we need to do is we need to pass this into our model.

54:37.420 --> 54:41.580
So we have xt and t. So we set up our batch accordingly.

54:41.580 --> 54:45.020
So this is the two things that we pass into our model.

54:45.020 --> 54:48.220
And of course we also have our target, which is our epsilon.

54:48.220 --> 54:49.800
And so that's what this is showing here.

54:49.800 --> 54:54.680
We passed in our Xt as well as our t here, right?

54:54.680 --> 54:56.280
And we pass that into a model.

54:56.280 --> 54:59.260

The model is represented here as epsilon theta.

54:59.260 --> 55:03.580
And theta is often used to represent, like, this is a neural network with some parameters and

55:03.580 --> 55:05.520
the parameters are represented by theta.

55:05.520 --> 55:08.840
So epsilon theta is just representing our noise predicting model.

55:08.840 --> 55:10.280
So this is our neural network.

55:10.280 --> 55:15.040
So we have passed in our Xt and our t into a neural network, and we are comparing

55:15.040 --> 55:18.640
it to our target here, which is the actual epsilon.

55:18.640 --> 55:20.820
And so that's what we're doing here.

55:20.820 --> 55:26.940
We have our batch where we have our xt and t and epsilon.

55:26.940 --> 55:30.120
And then here we have our prediction function.

55:30.120 --> 55:35.260
And because we actually have, I guess in this case, we have two things that are in a tuple

55:35.260 --> 55:37.140
that we need to pass into our model.

55:37.140 --> 55:43.980
So we just kind of get those elements from our tuple with this.

55:43.980 --> 55:48.820
We get the elements from the tuple, pass it into the model, and then Hugging Face has

55:48.820 --> 55:50.840
its own API in terms of getting the output.

55:50.840 --> 55:55.740
So you'd need to call .sample in order to get the predictions from your model.

55:55.740 --> 55:56.880
So we just do that.

55:56.880 --> 56:03.260
And then we do, we have learn.preds and that's what's going to be used later then when we're

56:03.260 --> 56:07.360
trying to do our loss function calculation.

56:07.360 --> 56:10.980
JEREMY: So the, just so, I mean, it's just worth looking at that a little bit more since we haven't

56:10.980 --> 56:12.800
quite seen something like this before.

56:12.800 --> 56:17.220
And it's something which I'm not aware of any other framework that would let you do

56:17.220 --> 56:20.780
this, you know, literally replace how prediction works.

56:20.780 --> 56:23.420
And miniai is kind of really fun for this.

56:23.420 --> 56:29.780
So because you're inherited from TrainCB, TrainCB has predict, a dot defined and

56:29.780 --> 56:31.080
you've defined a new version.

56:31.080 --> 56:33.700
So it's not going to use the TrainCB version anymore.

56:33.700 --> 56:36.080
It's going to use your version.

56:36.080 --> 56:43.900
And what you're doing is instead of passing learn.batch[0] to the model, you're, you've

56:43.900 --> 56:45.900
got a * in front of it.

56:45.900 --> 56:50.660
So the key thing is that * is going to, you know, and is, we know that actually learn.batch[0]

56:50.660 --> 56:56.200
has two things in it because that learn.batch that you showed at the end of the before_batch

56:56.200 --> 56:58.900
method has two things in learn dot zero.

56:58.900 --> 57:04.140
So that star will unpack them and send each one of those as a separate argument.

57:04.140 --> 57:10.740
So our model needs to take two things, which the diffusers U-Net does take two things.

57:10.740 --> 57:12.180
So that's the main interesting point.

57:12.180 --> 57:20.380
And then something I find a bit awkward honestly, about a lot of Hugging Face stuff, including

57:20.380 --> 57:25.900
diffusers is that generally their models don't just return the result, but they put it inside

57:25.900 --> 57:27.300
some name.

57:27.300 --> 57:28.700
And so that's what happens here.

57:28.700 --> 57:31.740
They put it in something inside something called sample.

57:31.740 --> 57:37.980
So that's why Tanishq added .sample at the end of the predict because of this somewhat

57:37.980 --> 57:42.300
awkward thing, which Hugging Face like to do for some reason.

57:42.300 --> 57:47.460
But yeah, now that you know, I mean, this is something that people often get stuck on.

57:47.460 --> 57:49.420

I see on Kaggle and stuff like that.

57:49.420 --> 57:51.620
It's like, how on earth do I use these models?

57:51.620 --> 57:55.300
Because they take things in weird forms and they give back things with weird forms.

57:55.300 --> 57:56.740
Well, this is hell.

57:56.740 --> 58:03.380
You know, if you inherit from TranCB, you can change predict to do whatever you want,

58:03.380 --> 58:06.340
which I think is quite sweet.

58:06.340 --> 58:07.340
TANISHQ: Yep.

58:07.340 --> 58:12.420
So yeah, that's the training loop.

58:12.420 --> 58:17.700
And then of course you have your regular training loop that's implemented in miniai where you

58:17.700 --> 58:19.940
are going to have.

58:19.940 --> 58:20.940
Yeah.

58:20.940 --> 58:29.460
So you have your loss function calculation, I mean, and the predictions, learn.preds.

58:29.460 --> 58:35.580
And of course the target is our learn.batch[1], which is our epsilon.

58:35.580 --> 58:38.660
So we have those and we pass it into the loss function.

58:38.660 --> 58:41.900
It calculates the loss function and does the back propagation.

58:41.900 --> 58:43.300
So I'll just go over that.

58:43.300 --> 58:45.620
We'll get back to the sampling in just a moment.

58:45.620 --> 58:51.380
But just to show the training loop.

58:51.380 --> 58:59.100
JEREMY: Most of this is copied from our, I think it's 14_augment notebook, the way you've got the

58:59.100 --> 59:01.980
tmax and the sched.

59:01.980 --> 59:07.500
The only thing I think you've added here is the DDPM callback, right?

59:07.500 --> 59:08.500
TANISHQ: Yes.

59:08.500 --> 59:09.500
The DDPM callback.

59:09.500 --> 59:11.060
JEREMY: And you change the loss function.

59:11.060 --> 59:12.180
TANISHQ: Yes.

59:12.180 --> 59:18.080
So basically we have to initialize our DDPM callback with the appropriate arguments.

59:18.080 --> 59:24.780
So like the number of timesteps and the minimum beta and maximum beta.

59:24.780 --> 59:32.340
And then yeah, obviously, and then of course we're using an MSE loss as we talked about.

59:32.340 --> 59:36.940
It just becomes a regular training loop and everything else is run before.

59:36.940 --> 59:37.940

Yeah.

59:37.940 --> 59:42.740
So we have your scheduler, your progress bar, all of that we've seen before.

59:42.740 --> 59:46.900
JEREMY: I think that's really cool that we're using basically the same code to train a diffusion

59:46.900 --> 59:51.580
model as we've used to train a classifier just with one extra callback.

59:51.580 --> 59:52.580
TANISHQ: Yeah.

59:52.580 --> 59:53.580
Yeah.

59:53.580 --> 59:54.580
Yeah.

59:54.580 --> 59:58.780
That's why I think callbacks are very powerful for allowing us to do such things.

59:58.780 --> 01:00:06.020
It's like pretty, you can take all this code and now we have a diffusion training loop

01:00:06.020 --> 01:00:12.460
and we can just call learn.fit and yeah, they can see got a nice training loop, nice

01:00:12.460 --> 01:00:13.460
loss curve.

01:00:13.460 --> 01:00:21.100
We can save our model on a torch saving functionality to be able to save our model and we could

01:00:21.100 --> 01:00:22.940
load it in.

01:00:22.940 --> 01:00:28.160
But now that we have our trained model, then the question is, what can we do to use it

01:00:28.160 --> 01:00:32.100
to sample the dataset?

01:00:32.100 --> 01:00:44.180
So the basic idea of course was that we have, like basically we're here, right?

01:00:44.180 --> 01:00:46.180
We have, let's see here.

01:00:46.180 --> 01:00:47.180
Okay.

01:00:47.180 --> 01:00:52.140
So we have, the basic idea is that we start out with a random data point and of course that's

01:00:52.140 --> 01:01:00.580
not going to be within the distribution at first, but now we've learned how to move from

01:01:00.580 --> 01:01:03.520
one point towards the data distribution.

01:01:03.520 --> 01:01:06.220
That's what our noise prediction, predicting function does.

01:01:06.220 --> 01:01:15.700
It basically tells you how, you know, in what direction and how much to, so the basic idea

01:01:15.700 --> 01:01:20.060
is that, yeah, I guess I'll start from maybe a new drawing here.

01:01:20.060 --> 01:01:30.080
Again, we have, distribution is, and we have a random point and we use our noise predicting

01:01:30.080 --> 01:01:33.300
model that we have trained to tell us which direction to move.

01:01:33.300 --> 01:01:36.860
So it tells us some direction.

01:01:36.860 --> 01:01:42.140
Or I guess what's the exact, other area.

01:01:42.140 --> 01:01:43.140
Okay.

01:01:43.140 --> 01:01:44.140

So like here, okay.

01:01:44.140 --> 01:01:46.160
So it tells us some direction to move.

01:01:46.160 --> 01:01:49.980
At first that direction is not going to be like, you cannot follow that direction all

01:01:49.980 --> 01:01:53.180
the way to get the correct data point.

01:01:53.180 --> 01:01:56.780
Because basically what we were doing is we're trying to reverse the path that we were following

01:01:56.780 --> 01:01:58.560
when we were adding noise.

01:01:58.560 --> 01:02:02.260
So like, cause we had originally data point and we kept adding noise to the data point

01:02:02.260 --> 01:02:06.020
and maybe, you know, it followed some path like this.

01:02:06.020 --> 01:02:11.140
And we want to reverse that path to get to.

01:02:11.140 --> 01:02:15.960
So our noise predicting function will give us an original direction, which, you know,

01:02:15.960 --> 01:02:22.220
would be, you know, some kind of, it's going to be kind of tangential to the actual path

01:02:22.220 --> 01:02:23.420
at that location.

01:02:23.420 --> 01:02:28.780
So what we would do is, you know, we would maybe follow that data point all the way towards,

01:02:28.780 --> 01:02:34.820
you know, we're just going to keep following that data point.

01:02:34.820 --> 01:02:39.780
You know, we're going to try to predict the fully denoised image by following this noise

01:02:39.780 --> 01:02:40.780
prediction.

01:02:40.780 --> 01:02:45.380
But our fully denoised image is also not going to be a real image.

01:02:45.380 --> 01:02:51.100
So what we, so let me, I'll show an example of that over here in the paper on why they

01:02:51.100 --> 01:02:56.180
show this a little bit more carefully.

01:02:56.180 --> 01:02:58.920
So x0, it's there.

01:02:58.920 --> 01:03:05.960
So basically you can see the different data...

01:03:05.960 --> 01:03:08.220
You can see the different data points here.

01:03:08.220 --> 01:03:10.560
It's not going to look anything like our real image.

01:03:10.560 --> 01:03:17.080
So you can see all these points, you know, it doesn't look anything… what we would do

01:03:17.080 --> 01:03:23.220
is we actually had a little bit of noise back to it.

01:03:23.220 --> 01:03:27.460
And we start, we have a new point where then we could maybe estimate a better, get a better

01:03:27.460 --> 01:03:33.900
estimate of which direction to move, follow that all the way again, we follow a new point.

01:03:33.900 --> 01:03:36.080
And then I can add back a little bit of noise.

01:03:36.080 --> 01:03:41.540
You get a new estimate, you make a new estimate of, you know, this noise prediction and removing

01:03:41.540 --> 01:03:46.420

the noise, you know, fall that all again, completely and add a little bit of noise again to the

01:03:46.420 --> 01:03:52.220
image and burst onto a image.

01:03:52.220 --> 01:03:54.060
So that's kind of what we're showing here.

01:03:54.060 --> 01:03:58.600
JEREMY: That's a lot like SGD, with SGD we don't take the gradient and jump all the way.

01:03:58.600 --> 01:04:03.180
We use a learning rate to go some of the way because each of those estimates of where we

01:04:03.180 --> 01:04:08.940
want to go, you know, not that great, but we just do it slowly.

01:04:08.940 --> 01:04:09.940
TANISHQ: Exactly.

01:04:09.940 --> 01:04:12.820
And at the end of the day, that's what we're doing with this noise prediction.

01:04:12.820 --> 01:04:20.500
We are predicting the sort of gradient of this p(x), but of course we need to keep

01:04:20.500 --> 01:04:23.580
making estimates of that gradient as we're progressing.

01:04:23.580 --> 01:04:30.140
So we have to keep evaluating our noise prediction function to get updated and better estimates

01:04:30.140 --> 01:04:35.780
of our gradient in order to finally converge onto our image.

01:04:35.780 --> 01:04:41.020
So and then you can see that here, you know, we have this, maybe this fully predicted denoised

01:04:41.020 --> 01:04:46.900
image which at the beginning doesn't look anything like a real image, but then as we

01:04:46.900 --> 01:04:50.880
continue throughout the sampling process, we finally converge on something that looks

01:04:50.880 --> 01:04:52.340
like an actual image.

01:04:52.340 --> 01:04:57.080
Again, these are CIFAR-10 images and it's still a little bit maybe unclear about how

01:04:57.080 --> 01:05:02.220
realistic these images, these very small images look, but that's kind of the general principle

01:05:02.220 --> 01:05:03.340
I would say.

01:05:03.340 --> 01:05:09.860
And so that's what I can show in the code.

01:05:09.860 --> 01:05:16.880
This idea of we're going to start out basically with a random image, right?

01:05:16.880 --> 01:05:21.660
And this random image is going to be like a pure noise image and it's not going to be

01:05:21.660 --> 01:05:23.260
part of the data distribution.

01:05:23.260 --> 01:05:26.580
You know, it’s not anything like a real image, it's just a random image.

01:05:26.580 --> 01:05:31.340
And so this is going to be our x, I guess, x uppercase t [x_t], right?

01:05:31.340 --> 01:05:32.720
That's what we start out with.

01:05:32.720 --> 01:05:36.400
And we want to go from x uppercase t all the way to x0.

01:05:36.400 --> 01:05:44.700
So what we do is we go through each of the timesteps and we create, we have to put it

01:05:44.700 --> 01:05:49.740
in this sort of batch format because that's what our neural network expects.

01:05:49.740 --> 01:05:54.300
So we just have to format it appropriately.

01:05:54.300 --> 01:05:56.660
And we'll get to z in just a moment.

01:05:56.660 --> 01:06:03.820
I'll explain that in just a moment, but of course we just take it have similar… alphabar,

01:06:03.820 --> 01:06:07.020
betabar, which is getting those variables that we need.

01:06:07.020 --> 01:06:09.660
JEREMY: And we faked beta bar because we couldn't figure out how to type it.

01:06:09.660 --> 01:06:11.460
So we used bbar instead.

01:06:11.460 --> 01:06:12.460
TANISHQ: Yeah.

01:06:12.460 --> 01:06:18.740
So yeah, so yeah, in, yeah, we, yeah, we were pretty able to get betabar to work, I guess.

01:06:18.740 --> 01:06:22.560
But anyway, at each step, what we're trying to do is to try to predict what direction

01:06:22.560 --> 01:06:23.560
we need to go.

01:06:23.560 --> 01:06:26.420
And that direction is given by our noise predicting model, right?

01:06:26.420 --> 01:06:32.060
So what we do is we pass in x_t and our current timestep into our model.

01:06:32.060 --> 01:06:36.420
And we get this noise prediction and that's the direction that we need to move in.

01:06:36.420 --> 01:06:42.380
So basically we take x_t. We first attempt to completely remove the noise, right?

01:06:42.380 --> 01:06:43.380
That's what this is doing.

01:06:43.380 --> 01:06:45.000

That's what x_0_hat is.

01:06:45.000 --> 01:06:48.500
That's completely removing the noise.

01:06:48.500 --> 01:06:53.140
And of course, as we said, that estimate at the beginning won't be very accurate.

01:06:53.140 --> 01:06:58.420
And so now what we do is we have some coefficients here where we have a coefficient of how much

01:06:58.420 --> 01:07:06.780
that we keep about this estimate of our denoise image and how much of the originally noisy

01:07:06.780 --> 01:07:08.580
image we keep.

01:07:08.580 --> 01:07:12.660
And on top of that, we're going to add in some additional noise.

01:07:12.660 --> 01:07:14.720
So that's what we do here.

01:07:14.720 --> 01:07:19.740
We have x_0_hat.

01:07:19.740 --> 01:07:25.260
And so, and we multiply by its coefficient and we have x_t we multiply it by some

01:07:25.260 --> 01:07:28.200
coefficient and we also add some additional noise.

01:07:28.200 --> 01:07:29.200
That's what the z is.

01:07:29.200 --> 01:07:30.200
It's just-

01:07:30.200 --> 01:07:35.180
JEREMY: That's basically a weighted average of the two plus the noise…

01:07:35.180 --> 01:07:36.180
TANISHQ: Exactly.

01:07:36.180 --> 01:07:44.380
And then the whole idea is that as we get closer and closer to a timestep equals to 0

01:07:44.380 --> 01:07:50.220
our estimate of x0 will be more and more accurate.

01:07:50.220 --> 01:07:58.820
So our x0_coeff will get closer as we're increasing our, going through the

01:07:58.820 --> 01:08:04.100
process and then our xt_coeff will get closer and closer to 0.

01:08:04.100 --> 01:08:10.020
So basically we're going to be weighting more and more of the x_0_hat estimate and less

01:08:10.020 --> 01:08:15.180
and less of the x_t as we're getting closer and closer to our final timestep.

01:08:15.180 --> 01:08:22.180
And so at the end of the day, we will have our estimated generated image.

01:08:22.180 --> 01:08:27.780
So that's kind of an overview of the sampling process.

01:08:27.780 --> 01:08:31.420
So yeah.

01:08:31.420 --> 01:08:35.780
So yeah, basically the way I implemented it here was I had the sample function that's

01:08:35.780 --> 01:08:44.620
part of our callback and it will take in the model and the kind of shape that you want

01:08:44.620 --> 01:08:47.280
for your images that you're producing.

01:08:47.280 --> 01:08:50.820
So like if you want to specify how many images you produce, that's going to be part of your

01:08:50.820 --> 01:08:51.820
batch size or whatever.

01:08:51.820 --> 01:08:54.060

And you'll just see that in a moment.

01:08:54.060 --> 01:08:56.380
But yeah, it's just part of the callback.

01:08:56.380 --> 01:09:04.900
So then we basically have our DDPM callback and then we could just call the sample method

01:09:04.900 --> 01:09:09.340
of our DDPM callback and we pass in our model.

01:09:09.340 --> 01:09:13.700
And then here you can see we're going to produce, for example, 16 images and it just has to

01:09:13.700 --> 01:09:20.740
be a 1 channel image of shape 32 by 32 and we get our samples.

01:09:20.740 --> 01:09:27.580
And one thing I forgot to note was that I am collecting each of the timestep, the x_t.

01:09:27.580 --> 01:09:34.380
So the predictions here, you can see that there are a thousand of them.

01:09:34.380 --> 01:09:38.700
We want the last one because that is our final generation.

01:09:38.700 --> 01:09:40.340
So we want the last one and that's what we should-

01:09:40.340 --> 01:09:42.340
JEREMY: They are not [sad] actually.

01:09:42.340 --> 01:09:43.340
TANISHQ: Yeah.

01:09:43.340 --> 01:09:44.340
So this is-

01:09:44.340 --> 01:09:45.860
JEREMY: We've come a long way since DDPM.

01:09:45.860 --> 01:09:49.900
So this is, like, slower and less great than it could be.

01:09:49.900 --> 01:09:55.060
But considering that, except for U-NET, we've done this from scratch, you know, actually

01:09:55.060 --> 01:09:59.660
from matrix multiplication, I think those are pretty decent.

01:09:59.660 --> 01:10:00.860
TANISHQ: Yeah.

01:10:00.860 --> 01:10:03.560
And we're only trained for about five epochs.

01:10:03.560 --> 01:10:07.540
It took like, you know, maybe like four minutes to train this model, something like that.

01:10:07.540 --> 01:10:09.300
It's pretty quick.

01:10:09.300 --> 01:10:13.140
And this is what we get with very little training.

01:10:13.140 --> 01:10:14.780
And it's pretty decent.

01:10:14.780 --> 01:10:20.740
You can see, of course, some clear shirts and shoes and pants and whatever else.

01:10:20.740 --> 01:10:21.740
JEREMY: Yeah.

01:10:21.740 --> 01:10:25.060
And you can see fabric and it's got texture and things have buckles and-

01:10:25.060 --> 01:10:26.060
TANISHQ: Yeah.

01:10:26.060 --> 01:10:34.540
JEREMY: You know, something to compare, like, we did generative modeling in the first time we did

01:10:34.540 --> 01:10:41.100
Part 2 back in the days when Wasserstein GAN was just new, which was actually

01:10:41.100 --> 01:10:45.900

created by the same guy that created PyTorch or one of the two guys, Soumith.

01:10:45.900 --> 01:10:51.900
And we trained for hours and hours and hours and got things that I'm not sure were any

01:10:51.900 --> 01:10:53.900
better than this.

01:10:53.900 --> 01:10:56.500
So things have come a long way.

01:10:56.500 --> 01:10:57.500
TANISHQ: Yeah.

01:10:57.500 --> 01:10:58.500
Yeah.

01:10:58.500 --> 01:11:08.520
And of course, then, yeah, so we can see then like how this sampling progresses over time,

01:11:08.520 --> 01:11:11.700
over the multiple timesteps.

01:11:11.700 --> 01:11:15.100
So that's what I'm showing here because I collected, during the sampling process, we

01:11:15.100 --> 01:11:20.140
are collecting at each timestep what that estimate looks like.

01:11:20.140 --> 01:11:21.860
And you can kind of see here.

01:11:21.860 --> 01:11:25.500
And so this is an estimate of the noisy image over the timesteps.

01:11:25.500 --> 01:11:26.500
Oops.

01:11:26.500 --> 01:11:27.980
And I guess I had to pause.

01:11:27.980 --> 01:11:28.980
Yeah.

01:11:28.980 --> 01:11:30.480
You can kind of see.

01:11:30.480 --> 01:11:34.180
But you'll notice that actually, so we actually, what we did is like, okay, so we selected

01:11:34.180 --> 01:11:36.780
an image, which is like the ninth image.

01:11:36.780 --> 01:11:38.560
So that's, that's this image here.

01:11:38.560 --> 01:11:42.820
So we're looking at this image particularly here and we're going over.

01:11:42.820 --> 01:11:43.820
Yeah.

01:11:43.820 --> 01:11:49.420
We have a function here that's showing the i-th timestep during the sampling process of

01:11:49.420 --> 01:11:51.300
that image.

01:11:51.300 --> 01:11:54.380
And we're just getting the images.

01:11:54.380 --> 01:12:01.780
And what we are doing is we're only showing basically from timestep 800 to a 1,000.

01:12:01.780 --> 01:12:05.020
And here we're just, we're just having it like where it's like, okay, we're looking

01:12:05.020 --> 01:12:09.180
at like maybe every 5 steps and we're going from 800 to 990.

01:12:09.180 --> 01:12:13.780
And this time it would make it visually easier to see the transition.

01:12:13.780 --> 01:12:16.480
But what you'll notice is I didn't start all the way from 0.

01:12:16.480 --> 01:12:18.460
I started from 800.

01:12:18.460 --> 01:12:25.260
And the reason we do that is because actually between 0 and 800 there's very little change

01:12:25.260 --> 01:12:29.380
in terms of like, it's just mostly a noisy image.

01:12:29.380 --> 01:12:33.620
And it turns out, but yeah, I didn't see as I make a note of this here, it's actually

01:12:33.620 --> 01:12:39.180
a limitation of the noise schedule that is used in the original DDPM paper.

01:12:39.180 --> 01:12:43.260
And especially when applied to some of these smaller images, when we're working with images

01:12:43.260 --> 01:12:47.020
of like size 32 by 32 or whatever.

01:12:47.020 --> 01:12:52.960
And so there are some other papers like the improved DDPM paper that propose other sorts

01:12:52.960 --> 01:12:54.300
of noise schedules.

01:12:54.300 --> 01:13:00.020
And what I mean by noise schedule is basically how beta is defined basically.

01:13:00.020 --> 01:13:05.720
So we had this definition of torch.linspace for our beta, but people have different ways

01:13:05.720 --> 01:13:09.100
of defining beta that lead to different properties.

01:13:09.100 --> 01:13:13.900
So things like that, people have come up with different improvements and those sorts of

01:13:13.900 --> 01:13:17.340
improvements work well when we're working with these smaller images.

01:13:17.340 --> 01:13:21.460
And basically the point is like, if we are working from 0 to 800 and it's just mostly

01:13:21.460 --> 01:13:27.300

just noise that entire time, we're not actually making full use of all this timesteps.

01:13:27.300 --> 01:13:30.540
So it would be nice if we could actually make full use of those time steps and actually

01:13:30.540 --> 01:13:33.460
have it do something during that time period.

01:13:33.460 --> 01:13:37.340
So all these, there are some papers that examine this a little bit more carefully and it would

01:13:37.340 --> 01:13:41.860
be kind of interesting for maybe some of you folks to also look at these papers and see

01:13:41.860 --> 01:13:48.500
if you can try to implement those sorts of models with this notebook as a starting point.

01:13:48.500 --> 01:13:51.580
And it should be a fairly simple change in terms of like noise schedule or something

01:13:51.580 --> 01:13:52.580
like that.

01:13:52.580 --> 01:13:58.260
JEREMY: So I actually think this is the start of our next journey, which is our previous journey

01:13:58.260 --> 01:14:08.340
was going from being totally rubbish at Fashion-MNIST classification to being really good at it.

01:14:08.340 --> 01:14:16.380
I heard you say now we're like a little bit rubbish at doing Fashion-MNIST generation.

01:14:16.380 --> 01:14:24.180
And yeah, I think we should all now work from here over the next few lessons and so forth

01:14:24.180 --> 01:14:33.380
and people trying things at home and all of us trying to make better and better generative

01:14:33.380 --> 01:14:37.260
models, initially a Fashion-MNIST and hopefully we'll get to the point where we're so good

01:14:37.260 --> 01:14:39.100
at that, that we're like, oh, this is too easy.

01:14:39.100 --> 01:14:44.820
And then we'll pick something harder.

01:14:44.820 --> 01:14:50.740
And eventually that'll take us to Stable Diffusion and beyond.

01:14:50.740 --> 01:14:55.740
I imagine.

01:14:55.740 --> 01:14:56.740
That's cool.

01:14:56.740 --> 01:15:02.340
I got some stuff to show you guys.

01:15:02.340 --> 01:15:12.200
If you're interested, I tried to better understand what was going on in Tanishq's notebook and

01:15:12.200 --> 01:15:15.220
tried doing it in a thousand different ways and also see if I could just start to make

01:15:15.220 --> 01:15:18.700
it a bit faster.

01:15:18.700 --> 01:15:29.800
So that's what's in notebook 17, which I will share.

01:15:29.800 --> 01:15:32.060
So we've already seen the start of notebook 17.

01:15:32.060 --> 01:15:36.880
Well, the one thing I did just do is just drew a picture for myself, partly just to

01:15:36.880 --> 01:15:39.060
remind myself what they, what the real ones look like.

01:15:39.060 --> 01:15:48.540
And they definitely have more detail than the samples that Tanishq was showing.

01:15:48.540 --> 01:15:50.780
But they're not, you know, they're just 28 by 28.

01:15:50.780 --> 01:15:53.600
I mean, they're not super amazing images and they're just black or white.

01:15:53.600 --> 01:15:58.780
So even if we're fantastic at this, they're never going to look great because we're using

01:15:58.780 --> 01:16:01.220
a small, simple data set.

01:16:01.220 --> 01:16:06.480
As you always should, when you're doing any kind of R&D or experiments, you should

01:16:06.480 --> 01:16:11.860
always use a small and simple data set up until you're so good at it that it's not challenging

01:16:11.860 --> 01:16:14.860
anymore.

01:16:14.860 --> 01:16:18.380
And even then when you're exploring new ideas, you should explore them on small, simple data

01:16:18.380 --> 01:16:19.380
sets first.

01:16:19.380 --> 01:16:20.380
Yeah.

01:16:20.380 --> 01:16:27.220
So after I drew the various things, what I like to do is one thing I found challenging

01:16:27.220 --> 01:16:32.420
about working with your class Tanishq is I find when stuff is inside a class, it's harder

01:16:32.420 --> 01:16:34.100
for me to explore.

01:16:34.100 --> 01:16:43.800
So I copied and pasted it, the before_batch contents and called it noisify.

01:16:43.800 --> 01:16:48.000
And so one of the things that's fun to do that is it forces you to figure out what are

01:16:48.000 --> 01:16:49.900
the actual parameters to it.

01:16:49.900 --> 01:16:55.660

And so now that I, rather than putting in the class, now that I've got all of my, you know,

01:16:55.660 --> 01:17:02.140
various things to do with, so these are the three parameters to the DDPM callbacks in

01:17:02.140 --> 01:17:03.140
it.

01:17:03.140 --> 01:17:06.300
And then these things we can calculate from that.

01:17:06.300 --> 01:17:13.420
So with those then actually all we need is yeah, what's the image that we're going to

01:17:13.420 --> 01:17:19.180
noisify and then what's the, what's the alphabar, which I mean, we can get from here, but

01:17:19.180 --> 01:17:22.500
it's sort of be more general if you can pass in your alphabar.

01:17:22.500 --> 01:17:28.220
So yeah, this is just copying and pasting from the class, but the nice thing is then

01:17:28.220 --> 01:17:29.880
I could experiment with it.

01:17:29.880 --> 01:17:39.380
So I can call noisify on my first 25 images and with, with a random t, each one's got a

01:17:39.380 --> 01:17:43.500
different random t, and so I can print out the t and then I could actually use those

01:17:43.500 --> 01:17:44.960
as titles.

01:17:44.960 --> 01:17:47.180
And so this lets me, I thought this is quite nice.

01:17:47.180 --> 01:17:52.540
I might actually rerun this cause actually none of these look like anything because as

01:17:52.540 --> 01:17:57.860
it turns out in this particular case, all of the t's are over 200.

01:17:57.860 --> 01:18:02.700
And as Tanishq mentioned, once you're over 200, it's almost impossible to see anything.

01:18:02.700 --> 01:18:09.980
So let me just rerun this and see if we get a better, there we go.

01:18:09.980 --> 01:18:10.980
There's a better one.

01:18:10.980 --> 01:18:13.540
So with a t of 7, right?

01:18:13.540 --> 01:18:17.300
So remember t naught, t equals naught is the pure image.

01:18:17.300 --> 01:18:21.200
So t equals 7, it's just a slightly speckledy image.

01:18:21.200 --> 01:18:24.220
And by 67, it's a pretty bad image.

01:18:24.220 --> 01:18:30.100
And by 94, it's very hard to see what it is at all.

01:18:30.100 --> 01:18:34.620
And by 293, maybe I can see a pair of pants.

01:18:34.620 --> 01:18:36.700
I'm not sure I can see anything.

01:18:36.700 --> 01:18:40.260
So yeah.

01:18:40.260 --> 01:18:45.580
By the way, there's a handy little, so we've, I think we've looked at map before in in the

01:18:45.580 --> 01:18:48.700
course there's an extended version of map in fastcore.

01:18:48.700 --> 01:18:53.600
And one of the nice things is you can pass it a string and it basically just calls this

01:18:53.600 --> 01:18:57.340
format string if you pass it a string rather than a function.

01:18:57.340 --> 01:19:00.860
And so this is going to stringify everything using its representations.

01:19:00.860 --> 01:19:06.580
This is how I got the titles out of it just by the way.

01:19:06.580 --> 01:19:12.640
So yeah, I found this useful to be able to draw a picture of everything.

01:19:12.640 --> 01:19:17.420
And then I wanted to, yeah, look at what, what, what else, what else can I do?

01:19:17.420 --> 01:19:18.420
So then I took nuts.

01:19:18.420 --> 01:19:19.420
You won't be surprised to see.

01:19:19.420 --> 01:19:22.980
I took the sample method and turn that into a function.

01:19:22.980 --> 01:19:25.540
And I actually decided to pass everything that it needs.

01:19:25.540 --> 01:19:29.820
Even, I mean, you could actually calculate pretty much all of these.

01:19:29.820 --> 01:19:32.100
But I thought since I've calculated them before, just pass them in.

01:19:32.100 --> 01:19:35.980
So this is all copied and pasted from Tanishq's version.

01:19:35.980 --> 01:19:39.460
And so that means the callback now is tiny, right?

01:19:39.460 --> 01:19:47.700
Because before_batch is just noisify and the sample method just calls the sample function.

01:19:47.700 --> 01:19:52.260
Now, what I did do is I decided just to, yeah, I wanted to try like as many different ways

01:19:52.260 --> 01:19:54.460

of doing this as possible.

01:19:54.460 --> 01:20:01.340
Partly it's an exercise to help everybody like see all the different ways we can work

01:20:01.340 --> 01:20:03.420
with our framework, you know?

01:20:03.420 --> 01:20:07.940
So I decided not to inherit from TrainCB, but I instead I inherited from Callback.

01:20:07.940 --> 01:20:15.980
So that means I can't use Tanishq's nifty trick of replacing predict.

01:20:15.980 --> 01:20:22.380
So instead I now need some way to pass in the two parts of the first element of the

01:20:22.380 --> 01:20:28.140
tuple as separate things to the model and return the sample.

01:20:28.140 --> 01:20:29.740
So how else could we do that?

01:20:29.740 --> 01:20:35.540
Well what we could do is we could actually inherit from UNet2DModel, which is what

01:20:35.540 --> 01:20:40.460
Tanishq used directly, unit 2d model, and we could replace the model.

01:20:40.460 --> 01:20:43.180
And so we could replace specifically the forward function.

01:20:43.180 --> 01:20:45.140
That's the thing that gets called.

01:20:45.140 --> 01:20:50.340
And we could just call the original forward function, but rather than passing an x

01:20:50.340 --> 01:20:55.620
we're passing a *x, and rather than returning that, we'll return that .sample.

01:20:55.620 --> 01:20:56.620
Okay.

01:20:56.620 --> 01:21:03.140
So if we do that, then we don't need the TrainCB anymore and we don't need the predict.

01:21:03.140 --> 01:21:09.100
And so if you're not working with something as beautifully flexible as miniai, you can

01:21:09.100 --> 01:21:16.460
always do this, you know, to make, to replace your model so that it has the interface that

01:21:16.460 --> 01:21:18.420
you need it to have.

01:21:18.420 --> 01:21:23.180
So now again, we did the same as Tanishq had of create the callback.

01:21:23.180 --> 01:21:28.500
And now when we create the model, we'll reuse our UNet class, which we just created.

01:21:28.500 --> 01:21:30.300
I wanted to see if I can make things faster.

01:21:30.300 --> 01:21:37.700
I tried dividing all of Tanishq's channels by two and I found it worked just as well.

01:21:37.700 --> 01:21:43.940
One thing I noticed is that it uses group norm in the U-Net, which we have briefly learned

01:21:43.940 --> 01:21:51.300
about before and in group norm, it splits the channels up into a certain number of groups.

01:21:51.300 --> 01:21:57.420
And I needed to make sure that those groups had more than one thing in.

01:21:57.420 --> 01:22:03.020
So you can actually pass in how many groups do you want to use in the normalization.

01:22:03.020 --> 01:22:04.020
So that's what this is for.

01:22:04.020 --> 01:22:08.380
You gotta be a little bit careful of these things, I didn't think of it at first and

01:22:08.380 --> 01:22:14.820
I ended up, I think the num groups might've been 32 and I got an error saying you can't

01:22:14.820 --> 01:22:18.140
split 16 things into 32 groups.

01:22:18.140 --> 01:22:22.340
But it also made me realize actually, even in Tanishq's maybe you probably had 32 in

01:22:22.340 --> 01:22:23.940
the first with 32 groups.

01:22:23.940 --> 01:22:26.300
And so maybe the group norm wouldn't have been working as well.

01:22:26.300 --> 01:22:30.860
So they're little subtle things to look out for.

01:22:30.860 --> 01:22:36.580
So now that we're not using anything inherited from TrainCB, that means we either need to

01:22:36.580 --> 01:22:41.460
use TrainCB itself or just use our train learner and that everything else is the same as what

01:22:41.460 --> 01:22:44.300
Tanishq had.

01:22:44.300 --> 01:22:51.660
So then I wanted to look at the results of noisify here and we've seen this trick before,

01:22:51.660 --> 01:22:58.340
which is we call fit, but don't call the training part of the fit and use the SingleBatchCB

01:22:58.340 --> 01:23:02.460
callback that we created way back when we first created Learner.

01:23:02.460 --> 01:23:10.060
And now learn.batch will contain the tuple of tuples, which we can then use that trick

01:23:10.060 --> 01:23:11.380
to show.

01:23:11.380 --> 01:23:15.740
So I mean, obviously we'd expect it to look the same as before, but it's nice.

01:23:15.740 --> 01:23:18.700

I always like to draw pictures of everything all along the way.

01:23:18.700 --> 01:23:19.700
Cause it's very, very of..

01:23:19.700 --> 01:23:24.420
I mean, I, the first six to seven times I do anything, I do it wrong.

01:23:24.420 --> 01:23:29.060
So given that I know that I might as well draw a picture to try and see how it's wrong

01:23:29.060 --> 01:23:30.060
until it's fixed.

01:23:30.060 --> 01:23:33.260
It also tells me when it's not wrong.

01:23:33.260 --> 01:23:38.860
TANISHQ: Isn't there a show_batch function now that does something similar?

01:23:38.860 --> 01:23:44.180
JEREMY: Um, yes, you wrote that show_image_batch, didn't you?

01:23:44.180 --> 01:23:45.500
I can't quite remember.

01:23:45.500 --> 01:23:46.500
Yeah.

01:23:46.500 --> 01:23:51.880
We should, uh, remind ourselves how that worked.

01:23:51.880 --> 01:23:54.660
That's a good point.

01:23:54.660 --> 01:23:55.900
Thanks for a reminder.

01:23:55.900 --> 01:23:56.900
Okay.

01:23:56.900 --> 01:24:00.220
So then, um, I'll just go ahead and do the same thing that Tanishq did.

01:24:00.220 --> 01:24:05.780
And um, uh, but then the next thing I looked at was I looked at the, you know, how am I

01:24:05.780 --> 01:24:07.300
going to make this train faster?

01:24:07.300 --> 01:24:09.980
I want a bigger, I want a higher learning rate.

01:24:09.980 --> 01:24:18.660
Um, and I realized, oddly enough, the diffusers code does not initialize anything at all.

01:24:18.660 --> 01:24:24.500
They use the defaults, um, which just goes to show like even, you know, the experts at

01:24:24.500 --> 01:24:31.580
Hugging Face that don't necessarily really think like, oh, maybe the PyTorch defaults

01:24:31.580 --> 01:24:34.660
aren't, you know, perfect for my model.

01:24:34.660 --> 01:24:38.660
Of course they're not because they depend on what activation function do you have and

01:24:38.660 --> 01:24:41.620
what res blocks do you have and so forth.

01:24:41.620 --> 01:24:47.140
Um, so I wasn't exactly sure how to initialize it.

01:24:47.140 --> 01:24:53.100
Um, I, um, partly by chatting to, um, Kat Crowson, who's the author of K-diffusion,

01:24:53.100 --> 01:24:59.220
um, and partly by looking at papers and partly by thinking about my own experience, I ended

01:24:59.220 --> 01:25:02.500
up doing a few things.

01:25:02.500 --> 01:25:08.360
One is I did do the thing that we talked about a while ago, which is to take every second

01:25:08.360 --> 01:25:10.740
convolutional layer and zero it out.

01:25:10.740 --> 01:25:13.820
You could do the same thing with using batch norm, which is what we tried.

01:25:13.820 --> 01:25:17.740
And since we've got quite a deep network, you know, that seemed like it might, you know,

01:25:17.740 --> 01:25:25.740
it helps basically by having the, the, the non-id paths in the ResNets do nothing at

01:25:25.740 --> 01:25:28.780
first so they can't cause problems.

01:25:28.780 --> 01:25:35.760
Um, we haven't talked about, um, orthogonalized weights before, and we probably won't because

01:25:35.760 --> 01:25:42.340
you would need to take our, um, computational linear algebra course to learn about that,

01:25:42.340 --> 01:25:45.420
which is a great course, Rachel Thomas did a fantastic job of it.

01:25:45.420 --> 01:25:49.460
I highly recommend it, but I don't want to make it a prerequisite, but, um, Kat mentioned,

01:25:49.460 --> 01:25:55.780
she thought that using orthogonal weights for the downsamplers was a good idea.

01:25:55.780 --> 01:26:03.160
Um, and then, well, the up_blocks, they also set the second convs to zero and something

01:26:03.160 --> 01:26:09.540
Kat mentioned, she found useful, which is also from, um, I think it's from the Dhariwal

01:26:09.540 --> 01:26:14.340
Google paper is to also zero out the weights of basically the very last layer.

01:26:14.340 --> 01:26:21.100
Um, and so it's going to start by predicting zero as the noise, which is, you know, something

01:26:21.100 --> 01:26:22.100
that can't hurt.

01:26:22.100 --> 01:26:26.580
Um, so that was, that's how I initialized the weights.

01:26:26.580 --> 01:26:33.100
Um, so call init_ddpm on my model, uh, something that I found that a huge difference is I replaced

01:26:33.100 --> 01:26:38.260
the normal Adam optimizer with one that has an epsilon of 1e-5, the default,

01:26:38.260 --> 01:26:40.260
I think is 1e-8.

01:26:40.260 --> 01:26:49.220
And so to remind you, this is, we, is we, when we divide by the exponentially weighted

01:26:49.220 --> 01:26:52.980
moving average of the squared gradients, we, when we divide by that, if that's a very,

01:26:52.980 --> 01:27:00.540
very small number, um, then it makes the effective learning rate huge.

01:27:00.540 --> 01:27:04.260
And so we add this to it to make it not too huge.

01:27:04.260 --> 01:27:07.460
And it's nearly always a good idea to make this bigger than the default.

01:27:07.460 --> 01:27:09.620
I don't know why the default is so small.

01:27:09.620 --> 01:27:15.140
And I found, until I did this, anytime I tried to use a reasonably large learning rate somewhere

01:27:15.140 --> 01:27:18.900
around the middle of the 1-Cycle training, it would explode.

01:27:18.900 --> 01:27:22.700
Um, uh, so that makes a big difference.

01:27:22.700 --> 01:27:27.300
Um, so this way, yeah.

01:27:27.300 --> 01:27:33.180

Uh, I could train, I could get 0.016 after 5 epochs.

01:27:33.180 --> 01:27:37.820
Um, and then sampling, so it looks all pretty similar.

01:27:37.820 --> 01:27:41.180
We've got some pretty nice textures, I think.

01:27:41.180 --> 01:27:43.900
So then I was thinking, how do I get faster?

01:27:43.900 --> 01:27:49.500
So one way we can make it faster is we can take advantage of, um, something called

01:27:49.500 --> 01:27:51.500
mixed precision.

01:27:51.500 --> 01:27:57.620
Um, so currently we're using 32 bit floating point values.

01:27:57.620 --> 01:28:02.040
Um, that's the defaults and also known as single precision.

01:28:02.040 --> 01:28:09.020
And um, GPUs are pretty fast at doing 32 bit floating point values, but they're much, much,

01:28:09.020 --> 01:28:12.820
much, much faster during 16 bit floating point values.

01:28:12.820 --> 01:28:17.020
So I'm 16 bit floating point values.

01:28:17.020 --> 01:28:22.300
I'm able to represent a very, you know, wide range of numbers or much precision at the

01:28:22.300 --> 01:28:23.640
difference between numbers.

01:28:23.640 --> 01:28:29.540
And so they're quite difficult to use, but if you can, you'll get a huge benefit because,

01:28:29.540 --> 01:28:37.940
um, modern GPUs, modern Nvidia GPU specifically have special units that do matrix multiplies

01:28:37.940 --> 01:28:41.380
of 16 bit values extremely quickly.

01:28:41.380 --> 01:28:49.120
Um, you can't just cast everything to 16 bit because then you, there's not enough precision

01:28:49.120 --> 01:28:51.020
to calculate gradients and stuff properly.

01:28:51.020 --> 01:28:53.580
So we have to use something called Mixed Precision.

01:28:53.580 --> 01:29:03.500
Um, depending on how enthusiastic I'm feeling, I guess we ought to do this from scratch as

01:29:03.500 --> 01:29:04.500
well.

01:29:04.500 --> 01:29:06.660
Um, we'll, we'll see.

01:29:06.660 --> 01:29:11.700
Um, we do have an implementation from scratch cause we actually implemented this before

01:29:11.700 --> 01:29:15.740
NVIDIA implemented it, um, in an earlier version of fastai.

01:29:15.740 --> 01:29:19.900
Um, um, anyway, we'll see.

01:29:19.900 --> 01:29:24.780
So basically the idea is that we use 32 bit for things where we need 32 bit and we use

01:29:24.780 --> 01:29:26.940
16 bit for things where you use 16 bit.

01:29:26.940 --> 01:29:29.940
Um, so that's what we're going to do is we're going to use this mixed precision.

01:29:29.940 --> 01:29:37.460
Um, but for now we're going to use, um, NVIDIA's, you know, semi-automatic or fairly automatic

01:29:37.460 --> 01:29:40.500
code to do that for us.

01:29:40.500 --> 01:29:44.660
Actually we had a slight change of plan at this point when we realized, uh, this lesson

01:29:44.660 --> 01:29:48.740
was going to be over three hours in length and we should actually split it into two.

01:29:48.740 --> 01:29:55.980
So we're going to wrap up this lesson here and we're going to, um, come back and implement

01:29:55.980 --> 01:29:59.820
this mixed precision thing in Lesson 20.

01:29:59.820 --> 01:30:23.780
So we'll see you then.